Numerical Methods with MATLAB for Complex Systems

Auralius Manurung
Mechanical Engineering, Universitas Pertamina

Table of Contents

1. Case 1: One-Dimensional Linear Advection[1]

Given a hyperbolic PDE as follows.

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

with the following initial condition:

$$u(x,0) = u_0(x) \tag{2}$$

where c is a positive real number. Here, we want to to find the solution to (1). When $f(x,t) = 0$, its analytical solution is given by: $u(x,t) = u_0(x-ct)$. Simply put, u_0 propagates along x with a constant speed of c.

This kind of PDE is widely used for real-time modeling and control applications. Therefore, it is essential to investigate its non-homogenous version to allow the implementation of a PDE system further. A PDE system is composed of more than one interacting PDE. For example, there are at least two PDEs in a heat exchanger: 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 but 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.

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

Next, substituting (3) and (4) back to (1) gives us

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

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.

$$u_t \approx \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} \tag{5}$$

While to approach u_x, we use the following approximation.

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

By substituting (5) and (6) to (1), we then have:

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

Let us 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 then have the following shorter terms:

$$\boxed{U_{j,k+1} = U_{j,k} - r(U_{j,k} - U_{j-1,k}) + F_{j,k}\Delta t} \tag{7}$$

MATLAB implementation of the upwind method (7) 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 the efficiency, the listing 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
```

Next, we will implement the PDE in (8) into the MATLAB upwind function above.

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

The implementation is provided in the listing below. The length of the rod 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;

c = 0.5; % Upwind

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

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

figure;
h = plot(0,0);
title('Upwind')
ylim([0 1]);

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

The analytical solution to the PDE on (8) is $u(x,t) = e^{-((x-0.5t)-2)^2}$. However, when executed, the code above will generate a numerical solution as in Figure 1. As we can see, the numerical solution is not exactly the same as its analytical solution. Over time, its amplitude is decreasing.

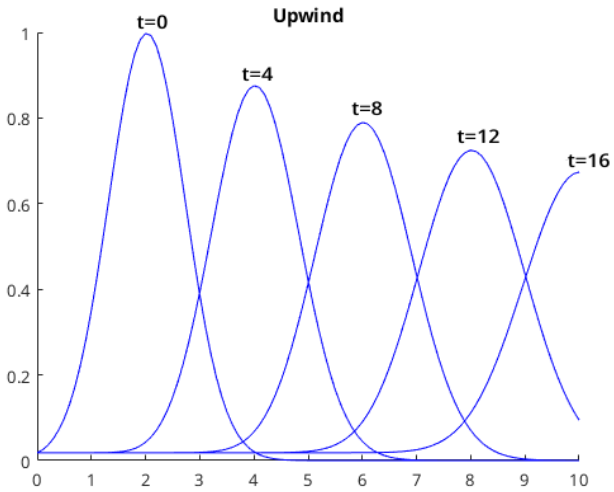


Figure 1: Upwind method for $u_t + 0.5u_{xx} = 0$

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.

$$u_t \approx \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} \tag{9}$$

While to approach u_x, we use the following approximation.

$$u_x \approx \frac{u(x+\Delta x,t)-u(x,t)}{\Delta x} \tag{10}$$

Substituting (9) and (10) back to (1) yields the following.

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

Let us 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 then have the following shorter terms.

$$\boxed{U_{j,k+1} = U_{j,k} - r(U_{j+1,k} - U_{j,k}) + F_{j,k}\Delta t} \tag{11}$$

MATLAB implementation of an upwind method expressed in (11) 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
```

Further, to increase its efficiency, we can turn the MATLAB code above in to vectorized 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
```

To test the listing above, let us solve the following first order linear PDE.

$$\begin{gathered}
u_t - 0.5u_x = 0 \\
u(x,0) = e^{-(x-8)^2} \\
0 \leqslant x \leqslant 10
\end{gathered} \tag{12}$$

The implementation is provided in the listing below. The length of the rod 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;
```

```
4
5   dt = 0.05;
6   T = 20;
7   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 (12) is $u(x,t) = e^{-((x+0.5t)-8)^2}$. However, when executed, the provided listing above generates the a solutoinas in Figure 2. As we can see, the numerical solution is not exactly the same as its analytical solution. Over time, its amplitude is actually decreasing, similar to the upwind cases.

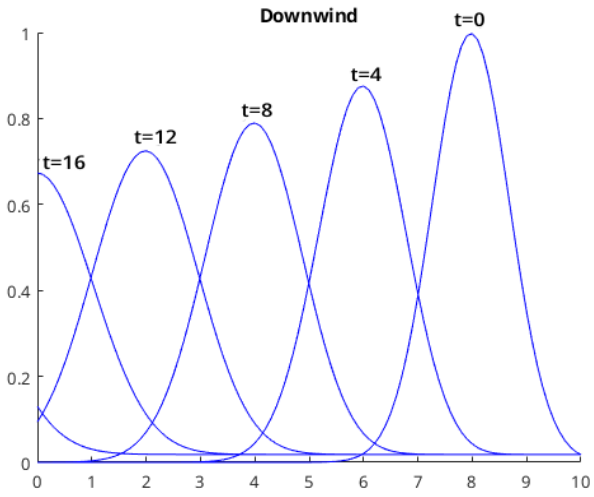


Figure 2: Downwind method for $u_t - 0.5u_{xx} = 0$

1.4. The Lax Method

The Lax method is based on the FTCS method.

$$u_t \approx \frac{u(x,t+\Delta t) - \boxed{u(x,t)}}{\Delta t} \tag{13}$$

However, $u(x,t)$ is replaced by an average of its two neighbors.

$$u(x,t) = \frac{u(x+\Delta x,t)+u(x-\Delta x,t)}{2} \tag{14}$$

Thus, by substituting (14) to (13), the approximation of u_t can be rewritten as

$$u_t \approx \frac{u(x,t+\Delta t) - \left[\frac{u(x+\Delta x,t)+u(x-\Delta x,t)}{2}\right]}{\Delta t} \tag{15}$$

On the other hand, u_x can be approximated as

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

Substituting (15) and (16) back to (1) yields

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

Now, let us consider $r = \frac{c\Delta t}{\Delta x}$, $U_{j,k} = u(x_j,t_k)$, $F_{j,k} = f(x_j,t_k)$, $x_j = j\Delta x$, $t_k = k\Delta t$, $U_{j,k+1} = u(x_j,t_k+\Delta t)$ and $U_{j-1,k} = u(x_j-\Delta x,t_k)$, we can then have the following shorter terms.

$$\boxed{U_{j,k+1} = \frac{U_{j+1,k}+U_{j-1,k}}{2} - \frac{r}{2}(U_{j+1,k}-U_{j,k}) + F_{j,k}\Delta t} \tag{17}$$

The Lax method can handle both upwind and downwind direction. There is no need for us to check the sign of c as in the previous methods. The listing below shows MATLAB implementation of the Lax method.

```
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;

for i = 2:N-1 % i = 1 and i = N are untouched !!!
    next_u_array(i) = 0.5*(u_array(i+1)+u_array(i-1)) - ...
                      0.5*r*(u_array(i+1)-u_array(i-1)) + ...
                      f_array(i)*dt;
end

% 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;

```

```
23 end
```

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

Further, to increase the efficiency of the Lax method, we can turn the MATLAB code above in to vectorized operations 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  next_u_array(2:N-1) = 0.5.*(u_array(3:N)+(u_array(1:N-2))) - ...
9                        (0.5*r).* (u_array(3:N)- u_array(1:N-2)) + ...
10                       f_array(2:N-1).*dt;
11
12
13 % fill up the two ends by using upwind or downwind method
14 if c > 0
15     next_u_array(N) = u_array(N) - r* ...
16                       (u_array(N)- u_array(N-1)) + f_array(N).*dt;
17
18 elseif c < 0
19     next_u_array(1) = u_array(1) - r* ...
20                       (u_array(2) - u_array(1)) + f_array(1).*dt;
21
22 end
```

Next, let us test the code above for both the upwind and the downwind cases. For the upwind case, we will use the PDE in (8). As for the downwind direction, we will use the PDE in (12). For both methods, the length of the rod is divided into 100 segments ($dx = 0.1$) and the time is divided into 400 segments ($dt = 0.05$). The listing below is the implementation of (8) and (12).

```
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 %% ---------------------------------------------------------------
13 c = 0.5; % moving to right
14
15 for k = 1:length(x)
16     u(k) = exp(-10*(x(k)-2).^2);
17 end
18
19 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
```

When executed, the above listing will generate Figure 3 and Figure 4 where we can see the resulting amplitude is also decreasing over time.

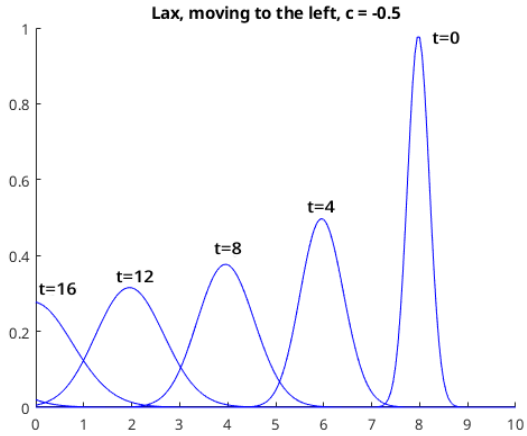


Figure 3: Lax method for $u_t - 0.5u_{xx} = 0$

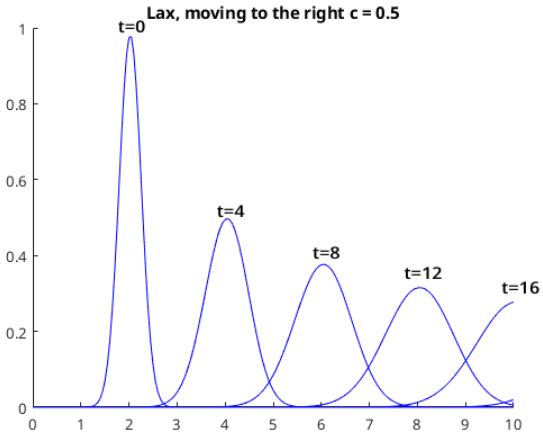


Figure 4: Lax method for $u_t + 0.5u_{xx} = 0$

1.5. The Lax-Wendroff Method[2]

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

$$f(x+\Delta x) = f(x) + \Delta x f'(x) + \frac{\Delta x^2}{2!} f''(x) + \frac{\Delta x^3}{3!} f'''(x) + \ldots \tag{18}$$

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

$$u(x+\Delta x,t) = u(x,t)+\Delta x u_x + \frac{\Delta x^2}{2}u_{xx} \tag{19}$$

Remember that

$$u_t = \frac{u(x,t+\Delta t)-u(x,t)}{\Delta t} \tag{20}$$

and

$$u_x = \frac{u(x+\Delta x,t)-u(x,t)}{\Delta x} \tag{21}$$

Thus, $u_t = -cu_x + f(x,t)$ can be rewritten as

$$\frac{u(x,t+\Delta t)-u(x,t)}{\Delta t} = -c\frac{u(x+\Delta x,t)-u(x,t)}{\Delta x} + f(x,t) \tag{22}$$

By substituting (19) to (22)

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

Since c is the propagation speed of $u(x,t) = u_0(x-ct)$, thus, let us take $\Delta x = -c\Delta t$. As the result, now we have the following equation.

$$\boxed{u(x,t+\Delta t) = \approx u(x,t) - c\Delta t u_x + c^2\frac{\Delta t^2}{2}u_{xx} + f(x,t)\Delta t} \tag{23}$$

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

$$u_x \approx \frac{u(x+\Delta x,t)-u(x-\Delta x,t)}{2\Delta x} \tag{24}$$

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.

$$\begin{aligned}
u_{xx} &\approx \frac{u_x(x+\Delta x,t)-u_x(x,t)}{\Delta x}\\
&\approx \frac{\frac{u(x+\Delta x,t)-u(x,t)}{\Delta x} - \frac{u(x,t)-u(x-\Delta x,t)}{\Delta x}}{\Delta x}\\
&\approx \frac{u(x+\Delta x,t)-2u(x,t)+u(x-\Delta x,t)}{\Delta x^2}
\end{aligned} \tag{25}$$

Now that we have (24) and (25), we can substitute them back to (23):

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

$$\approx u(x,t) - \frac{c\Delta t}{2\Delta x}[u(x+\Delta x,t) - u(x-\Delta x,t)] + \frac{c^2\Delta t^2}{2\Delta x^2}[u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)] + f(x,t)\Delta t$$

Now let us consider $r = \frac{c\Delta t}{\Delta x}$, $U_{j,k} = u(x_j, t_k)$, $F_{j,k} = f(x_j, t_k)$, $x_j = j\Delta x$, $t_k = k\Delta t$, $U_{j,k+1} = u(x_j, t_k + \Delta t)$ and $U_{j-1,k} = u(x_j - \Delta x, t_k)$, we can then have the following shorter terms.

$$\boxed{U_{j,k+1} = U_{j,k} - \frac{r}{2}(U_{j+1,k} - U_{j-1,k}) + \frac{r^2}{2}(U_{j+1,k} - 2U_{j,k} + U_{j-1,k}) + F_{j,k}\Delta t} \quad (26)$$

MATLAB implementation of (26) is as follows (see line 6 to 11).

```
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  for i = 2:N-1
8      next_u_array(i) = u_array(i)-0.5*r*(u_array(i+1)-u_array(i-1)) ...
9                        + 0.5*r^2 * (u_array(i+1)-2*u_array(i)+u_array(i-1)) ...
10                       + f_array(i)*dt;
11 end
12
13 % fill up the two ends by using upwind or downwind method
14 if c > 0
15     next_u_array(N) = u_array(N) - r* ...
16                      (u_array(N)- u_array(N-1)) + f_array(N)*dt;
17
18 elseif c < 0
19     next_u_array(1) = u_array(1) - r* ...
20                       (u_array(2) - u_array(1)) + f_array(1)*dt;
21
22 end
```

Here, we encounter the same issue as in the previous 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 (see line 7 of the listing above). To address this issue, we apply upwind and downwind method for the two ends only (see line 14 to 20 of the listing above).

Further, to increase its efficiency, we can turn the MATLAB code above in to vectorized 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
```

```
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, let us test the code listings above for both the upwind and the downwind cases. For the upwind case, we will use the same system as in (8) As for the downwind direction, we will also use the same system as in (12). We will divide the length of the rod into 200 segments $(dx = 0.05)$ and the time into 200 segments as well $(dt = 0.05)$. See the listing below.

```
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

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);
```

```
45      drawnow
46  end
```

When executed, the listing above will generate the Figure 5 and Figure 6. As shown by those two figures, the, Lax-Wendroff method does provide more stable solutions. Even tough the amplitudes are still decreasing, the decrease in the amplitude is much smaller than other methods that have been previously discussed.

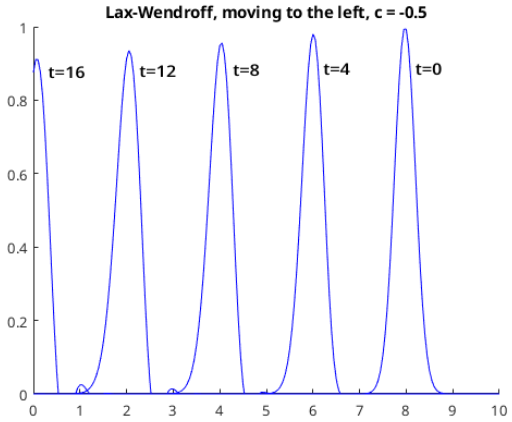


Figure 5: Lax method for $u_t - 0.5u_{xx} = 0$

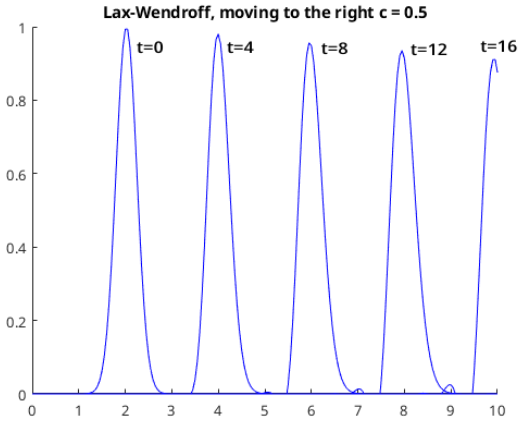


Figure 6: Lax method for $u_t + 0.5u_{xx} = 0$

1.6. Cross Current Heat Exchanger as An Example[3]

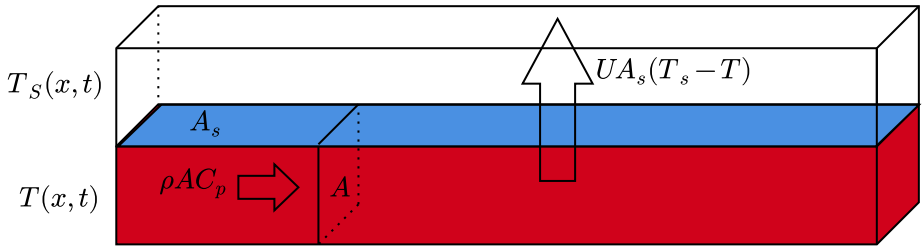
A heat exchanger is modeled with two or more interacting non-homogenous PDEs. The simplest model involves two PDEs only: the hot flow and the cold flow. Yet another simple method, three PDEs are involved: the hot flow, the cold flow, and the wall between them.

1.6.1. Two-Equation Model

Flowing liquid inside a container can be modeled as an advective PDE. When external environment is involved, the PDE becomes non-homogenous (see Drawing 1). Interaction with the environment is modeled with Fourier's Law. The overall process can be modeled as follows.

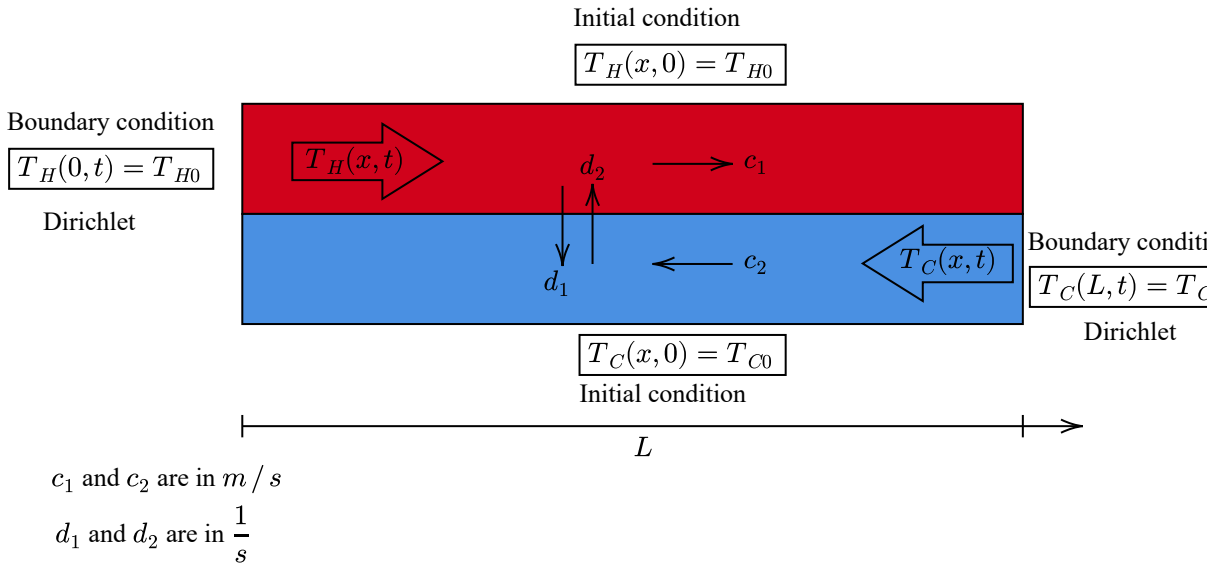
$$\underbrace{\frac{\partial T(x,t)}{\partial t} + v\frac{\partial T(x,t)}{\partial z}}_{\text{Advection}} = \underbrace{-\frac{UA_S}{\rho AC_p}(T(x,t) - T_S(x,t))}_{\text{Fourier's law}} \tag{27}$$

where
C_p is the liquid specific heat capacity
A is cross sectional are of the tube where the liquid is flowing
A_S is the heat transfer area (blue surface)
U is the overall heat transfer coefficient between the tube to its environment
ρ is the density of the liquid



Drawing 1: Flowing liquid in red container while transferring heat to its environment

A diagram of a simplest heat exchanger is presented in Drawing 2. From Drawing 2, we can see that hot liquid flows from left to right and cold liquid flows from right to left. Source temperature of the hot flow is kept at T_{H0}. While source temperature of the cold flow is kept at T_{C0}. Those two temperatures never change. This condition is called Dirichlet condition.



Drawing 2: Cross-current heat exchanger modeled with two PDEs

The system model of a heat exchanger based on Drawing 2 can be described as follows.

$$\begin{aligned} \frac{\partial}{\partial t}T_H(x,t) + c_1\frac{\partial}{\partial x}T_H(x,t) &= -d_1(T_H(x,t) - T_C(x,t)) \\ \frac{\partial}{\partial t}T_C(x,t) - c_2\frac{\partial}{\partial x}T_C(x,t) &= -d_2(T_C(x,t) - T_H(x,t)) \end{aligned} \tag{28}$$

or

$$\begin{aligned} \frac{\partial}{\partial t}T_H(x,t) &= -c_1\frac{\partial}{\partial x}T_H(x,t) - d_1(T_H(x,t) - T_C(x,t)) \\ \frac{\partial}{\partial t}T_C(x,t) &= c_2\frac{\partial}{\partial x}T_C(x,t) + d_2(T_H(x,t) - T_C(x,t)) \end{aligned} \tag{29}$$

where:

- d_1 and d_2 describe **convective** heat transfer **between** the the two flowing liquids: **from** hot liquid **to** cold liquid and **from** cold liquid **to** hot liquid, respectively.
- c_1 and c_2 describes **advective** heat transfer **inside** the the hot and cold liquid, respectively. This is

analogous to the actual movement of the liquids since the heat moves together with the liquids.

For the simulation, we will set the parameters according to the paper by Zobiri et.al.[4] In their paper, it is mentioned that they use upwind-downwind method. However, we will solve the system above by using both the upwind-downwind method and the Lax-Wendroff method so that we can compare their results. The listing below shows the implementation of a cross-current heat exchanger as in (28) which runs for $20s$ with $0.01s$ step-time.

```
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;
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
% ========================================================================
```

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

```
50 TH = zeros(length(x), 1);
51 TC = zeros(length(x), 1);
52 f = zeros(length(x), 1);
53
54 % Intial conditoin
55 TH(1:length(x)) = 273+30;
56 TC(1:length(x)) = 273+10;
57
58 figure;
59 hold on
60 h3 = plot(0,0,'r');
61 h4 = plot(0,0,'b');
62 title('Cross-current heat exchanger');
63 ylim([270 320]);
64 xlabel('x');
65 ylabel('Temperature');
66 legend('Hot flow','Cold flow');
67
68 for k = 1:length(t)
69     set(h3,'XData',x,'Ydata',TH);
70     set(h4,'XData',x,'Ydata',TC);
71     drawnow;
72
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 of the listing above, we use the upwind method for the hot flow since the flow moves from left to right. As for the cold flow, we use the downwind method since the cold flow moves from right to left. As for the Lax-Wendroff method, there is only one function that we call (line 73 to 76). However, we still need to call that function twice, one for the cold flow, another one is for the hot flow. When executed, the provided lisiting above will generate animations of the temperature evolutions for both methods.

Figure 7 and Figure 8 below show the temperature distribution of the modeled heat exchanger after 20 seconds for two different methods. From Figure 8, 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 with a practical perspective, the upwind-and downwind method might be more desirable since a theoretically perfect advection might be impossible in a real world.

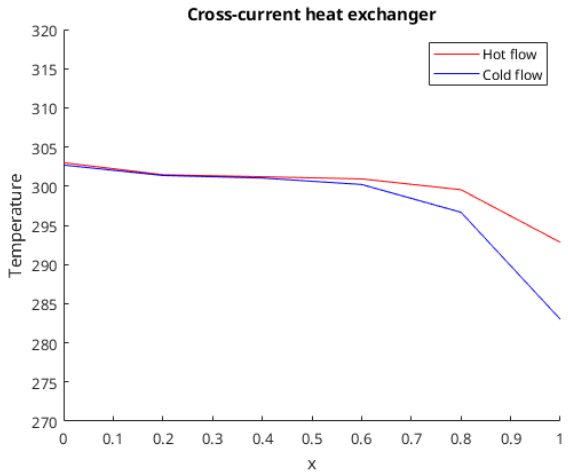


Figure 7: After 20 s, with upwind-downwind method

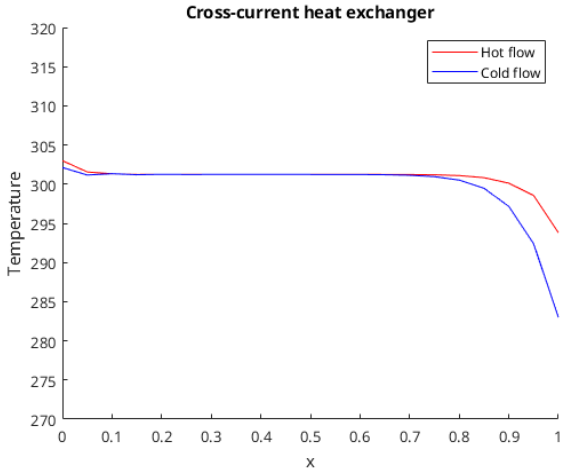
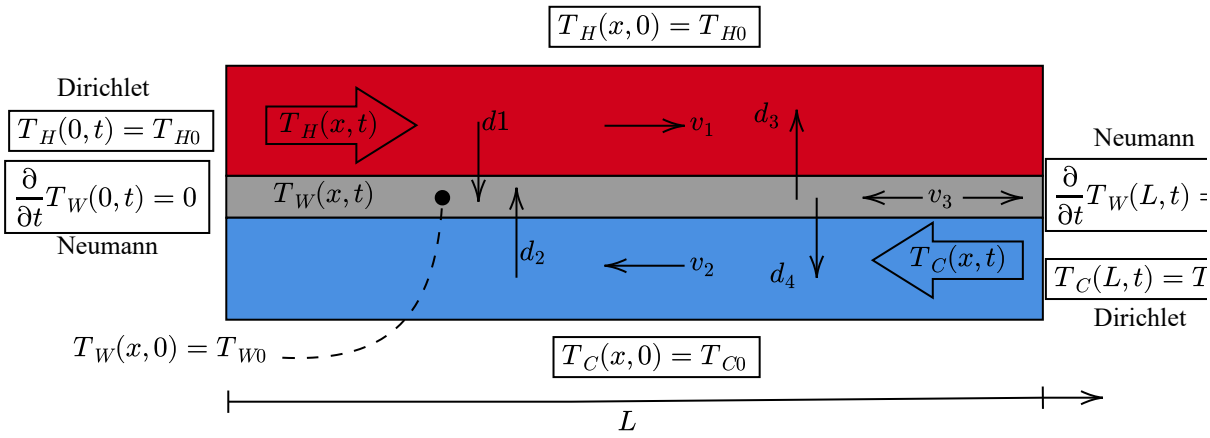


Figure 8: After 20 s, with Lax-Wendroff method

1.6.2. Three-Equation Model



Drawing 3: Cross-current heat exchanger modeled with three PDEs

In three-equation model (see Drawing 3), 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 **from** the the liquid **to** the wall: from the hot liquid to the wall and from the cold liquid to the wall, respectively.
- d_3 and d_4 describe **convective** heat transfer **from** the the wall **to** the liquid: from the wall to the hot liquid and from the wall to the cold liquid, respectively.
- v_1 and v_2 describes **advective** heat transfer **inside** the the hot and cold liquid, respectively. This is analogous to the actual movement of the liquids since the heat moves together with the liquids.
- v_3 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.

$$\begin{aligned}
\frac{\partial}{\partial t}T_H(x,t) &= -v_1\frac{\partial}{\partial x}T_H(x,t) - d_1(T_H(x,t) - T_W(x,t)) \\
\frac{\partial}{\partial t}T_C(x,t) &= v_2\frac{\partial}{\partial x}T_C(x,t) + d_2(T_W(x,t) - T_C(x,t)) \\
\frac{\partial}{\partial t}T_W(x,t) &= v_3\frac{\partial^2}{\partial x^2}T_W(x,t) + d_3(T_H(x,t) - T_W(x,t)) - d_4(T_W(x,t) - T_C(x,t))
\end{aligned} \tag{30}$$

Where[5]:

$$\begin{aligned} v_1 &= 0.6366197724 \\ v_2 &= 0.1657863990 \\ v_3 &= 0.00003585526318 \end{aligned} \qquad \begin{aligned} d_1 &= 0.9792 \\ d_2 &= 0.2986 \\ d_3 &= 2.3923 \\ d_4 &= 2.8708 \end{aligned}$$

The initial and boundary conditions are:
$T_H(0,t) = 360$ and $T_H(x,0) = 360$
$T_C(L,t) = 300$ and $T_C(x,0) = 300$
$T_W(x,0) = 330, \dfrac{\partial T_W(0,t)}{\partial x} = 0$, and $\dfrac{\partial T_W(L,t)}{\partial x} = 0$

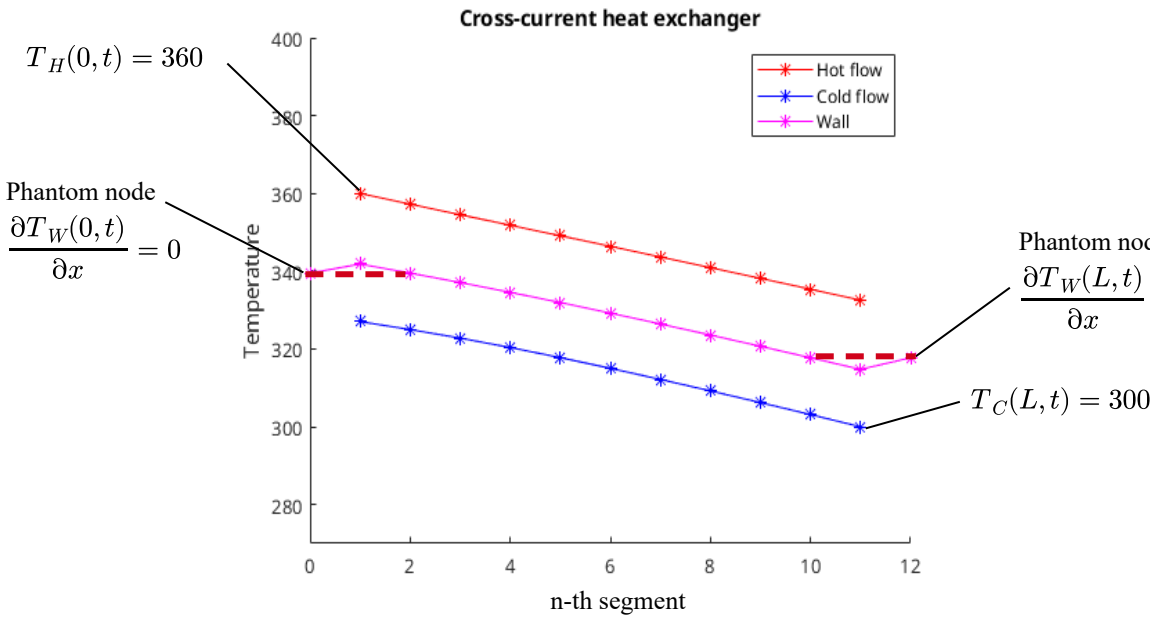
In (30), the third equation is different than the first two equations. The third equation is a non-homogenous one-dimensional heat equation (diffusion) with Neumann boundary conditions. We will discuss this in the next section. Nevertheless, we will directly show the implementation of the three-equation model of a heat exchanger in MATLAB. For the first and second equations, we will use upwind and downwind method, respectively (see line 51 to 55). The rest of the code can be understood after reading the next section.

```
1  L = 1;
2  tF = 5;
3  Nx = 10;
4  Nt = 100;
5  dt = tF/Nt;
6  dx = L/Nx;
7
8  v1 = 0.6366197724;
9  v2 = 0.1657863990;
10 v3 = 0.00003585526318;
11 d1 = 0.9792;
12 d2 = 0.2986;
13 d3 = 2.3923;
14 d4 = 2.8708;
15
16 TH = zeros(1, Nx+1);
17 TC = zeros(1, Nx+1);
18 TW = zeros(1, Nx+1+2);  % +2 phantom nodes
19 fH = zeros(1, Nx+1);
20 fC = zeros(1, Nx+1);
21 fW = zeros(1, Nx+1+2);  % +2 phantom nodes
22
23 % Intial condition
24 TH(1:Nx+1) = 360;
25 TC(1:Nx+1) = 300;
26 TW(1:Nx+1+2) = 330;
27
28 h=figure;
29 hold on
```

[5] https://www.maplesoft.com/applications/view.aspx?SID=119402&view=html

```
h1 = plot(0,0,'r-*');
h2 = plot(0,0,'b-*');
h3 = plot(0,0,'m-*');
title('Cross-current heat exchanger')
ylim([270 400])
xlabel('x')
ylabel('Temperature')
legend('Hot flow', 'Cold flow', 'Wall')

xa = 1:Nx+1;
xb = 0:Nx+2;
for k = 1 : Nt
    set(h1,'XData',xa,'Ydata',TH);
    set(h2,'XData',xa,'Ydata',TC);
    set(h3,'XData',xb,'Ydata',TW);
    drawnow;

    TW = apply_bc(TW, L/Nx, ["Neumann", "Neumann"], [0, 0]);

    % TW(2:end-1) : it means we ignore phantom nodes

    fH = -d1.*(TH-TW(2:end-1));
    TH = upwind(TH, fH, v1, dt, dx);

    fC = d2.*(TH-TW(2:end-1));
    TC = downwind(TC, fC, -v2, dt, dx);

    fW(2:end-1) = d3.*(TH-TW(2:end-1)) - d4.*(TW(2:end-1)-TC);
    TW = diffusion_1d(TW, fW, sqrt(v3), dx, dt);
end
```

When executed for 20 seconds, temperature distributions of different parts of the heat exchanger are shown in Drawing 4.



Drawing 4: Temperature distributions ofter 5 seconds

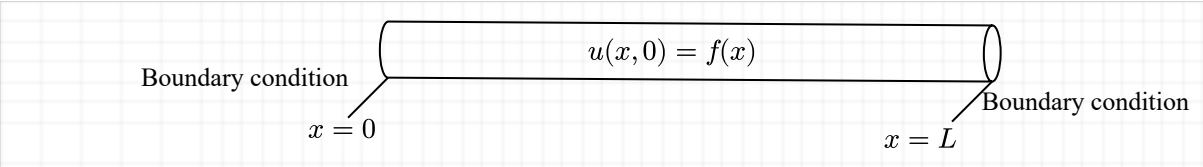
2. Case 2: One-Dimensional Heat Equation[6]

The equation of temperature distribution (u) in a one-dimensional rod (see Drawing 5) over a certain distance (x) and time (t) is given by (31). α is the diffusivity coefficient. When $g(x,t)=0$, the system is homogenous.

$$\begin{gathered}\alpha^2 u_{xx}-u_t=g(x,t)\\ t\geqslant 0 \text{ and } 0\leqslant x\leqslant L\end{gathered} \tag{31}$$

when at $t=0$, the following initial condition is applied.

$$u(x,0)=f(x)$$



Drawing 5: Temperature distribution (u) in a very thin rod

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.

$$u_t\approx\frac{u(x,t+\Delta t)-u(x,t)}{\Delta t} \tag{32}$$

$$u_x\approx\frac{u(x+\Delta x,t)-u(x,t)}{\Delta x} \tag{33}$$

While to approach u_x, we use the following approximation.

$$\begin{aligned}u_{xx}&\approx\frac{u_x(x+\Delta x,t)-u_x(x,t)}{\Delta x}\\ &\approx\frac{\frac{u(x+\Delta x,t)-u(x,t)}{\Delta x}-\frac{u(x,t)-u(x-\Delta x,t)}{\Delta x}}{\Delta x}\\ &\approx\frac{u(x+\Delta x,t)-2u(x,t)+u(x-\Delta x,t)}{(\Delta x)^2}\end{aligned} \tag{34}$$

Substituting (32) and (33) back to (31) gives us the following equations.

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

where $r=\dfrac{\alpha^2\Delta t}{(\Delta x)^2}$. Now, let us consider $U_{j,k}=u(x_j,t_k)$, $G_{j,k}=g(x_j,t_k)$, $x_j=j\Delta x$, $t_k=k\Delta t$, $U_{j,k+1}=u(x_j,t_k+\Delta t)$ and $U_{j-1,k}=u(x_j-\Delta x,t_k)$, we can then have the following shorter terms.

$$\boxed{U_{j,k+1} = U_{j,k} + r\left[U_{j+1,k} - 2U_{j,k} + U_{j-1,k}\right] + G_{j,k}\Delta t} \tag{35}$$

2.1.1. Dirichlet Boundary Conditions

In (31), certain conditions are applied and kept true at all time. These conditions are called boundary conditions. There are several types of boundary conditions. One of them is Dirichlet boundary condition. A typical Dirichlet boundary condition would be

$$u(0,t) = A$$
$$u(L,t) = B$$

where A and B are real numbers. The implementation of the Dirichlet boundary condition is very straightforward by directly controlling values at the two end nodes.

2.1.2. Neumann Boundary Condition

Another type of boundary condition is the Neumann boundary condition. In Neumann boundary condition, the derivative of the solution function $\left(\partial u(x,t)/\partial x \text{ or } \partial u(x,t)/\partial t\right)$ is kept constant. However, here we will only implement the derivative of the solution with respect to x.

$$\frac{\partial u(x,t)}{\partial x} = C \tag{36}$$

where C is a real number. At the boundary when $x = 0$, we can write

$$\frac{u(\Delta x,t) - u(-\Delta x,t)}{2\Delta x} = C \tag{37}$$

therefore

$$u(-\Delta x,t) = u(\Delta x,t) - 2\Delta x C \tag{38}$$

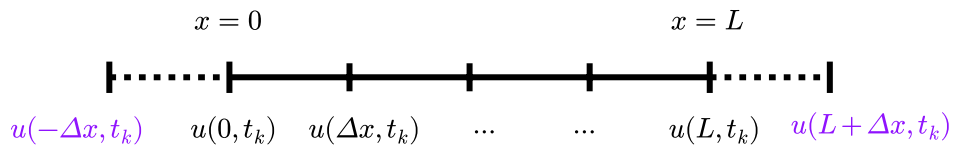
At another boundary when $x = L$

$$\frac{u(L+\Delta x,t) - u(L-\Delta x,t)}{2\Delta x} = D \tag{39}$$

where D is a real number. Therefore

$$u(L+\Delta x,t) = u(L-\Delta x,t) + 2\Delta x D \tag{40}$$

The implementation of Neumann boundary condition will be done by adding additional phantom nodes as shown in Drawing 6.



Drawing 6: Extra phantom nodes for Neumann boundary condition

2.1.3. MATLAB Implementation

MATLAB implementation of (35) is as follows.

```
% alpha^2 * u_xx = u_tt
% Divide 0<=x<=L with Delta_x increments
% Divide 0<=t<=tF with Delta_t increments

```

```
5  function u_array = diffusion_1d(u_array, g_array, alpha, Delta_x, Delta_t)
6
7  r = alpha^2 * Delta_t / Delta_x^2;
8
9  j = 2 : length(u_array)-1;
10 u_array(j) = ...
11     u_array(j) + r*(u_array(j+1) -2 * u_array(j) + u_array(j-1)) + ...
12     g_array(j)*Delta_t;
13 end
```

The implementation above is already vectorized. In line 9 of the listing above, we can see that the diffusion is not applied to the first and last nodes. At those two end-nodes, we will apply the boundary conditions. Therefore, we also create two more functions for the initial and boundary conditions (see the listing below).

```
1  % Initial condition
2  function u_array = apply_ic(u_array, f, Delta_x)
3
4  % Apply the initial condition U(x,0)=f(x)
5  j = 1:length(u_array);
6  u_array(j) = f((j-1)*Delta_x);
7
8  end
9
10 %============================================================
11
12 % Boundary condition
13 function u_array = apply_bc(u_array, Delta_x, types, values)
14
15 % left-end
16 if lower(types(1)) == "dirichlet"
17     u_array(1) = values(1);
18 elseif lower(types(1)) == "neumann"
19     u_array(1) = u_array(3) - 2*Delta_x * values(1);
20 end
21
22 % right-end
23 if lower(types(2)) == "dirichlet"
24     u_array(end) = values(2);
25 elseif lower(types(2)) == "neumann"
26     u_array(end) = u_array(end-2) + 2 * Delta_x * values(2);
27 end
28 end
```

In the Neumann boundary condition, we need to add additional phantom nodes. There are some examples provided in the next section to understand how to use the aforementioned three functions.

2.1.4. Example 1

Given the following heat transfer equation:

$$1.14u_{xx} - u_t = 0,\ 0 \leqslant x \leqslant 10 \tag{41}$$

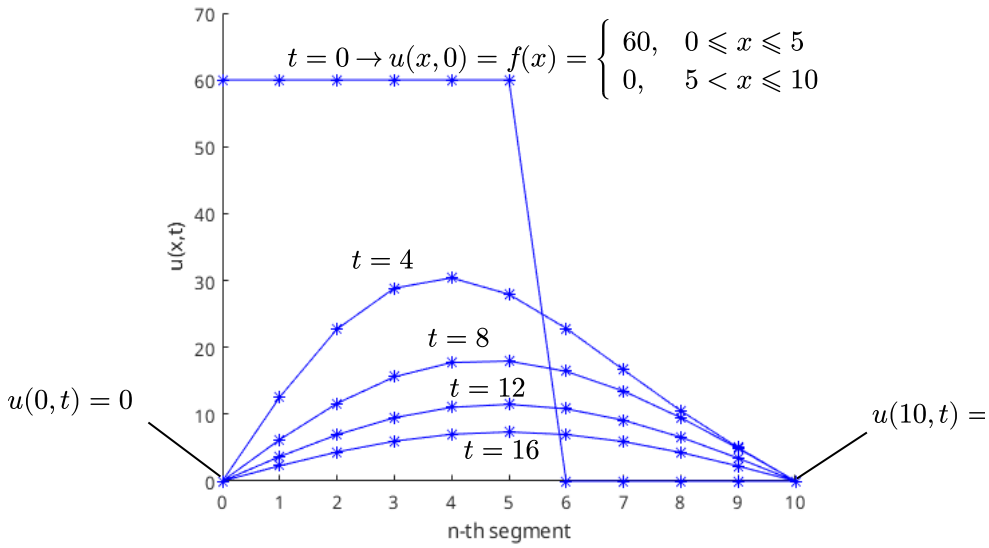
with the following boundary and initial conditions:

$$\begin{aligned} u(0,t) &= u(10,t) = 0 \\ u(x,0) = f(x) &= \begin{cases} 60, & 0 \leqslant x \leqslant 5 \\ 0, & 5 < x \leqslant 10 \end{cases} \end{aligned} \tag{42}$$

MATLAB implementation of (41) is as follows.

```
L = 10;
alpha = sqrt(1.14);

tF = 20;
Nx = 10;
Nt = 1000;

u_array = zeros(1,Nx+1); % +1 because we start from 0 to Nx
g_array = zeros(1,Nx+1); % homogenous
u_array = apply_ic(u_array,@f, L/Nx);

figure
hold on
h = plot(0,0,'-*');
ylim([0 70])

x = 0 : Nx;

for k = 1 : Nt
    set(h, 'XData', x, 'YData', u_array)
    drawnow;

    u_array = apply_bc(u_array, L/Nx, ["Dirichlet", "Dirichlet"], [0, 0]);
    u_array = diffusion_1d(u_array, g_array, alpha, L/Nx, tF/Nt);
end
xlabel('n-th segment')
ylabel('u(x,t)')

%% Intial Condition
function u=f(x)
L = 10;
u = zeros(1,length(x));

for k = 1 : length(x)
    if x(k) >= 0 && x(k) <=L/2
        u(k) = 60;
    elseif x(k) > L/2 && x(k) <= L
        u(k) = 0;
    end
end

end
```

The result is shown in Drawing 7 below.



Drawing 7: Simulation result: Dirichlet-Dirichlet

2.1.5. Example 2

Given the following heat transfer equation:

$$u_{xx} - u_t = 0, \ 0 \leqslant x \leqslant \pi \tag{43}$$

with the following boundary and initial conditions:

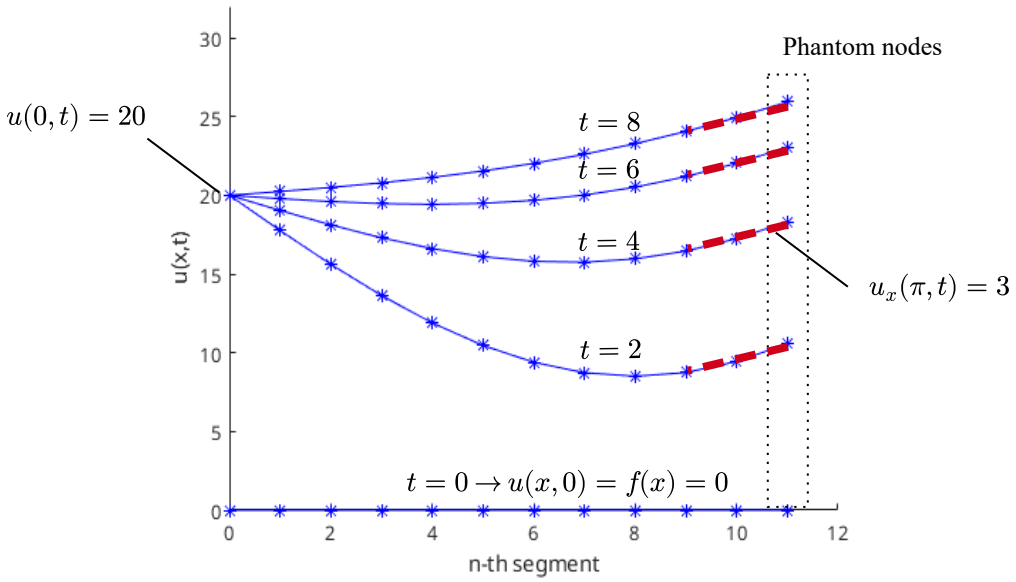
$$\begin{aligned} u(0,t) &= 20 \\ u_x(\pi,t) &= 3 \\ u(x,0) &= 0 \end{aligned} \tag{44}$$

MATLAB implementation of (43) is as follows.

```
L = pi;
alpha = 1;

tF = 10;
Nx = 10;
Nt = 1000;

u_array = zeros(1,Nx+1+1); % +1 -> start from 0 to Nx
                           % another +1 -> add one phantom nodes
g_array = zeros(1,length(u_array));
u_array = apply_ic(u_array,@f, L/Nx);

figure
hold on
h = plot(0,0,'-*');
ylim([0 32])

x = 0 : Nx+1;

for k = 1 : Nt
    set(h, 'XData', x, 'YData', u_array)
    drawnow;

    u_array = apply_bc(u_array, L/Nx, ["Dirichlet", "Neumann"], [20, 3]);
```

```
    u_array = diffusion_1d(u_array, g_array, alpha, L/Nx, tF/Nt);
end
xlabel('n-th segment')
ylabel('u(x,t)')

%% Intial Condition
function u=f(x)
    u = zeros(1,length(x));
end
```

The result is shown in Drawing 8 below.



Drawing 8: Simulation result: Dirichlet-Neumann

2.1.6. Example 3

Given the following heat equation:

$$u_{xx} - u_t = 0,\ 0 \leqslant x \leqslant 1 \tag{45}$$

with the following initial and boundary conditions:

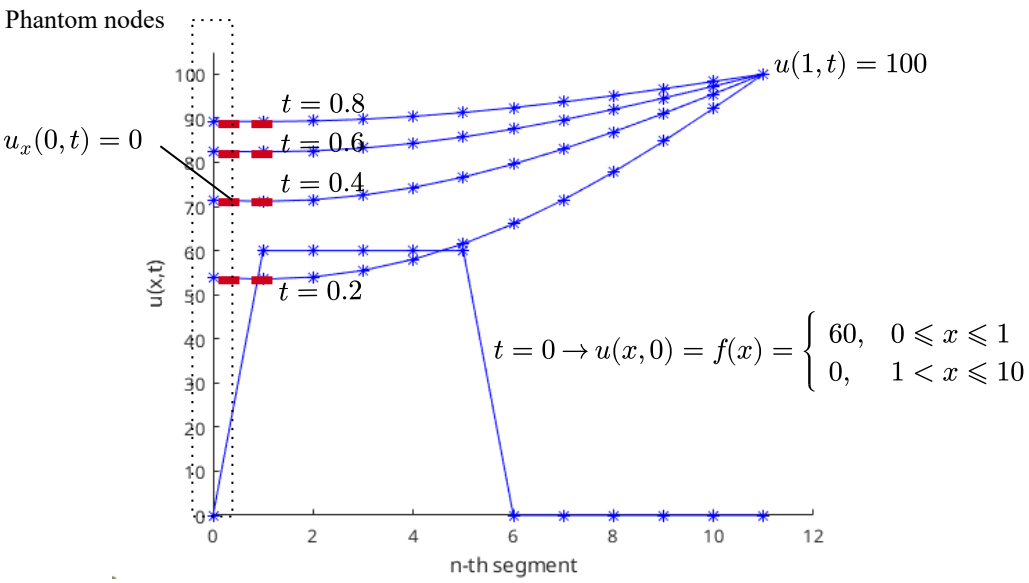
$$\begin{gathered} u_x(0,t) = 0 \\ u(1,t) = 100 \\ u(x,0) = f(x) = \begin{cases} 60, & 0 \leqslant x \leqslant 1/2 \\ 0, & 1/2 < x \leqslant 1 \end{cases} \end{gathered} \tag{46}$$

MATLAB implementation of (45) is as follows.

```
L = 1;
alpha = 1;

tF = 1;
Nx = 10;
Nt = 1000;

u_array = zeros(1,Nx+1+1); % +1 because we start from 0 to Nx
                           % another +1 -> add one phantom nodes
g_array = zeros(1,length(u_array));
```

```
u_array = apply_ic(u_array,@f, L/Nx);

hf = figure;
hold on
h = plot(0,0,'-*');
ylim([0 105])
xlabel('n-th segment')
ylabel('u(x,t)')

x = 0 : Nx+1;

for k = 1 : Nt
    set(h, 'XData', x, 'YData', u_array)
    drawnow;

    u_array = apply_bc(u_array, L/Nx, ["Neumann", "Dirichlet"], [0, 100]);
    u_array = diffusion_1d(u_array, g_array, alpha, L/Nx, tF/Nt);
end

%% Initial condition
function u=f(x)
L = 1;
u = zeros(1,length(x));

for k = 1 : length(x)
    if x(k) >= 0 && x(k) <=L/2
        u(k) = 60;
    elseif x(k) > L/2 && x(k) <= L
        u(k) = 0;
    end
end

end
```

The result is shown in Drawing 9 below.



Drawing 9: Simulation result: Neumann-Dirichlet

2.2. Implicit Method

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

$$u_{xx}(x,t) = (1-\theta)u_{xx}(x,t) + \theta u_{xx}(x,t+\Delta t),\ 0 \leqslant \theta \leqslant 1 \tag{47}$$

where θ acts as a weighting factor. When $\theta = 0$, (47) becomes similar to (35). What we currently have in this case is an explicit method. On the other hand, when $\theta = 1$, in this case, what we have is an implicit method. However, when $\theta = 0.5$, the method is called Crank-Nicolson method. In this section, let us focus on mathematical derivations of the Crank-Nicolson method.

Expanding (47) bu first setting $\theta = \dfrac{1}{2}$ gives us:

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

which can be shortened as

$$u_{xx} = \frac{U_{j+1,k}-2U_{j,k}+U_{j-1,k}+U_{j+1,k+1}-2U_{j,k+1}+U_{j-1,k+1}}{2\Delta x^2} \tag{48}$$

Substituting (48) back to: $u_t = \alpha^2 u_{xx}$ gives us:

$$\begin{aligned} \underbrace{\frac{U_{j,k+1}-U_{j,k}}{\Delta t}}_{u_t} &= \alpha^2 \underbrace{\frac{U_{j+1,k}-2U_{j,k}+U_{j-1,k}+U_{j+1,k+1}-2U_{j,k+1}+U_{j-1,k+1}}{2\Delta x^2}}_{u_{xx}} \\ U_{j,k+1}-U_{j,k} &= \frac{1}{2}\frac{\alpha^2\Delta t}{\Delta x^2}(U_{j+1,k}-2U_{j,k}+U_{j-1,k}+U_{j+1,k+1}-2U_{j,k+1}+U_{j-1,k+1}) \end{aligned} \tag{49}$$

Since $r = \dfrac{\alpha^2\Delta t}{\Delta x^2}$, (49) can then be simplified as:

$$\begin{aligned} U_{j,k+1}-U_{j,k} &= \frac{r}{2}(U_{j+1,k}-2U_{j,k}+U_{j-1,k}+U_{j+1,k+1}-2U_{j,k+1}+U_{j-1,k+1}) \\ U_{j,k+1}+rU_{j,k+1} &= \frac{r}{2}U_{j+1,k}+(1-r)U_{j,k}+\frac{r}{2}U_{j-1,k}+\frac{r}{2}U_{j+1,k+1}+\frac{r}{2}U_{j-1,k+1} \\ 2(1+r)U_{j,k+1} &= rU_{j+1,k}+2(1-r)U_{j,k}+rU_{j-1,k}+rU_{j+1,k+1}+rU_{j-1,k+1} \\ -rU_{j-1,k+1}+2(1+r)U_{j,k+1}-rU_{j+1,k+1} &= rU_{j-1,k}+2(1-r)U_{j,k}+rU_{j+1,k} \end{aligned} \tag{50}$$

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

$$\boxed{-rU_{j-1,k+1}+2(1+r)U_{j,k+1}-rU_{j+1,k+1} = rU_{j-1,k}+2(1-r)U_{j,k}+rU_{j+1,k}} \tag{51}$$

or in matrix form:

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

$$\underbrace{\begin{bmatrix} r & 2(1-r) & r & & & \cdots & 0 \\ 0 & r & 2(1-r) & r & & & \vdots \\ & & r & 2(1-r) & r & & \\ & & & \ddots & & & \\ \vdots & & & & & r & 2(1-r) \\ 0 & \cdots & & & & 0 & r \end{bmatrix}}_{B} \begin{bmatrix} p(k\Delta t) \\ U_{1,k} \\ U_{2,k} \\ \vdots \\ U_{N-2,k} \\ q(k\Delta t) \end{bmatrix} \tag{52}$$

Red means the values are known. In (52), 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.

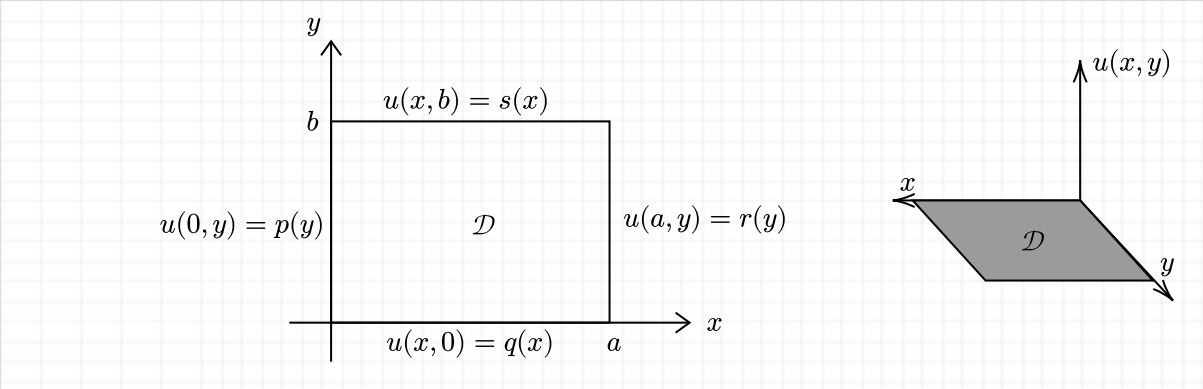
$$\underbrace{\begin{bmatrix} 2(1+r) & -r & & \cdots & 0 \\ -r & 2(1+r) & -r & & \vdots \\ & & \ddots & & \\ & & -r & 2(1+r) & -r \\ \cdots & & & -r & 2(1+r) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} U_{1,k+1} \\ U_{2,k+1} \\ \vdots \\ U_{N-2,k+1} \\ U_{N-1,k+1} \end{bmatrix}}_{U_{new}} =$$

$$\underbrace{\underbrace{\begin{bmatrix} 2(1-r) & r & & \cdots & \cdots \\ r & 2(1-r) & r & & \\ & \ddots & & & \\ & & r & 2(1-r) & r \\ \cdots & & & r & 2(1-r) \end{bmatrix}}_{b} \underbrace{\begin{bmatrix} U_{1,k} \\ U_{2,k} \\ \vdots \\ U_{N-2,k} \\ U_{N-1,k} \end{bmatrix}}_{U_{now}} + \underbrace{\begin{bmatrix} rp((k+1)\Delta t) + rp(k\Delta t) \\ 0 \\ \vdots \\ 0 \\ rq((k+1)\Delta t) + rq(k\Delta t) \end{bmatrix}}_{c}}_{B=(b\cdot U_{now})+c} \tag{53}$$

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. Two-Dimensional Steady Diffusion[7]

3.1. Dirichlet Boundary



Drawing 10: Heat distribution on a two-dimensional plane with Dirichlet boundary conditions

Heat transfer on a two-dimensional plane (see Drawing 10) is described as follows:

$$\begin{gathered} u_{xx}+u_{yy}=f(x,y) \\ 0\le x\le a, 0\le y\le b \end{gathered} \tag{54}$$

The PDE in (54) is known as the Poisson equation. Its homogenous form is known as the Laplace equation.

Finite difference method for the system above is as follows. First, u_{xx} and u_{yy} can be approximated as:

$$\begin{aligned} u_{xx}=\frac{\partial^2 u}{\partial x^2}&\approx\frac{u(x-\Delta x,y)-2u(x,y)+u(x+\Delta x,y)}{\Delta x^2} \\ u_{yy}=\frac{\partial^2 u}{\partial y^2}&\approx\frac{u(x,y-\Delta y)-2u(x,y)+u(x,y+\Delta y)}{\Delta y^2} \end{aligned} \tag{55}$$

Hence:

$$\begin{aligned} u_{xx}+u_{yy}&=f(x,y) \\ \frac{u(x-\Delta x,y)-2u(x,y)+u(x+\Delta x,y)}{\Delta x^2}+\frac{u(x,y-\Delta y)-2u(x,y)+u(x,y+\Delta y)}{\Delta y^2}&=f(x,y) \end{aligned} \tag{56}$$

For simplification purpose, let us take $\Delta x=\Delta x=h$, therefore:

$$\begin{aligned} u_{xx}+u_{yy}&=f(x,y) \\ \frac{u(x-h,y)-2u(x,y)+u(x+h,y)}{h^2}+\frac{u(x,y-h)-2u(x,y)+u(x,y+h)}{h^2}&=f(x,y) \end{aligned} \tag{57}$$

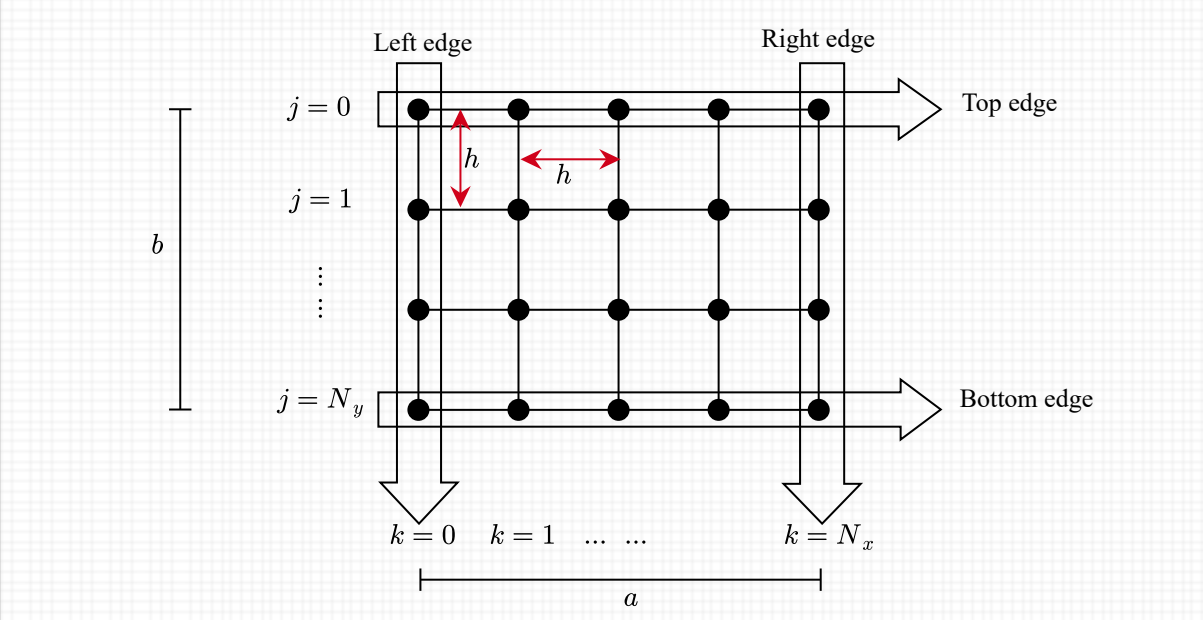
Let us take $U_{j,k}=u(x_j,y_k)$, $F_{j,k}=f(x_j,y_k)$, $x_j=jh$, $y_k=kh$, $U_{j,k+1}=u(x_j,y_k+h)$ and $U_{j-1,k}=u(x_j-h,y_k)$, hence:

$$\begin{aligned} \frac{u(x-h,y)-2u(x,y)+u(x+h,y)}{h^2}+\frac{u(x,y-h)-2u(x,y)+u(x,y+h)}{h^2}&=f(x,y) \\ U_{j-1,k}-2U_{j,k}+U_{j+1,k}+U_{j,k-1}-2U_{j,k}+U_{j,k+1}&=h^2F_{j,k} \\ U_{j-1,k}+U_{j+1,k}+U_{j,k-1}+U_{j,k+1}-4U_{j,k}&=h^2F_{j,k} \end{aligned} \tag{58}$$

Therefore, the resulting discrete formula is as follows:

$$\boxed{U_{j-1,k}+U_{j+1,k}+U_{j,k-1}+U_{j,k+1}-4U_{j,k}=h^2F_{j,k}} \tag{59}$$

In order to solve (59), we need to reformulate it into a matrix form. In order to do so, we first define the following girds:



Drawing 11: Grid discretization of the 2D plate.

Next, based on the Drawing 11, we can then get the following equation:

$$\underbrace{\begin{bmatrix} -4 & 1 & 0 & 1 & & & & \cdots & 0 \\ 1 & -4 & 1 & 0 & 1 & & & & \vdots \\ & \ddots & \ddots & \ddots & \ddots & \ddots & & & \\ 0 & 1 & -4 & 0 & 0 & 1 & & & \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & & \\ & 1 & 0 & 1 & -4 & 1 & 0 & 1 & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ & & 1 & 0 & 1 & -4 & 1 & 0 & 1 \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & 1 & 0 & 0 & -4 & 1 & 0 \\ & & & & 1 & 0 & 1 & -4 & 1 \\ \vdots & & & & & \ddots & \ddots & \ddots & \ddots \\ 0 & \ldots & & & & 1 & 0 & 1 & -4 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} U_{1,1} \\ U_{2,1} \\ \vdots \\ U_{N_y-1,1} \\ U_{1,2} \\ U_{2,2} \\ \vdots \\ U_{N_y-1,2} \\ \vdots \\ U_{1,N_x-1} \\ U_{2,N_x-1} \\ \vdots \\ U_{N_y-1,N_x-1} \end{bmatrix}}_{\widetilde{U}} = \underbrace{\begin{bmatrix} -(U_{1,0}+U_{0,1})+h^2F_{1,1} \\ -U_{2,0}+h^2F_{2,1} \\ \vdots \\ -(U_{N_y-1,0}+U_{N_y,1})+h^2F_{N_y,1} \\ \hline -U_{0,2}+h^2F_{1,2} \\ h^2F_{2,2} \\ \vdots \\ -U_{N_y,2}+h^2F_{N_y-1,2} \\ \hline \vdots \\ \hline -(U_{1,N_x}+U_{0,N_x-1})+h^2F_{1,N_x-1} \\ -U_{2,N_x}h^2F_{2,N_x-1} \\ \vdots \\ -(U_{N_y-1,N_x}+U_{N_y,N_x-1})+h^2F_{N_y-1,N_x-1} \end{bmatrix}}_{c} \tag{60}$$

Once we have (60), we can solve it to acquire $\widetilde{U}$. In MATLAB, this can be easily done by using: $\widetilde{U} = A \setminus c$.

Let us now take a more simple example. Given the following diffusion PDE:

$$u_{xx} + u_{yy} = f(x,y)$$

where:
$0 \leqslant x \leqslant 1,\ 0 \leqslant y \leqslant 1$
$N_x = N_y = 4$
$p(y) = 0$
$q(x) = 0$
$r(y) = 100\sin(\pi y)$
$s(x) = 0$

$f(x,y) = 0$
$h = 0.25$
Find $u(x,y)$ at its steady state condition.

First, we create the matrix U as follows:

$$U = \begin{array}{c} \begin{matrix} k=0 & k=1 & k=2 & k=3 & k=4 \end{matrix} \\ \left[\begin{array}{c|ccc|c} 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 70.71 & 100 & 70.71 & 0 \end{array}\right] \end{array} \begin{matrix} j=0 \\ j=1 \\ j=2 \\ j=3 \\ j=4 \end{matrix}$$

Red colored numbers represents values at the boundaries. These values are calculated from $p(y)$, $q(x)$, $r(y)$, and $s(x)$. Next, we generate an equation similar (60) and calculate the $\widetilde{U}$.

$$\left[\begin{array}{ccc|ccc|ccc} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{array}\right] \left[\begin{array}{c} U_{1,1} \\ U_{2,1} \\ U_{3,1} \\ \hline U_{1,2} \\ U_{2,2} \\ U_{3,2} \\ \hline U_{1,3} \\ U_{2,3} \\ U_{3,3} \end{array}\right] = \left[\begin{array}{c} -U_{1,0} - U_{0,1} \\ -U_{2,0} \\ -U_{3,0} - U_{4,1} \\ \hline -U_{0,2} \\ 0 \\ -U_{4,2} \\ \hline -U_{1,4} - U_{0,3} \\ -U_{2,4} \\ -U_{3,4} - U_{4,3} \end{array}\right]$$

3.2. Neumann Boundary

4. Two-Dimensional Unsteady Advection-Diffusion

Let us consider the flowing PDE:

$$\frac{\partial T}{\partial t}+\underbrace{u\frac{\partial T}{\partial x}+v\frac{\partial T}{\partial y}}_{\text{2D Advection}}=\alpha\underbrace{\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right)}_{\text{2D Diffusion}}+f \tag{61}$$

(61) is the general transient heat equation in 2D space. It is composed of advection process and diffusion process. α is defines as:

$$\alpha=\frac{\mathcal{K}}{\rho c_p} \tag{62}$$

where:

c_p is the specific heat at constant $\left(\text{J}\left(\text{kg K}\right)\right.$

ρ is the fluid density $\left(\text{kg}/\text{m}^3\right)$

$\mathcal{K}$ is the thermal conductivity $\left(\text{W}\left(\text{m K}\right)\right)$

f is the additional forcing component

In order to discretize (61), we will use the simplest methods, which are the upwind/downwind method for the advection and the explicit FTCS method for the diffusion.

4.1. UPWIND-UPWIND Method

The upwind-upwind method is used when both velocities are positive or zero. We start by discretizing (61) as follows:

$$\begin{aligned}&\frac{T(x,y,t+\Delta t)-T(x,y,t)}{\Delta t}+u(x,y,t)\frac{T(x,y,t)-T(x-\Delta x,y,t)}{\Delta x}+\ \ldots\\ v(x,y,t)&\frac{T(x,y,t)-T(x,y-\Delta y,t)}{\Delta y}=\alpha\left(\frac{T(x+\Delta x,y,t)-2T(x,y,z)+T(x-\Delta x,y,t)}{\Delta x^2}\right)+\\ &\alpha\left(\frac{T(x,y+\Delta y,t)-2T(x,y,z)+T(x,y-\Delta y,t)}{\Delta y^2}\right)+f(x,y,t)\end{aligned} \tag{63}$$

Rearranging (63) give us:

$$\begin{aligned}T(x,y,t+\Delta t)=\ &\frac{\alpha\Delta t}{\Delta x^2}(T(x+\Delta x,y,t)-2T(x,y,z)+T(x-\Delta x,y,t))+\\ &\frac{\alpha\Delta t}{\Delta y^2}(T(x,y+\Delta y,t)-2T(x,y,z)+T(x,y-\Delta y,t))-\\ &\frac{u(x,y,t)\Delta t}{\Delta x}(T(x,y,t)-T(x-\Delta x,y,t))-\\ &\frac{v(x,y,t)\Delta t}{\Delta y}(T(x,y,t)-T(x,y-\Delta y,t))\ +\ T(x,y,t)+f(x,y,t)\Delta t\end{aligned} \tag{64}$$

Taking: $T_{i,j,k}=T(x_i,y_j,t_k)$, $x_j=j\Delta x$, $y_k=k\Delta y$, and $t_k=k\Delta t$, we can rewrite (64) as follows:

$$T_{i,j,k+1} = \alpha\Delta t\left(\left(\frac{T_{i+1,j,k}-2T_{i,j,k}+T_{i-1,j,k}}{\Delta x^2}\right)+\left(\frac{T_{i,j+1,k}-2T_{i,j,k}+T_{i,j-1,k}}{\Delta y^2}\right)\right)- \Delta t\left(\frac{u_{i,j,k}}{\Delta x}(T_{i,j,k}-T_{i-1,j,k})+\frac{v_{i,j,k}}{\Delta y}(T_{i,j,k}-T_{i,j-1,k})\right)+ T_{i,j,k} + \Delta t\ f_{i,j,k} \tag{65}$$

4.2. DOWNWIND-DOWNWIND Method

The downwind-downwind method is used when both velocities are negative or zero. The discretization process is similar to the upwind method. The only difference is in the advection part. The discretization is done as follows:

$$\begin{aligned} &\frac{T(x,y,t+\Delta t)-T(x,y,t)}{\Delta t}+u(x,y,t)\frac{T(x+\Delta x,y,t)-T(x,y,t)}{\Delta x}+ \ \ldots \\ v(x,y,t)&\frac{T(x,y+\Delta y,t)-T(x,y,t)}{\Delta y} = \alpha\left(\frac{T(x+\Delta x,y,t)-2T(x,y,z)+T(x-\Delta x,y,t)}{\Delta x^2}\right)+ \\ &\alpha\left(\frac{T(x,y+\Delta y,t)-2T(x,y,z)+T(x,y-\Delta y,t)}{\Delta y^2}\right)+f(x,y,t) \end{aligned} \tag{66}$$

Rearranging (66) give us:

$$\begin{aligned} T(x,y,t+\Delta t) = &\frac{\alpha\Delta t}{\Delta x^2}(T(x+\Delta x,y,t)-2T(x,y,z)+T(x-\Delta x,y,t))+ \\ &\frac{\alpha\Delta t}{\Delta y^2}(T(x,y+\Delta y,t)-2T(x,y,z)+T(x,y-\Delta y,t))- \\ &\frac{u(x,y,t)\Delta t}{\Delta x}(T(x+\Delta x,y,t)-T(x,y,t))- \\ &\frac{v(x,y,t)\Delta t}{\Delta y}(T(x,y+\Delta y,t)-T(x,y,t))\ +\ T(x,y,t)+f(x,y,t)\Delta t \end{aligned} \tag{67}$$

Taking: $T_{i,j,k} = T(x_i, y_j, t_k)$, $x_j = j\Delta x$, $y_k = k\Delta y$, and $t_k = k\Delta t$, we can rewrite (67) as follows:

$$T_{i,j,k+1} = \alpha\Delta t\left(\left(\frac{T_{i+1,j,k}-2T_{i,j,k}+T_{i-1,j,k}}{\Delta x^2}\right)+\left(\frac{T_{i,j+1,k}-2T_{i,j,k}+T_{i,j-1,k}}{\Delta y^2}\right)\right)- \Delta t\left(\frac{u_{i,j,k}}{\Delta x}(T_{i+1,j,k}-T_{i,j,k})+\frac{v_{i,j,k}}{\Delta y}(T_{i,j+1,k}-T_{i,j,k})\right)+ T_{i,j,k} + \Delta t\ f_{i,j,k} \tag{68}$$

4.3. DOWNWIND-UPWIND Method

The downwind-upwind method is used when the x-velocity is negative and the y-velocity is positive.

$$T_{i,j,k+1} = \alpha\Delta t\left(\left(\frac{T_{i+1,j,k}-2T_{i,j,k}+T_{i-1,j,k}}{\Delta x^2}\right)+\left(\frac{T_{i,j+1,k}-2T_{i,j,k}+T_{i,j-1,k}}{\Delta y^2}\right)\right)- \Delta t\left(\frac{u_{i,j,k}}{\Delta x}(T_{i+1,j,k}-T_{i,j,k})+\frac{v_{i,j,k}}{\Delta y}(T_{i,j,k}-T_{i,j-1,k})\right)+ T_{i,j,k} + \Delta t\ f_{i,j,k} \tag{69}$$

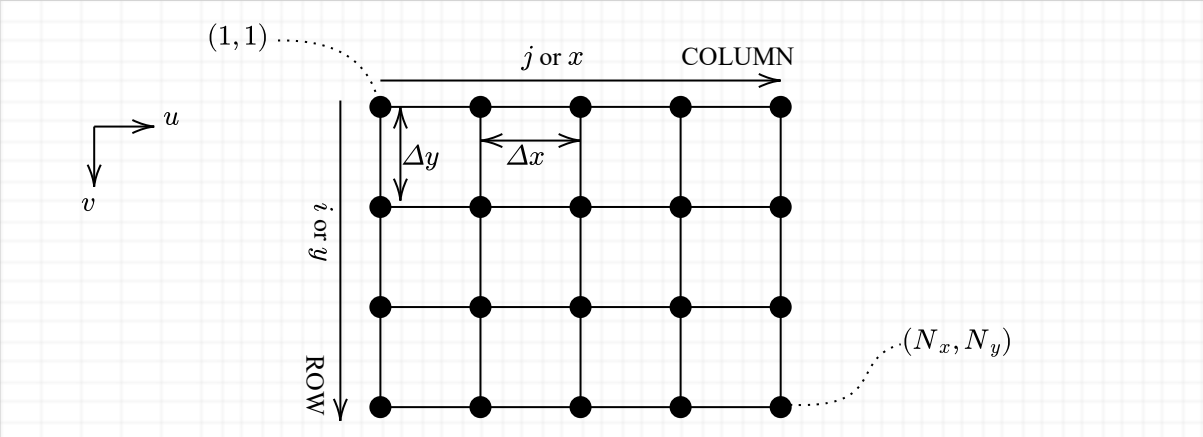
4.4. UPWIND-DOWNWIND Method

The downwind-upwind method is used when the x-velocity is positive and the y-velocity is negative.

$$T_{i,j,k+1} = \alpha\Delta t\left(\left(\frac{T_{i+1,j,k} - 2T_{i,j,k} + T_{i-1,j,k}}{\Delta x^2}\right) + \left(\frac{T_{i,j+1,k} - 2T_{i,j,k} + T_{i,j-1,k}}{\Delta y^2}\right)\right) - \Delta t\left(\frac{u_{i,j,k}}{\Delta x}(T_{i,j,k} - T_{i-1,j,k}) + \frac{v_{i,j,k}}{\Delta y}(T_{i,j+1,k} - T_{i,j,k})\right) + T_{i,j,k} + \Delta t\, f_{i,j,k} \tag{70}$$

4.5. MATLAB Implementation

First, we apply a grid discretization to the the two-dimensional rectangle as follows:



Drawing 12: A discretized two-dimensional plane

Next, we can then implement (65) and (68) in MATLAB, as follows:

```
function T = advection_diffusion(T, ...      % Temperature matrix
                                 u, ...      % x-velocity matrix
                                 v, ...      % y-velocity matrix
                                 dt, ...     % time step
                                 dx, ...     % delta x
                                 dy,...      % delta y
                                 alpha, ...  % thermal diffusivity
                                 f)          % external forcing term
r = size(T,1);
c = size(T,2);

for i = 2:r-1
    for j = 2:c-1
       if (u(i,j)>=0 && v(i,j)>=0)    % upwind-upwind
           T(i,j) = alpha*dt * ( (T(i+1,j)-2*T(i,j)+T(i-1,j))/dx^2 + ...
                    (T(i,j+1)-2*T(i,j)+T(i,j-1))/dy^2 ) - ...
                    dt * (u(i,j)/dx * (T(i,j)-T(i-1,j)) + ...
                    v(i,j)/dy * (T(i,j)-T(i,j-1)) ) + ...
                    T(i,j) + f(i,j)*dt;

       elseif (u(i,j)<0 && v(i,j)<0)    % downwind-downwind
            T(i,j) = alpha*dt * ( (T(i+1,j)-2*T(i,j)+T(i-1,j))/dx^2 + ...
                    (T(i,j+1)-2*T(i,j)+T(i,j-1))/dy^2 ) - ...
                    dt * (u(i,j)/dx * (T(i+1,j)-T(i,j)) + ...
                    v(i,j)/dy * (T(i,j+1)-T(i,j)) ) + ...
                    T(i,j) + f(i,j)*dt;

       elseif (u(i,j)>=0 && v(i,j)<0)    % upwind-downwind
```

```
            T(i,j) = alpha*dt * ( (T(i+1,j)-2*T(i,j)+T(i-1,j))/dx^2 + ...
                    (T(i,j+1)-2*T(i,j)+T(i,j-1))/dy^2 ) - ...
                    dt * (u(i,j)/dx * (T(i,j)-T(i-1,j)) + ...
                    v(i,j)/dy * (T(i,j+1)-T(i,j)) ) + ...
                    T(i,j) + f(i,j)*dt;

        elseif (u(i,j)<0 && v(i,j)>=0)    % downwind-upwind
            T(i,j) = alpha*dt * ( (T(i+1,j)-2*T(i,j)+T(i-1,j))/dx^2 + ...
                    (T(i,j+1)-2*T(i,j)+T(i,j-1))/dy^2 ) - ...
                    dt * (u(i,j)/dx * (T(i+1,j)-T(i,j)) + ...
                    v(i,j)/dy * (T(i,j)-T(i,j-1)) ) + ...
                    T(i,j) + f(i,j)*dt;
        end
    end
end
end
```

In the code listing above, we can see that the implemented functions never update the boundaries. Thus, we can conveniently set the boundary condition only once for the Dirichlet boundary condition. However, for he Neumann boundary condition, we will need to create phantom nodes. This topic will be further discussed in the next section.

Now that we have covered all possible directions for the velocity vector and implemented them in MATLAB, we will then test our MATLAB implementation in the following scenario. First, we will define a circular vector field on a thin plate. This vector field represents the velocity vector. Next, we constrained the temperature at a certain location in that plate (heat source) and watch how the heat propagates.

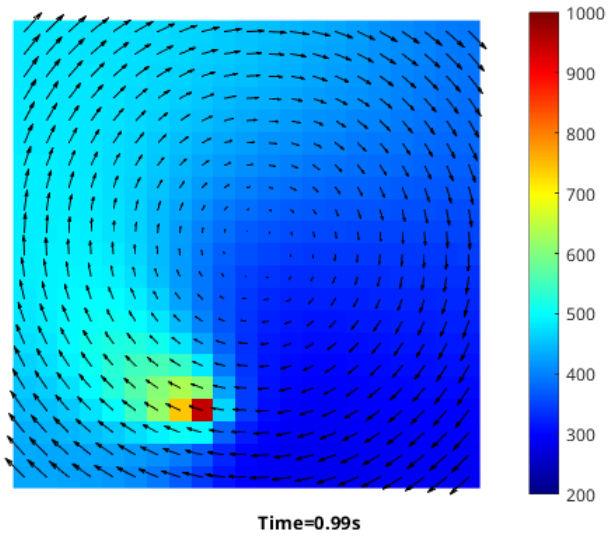
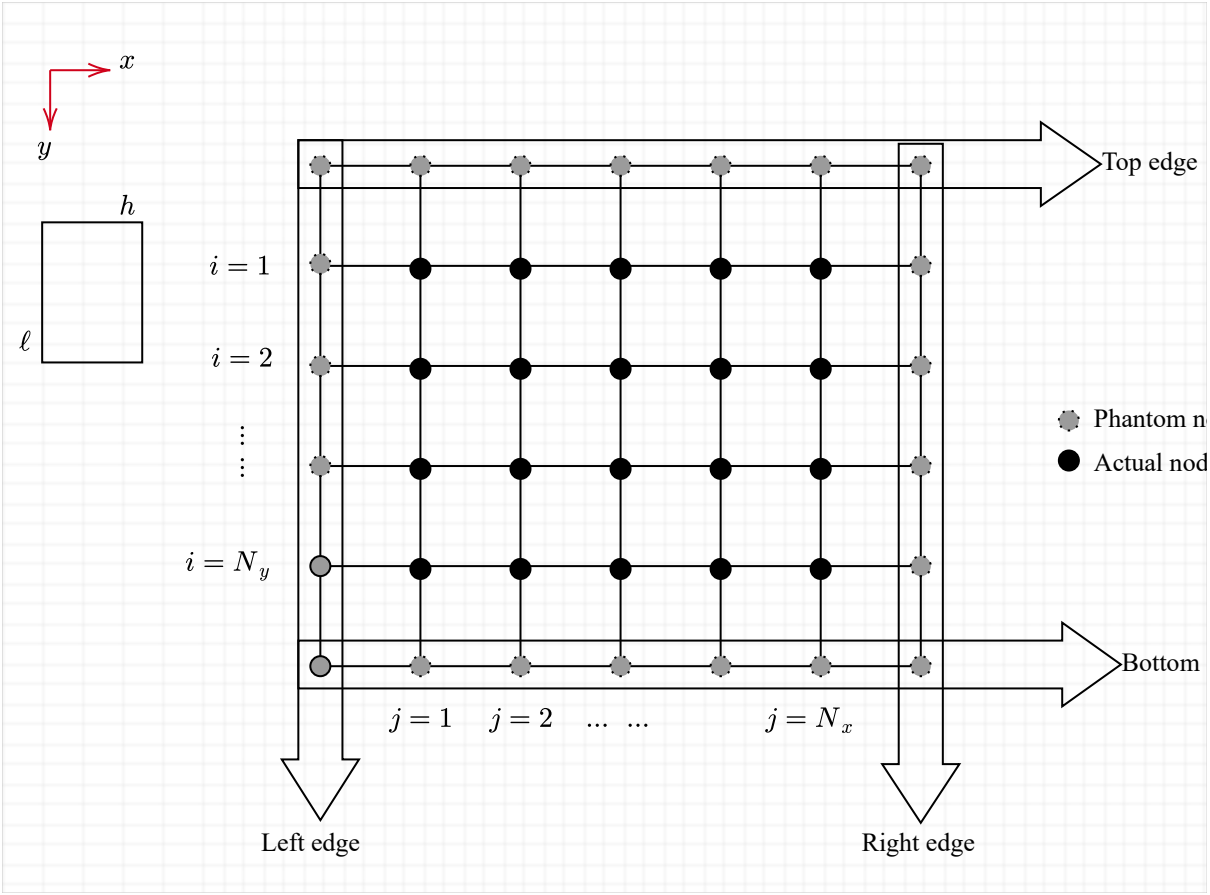


Figure 9: Heat advection on a plate with circular velocity vector field

4.6. Neumann Boundary Condition



Drawing 13: A discretized two-dimensional plane

4.6.1. Left Edge

Assuming that boundary condition for the left edge is $\frac{\partial T(x,0,t)}{\partial x} = A$. thus we have:

$$\frac{T(i,2)-T(i,0)}{2\Delta x} = A \tag{71}$$

Hence:

$$\boxed{T(i,0) = T(i,2) - 2\Delta x A} \tag{72}$$

4.6.2. Right Edge

Assuming that boundary condition for the left edge is $\frac{\partial T(x,h,t)}{\partial x} = B$. thus we have:

$$\frac{T(i,N_x+1)-T(i,N_x-1)}{2\Delta x} = B \tag{73}$$

Hence

$$\boxed{T(i,N_x+1) = T(i,N_x-1) + 2\Delta x B} \tag{74}$$

4.6.3. Top Edge

Assuming that boundary condition for the left edge is $\frac{\partial T(0,y,t)}{\partial y} = C$. thus we have:

$$\frac{T(2,j)-T(0,j)}{2\Delta y} = C \tag{75}$$

Hence

$$\boxed{T(0,j) = T(2,j) - 2\Delta y C} \tag{76}$$

4.6.4. Bottom Edge

Assuming that boundary condition for the left edge is $\dfrac{\partial T(\ell, y, t)}{\partial y} = D$. thus we have:

$$\frac{T(N_x+1, j) - T(N_x-1, j)}{2\Delta y} = D \tag{77}$$

Hence

$$\boxed{T(N_y+1, j) = T(N_y-1, j) + 2\Delta y D} \tag{78}$$

4.7. Energy Balance Model: Out-of-Plane Heat Transfer for a Thin Plate

Up to now, we only deal with homogenous equation. However, when the system interacts with its environment or with other system, the governing equations become non non-homogenous. To demonstrate this, we will implement the following scenario in MATLAB and compare the result with COMSOL. This scenario is taken from the COMSOL Application Gallery[8]. A very thin plate ($1\ \text{m} \times 1\ \text{m}$, and thickness of $0.001\ \text{m}$) is heated only at the left edge. The temperature applied heat is $800\ \text{K}$. Other edges are thermally insulated. Heat transfer to the external environment only happens from the two surfaces of the plate.

4.7.1. Transient Case

For the transient case, the governing heat equation of the system above is as follows:

$$\begin{aligned} d_z \rho c_p \frac{\partial T}{\partial t} &= d_z \mathcal{K}\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) - h_{\text{u}}(T - T_\infty) - h_{\text{d}}(T - T_\infty) \\ &\quad - \varepsilon_{\text{u}}\sigma\left(T^4 - T_\infty^4\right) - \varepsilon_{\text{d}}\sigma\left(T^4 - T_\infty^4\right) \end{aligned} \tag{79}$$

where

$d_z = 0.01\ \text{m}$

$h_{\text{u}} = h_{\text{d}} = h = 10\ \text{W}/\left(\text{m}^2\text{K}\right)$

$\varepsilon_{\text{u}} = \varepsilon_{\text{d}} = \varepsilon = 0.5$

$\sigma = 5.67 \times 10^{-8}\ \text{W}/\left(\text{m}^2\text{K}^4\right)$

The material is copper where:

$c_p = 385\ \text{J}/\left(\text{kg K}\right)$

$\mathcal{K} = 400\ \text{W}/\left(\text{m K}\right)$

$\rho = 8960\ \text{kg}/\text{m}^3$

We can simplify (79) as follows:

$$\frac{\partial T}{\partial t} = \frac{\mathcal{K}}{\rho c_p}\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) - \frac{2h}{d_z \rho c_p}(T - T_\infty) - \frac{2\varepsilon\sigma}{d_z \rho c_p}\left(T^4 - T_\infty^4\right) \tag{80}$$

The first thing we can do to see of the formula in (80) is correct is by checking the correctness of the unis. To check the units, we do it as follows:

$$\frac{\text{K}}{\text{s}} = \frac{\text{W}/\left(\text{m K}\right)}{\text{kg}/\text{m}^3\ \text{J}/\left(\text{kg K}\right)}\left(\frac{\text{K}}{\text{m}^2} + \frac{\text{K}}{\text{m}^2}\right) - \frac{\text{W}/\left(\text{m}^2\text{K}\right)}{\text{m}\ \ \text{kg}/\text{m}^3\ \text{J}/\left(\text{kg K}\right)}\text{K} - \frac{\text{W}/\left(\text{m}^2\text{K}^4\right)}{\text{m}\ \ \text{kg}/\text{m}^3\ \text{J}/\left(\text{kg K}\right)}\text{K}^4$$

[8] https://www.comsol.com/model/out-of-plane-heat-transfer-for-a-thin-plate-493

$$\frac{K}{s} = \frac{K}{s} - \frac{K}{s} - \frac{K}{s}$$

The units are indeed correct.

MATLAB implementation of (80) is as follows:

```
dz = 0.01;
cp = 385;
K = 400;
rho = 8960;
alpha = K/(rho*cp);
beta = dz*rho*cp;

dx = 0.01;
dy = 0.01;
dt = 0.001; % Small enough to not cause instability

ly = 1;     % 1x1 m^2
lx = 1;
Nx = lx/dx;
Ny = ly/dy;
Nt = tF/dt;

tF = 10000; % Simulation time, modify accordingly

h = 10;
sigma = 5.67e-8;
e = 0.5;
T_inf = 300;

clear T_array u_array v_array f_array;

% Some extra nodes for phantom nodes
u_array = zeros(Nx+2,Ny+1); % Disable advection
v_array = zeros(Nx+2,Ny+1); % Disable advection
T_array = 297.15*ones(Nx+2,Ny+1);
f_array = zeros(Nx+2,Ny+1);

% Preparation for plotting
hfig = figure;
axis off;
hold on;
title('Diffusion')
caxis([300 800])

for k = 1 : Nt
    % Heat source at the center of the plate, keep it true at all time
    T_array = apply_bc(T_array, dx, dy, ...
              ["Dirichlet", "Neumann", "Neumann", "Neumann"], ...
              [800, 0, 0, 0]);

    % Calculate the forcing component
    f_array(2:end-1,2:end-1) = -(2*h/beta*(T_array(2:end-1,2:end-1) - T_inf) ...
                      + 2*e*sigma/beta*(T_array(2:end-1,2:end-1).^4 - T_inf^4));

    % Solve the PDE
    T_array = advection_diffusion_upwind(T_array, ...
                                         u_array, ...
                                         v_array, ...
```

```
54                                          dt, ...
55                                          dx, ...
56                                          dy, ...
57                                          alpha, ...
58                                          f_array);
59 end
60
61 heatmap(T_array, false);
```

In the listing above, we disable the advection by setting $u = v = 0$. The main iteration starts from line 40 to line 59. There are three things we put in the iteration: boundary condition update, extremal forcing component calculation, and PDE solving. After 10000 second, we will acquire the following result:

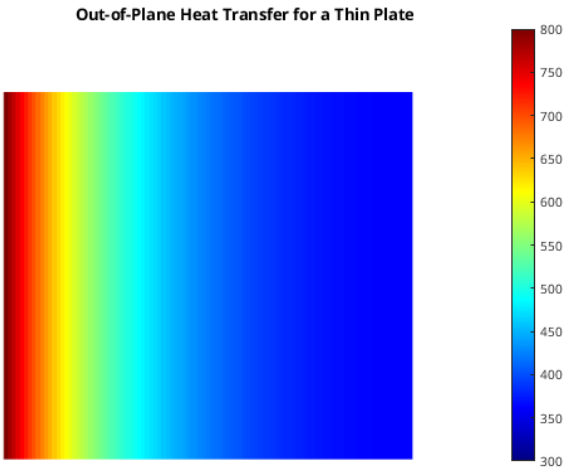


Figure 10: Temperature distribution of the plate after 10000 seconds.

We can compare the result with COMSOL. However, the provided COMSOL file only simulates the stationary case.

4.7.2. Stationary Case

For the stationary case, the governing heat equation of the system above is as follows:

$$\begin{aligned} 0 = \frac{\mathcal{K}}{\rho c_p}\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) - \frac{h_\mathrm{u}}{d_z \rho c_p}(T - T_\infty) - \frac{h_\mathrm{d}}{d_z \rho c_p}(T - T_\infty) \\ - \frac{\varepsilon_\mathrm{u}\sigma}{d_z \rho c_p}\left(T^4 - T_\infty^4\right) - \frac{\varepsilon_\mathrm{d}\sigma}{d_z \rho c_p}\left(T^4 - T_\infty^4\right) \end{aligned} \tag{81}$$

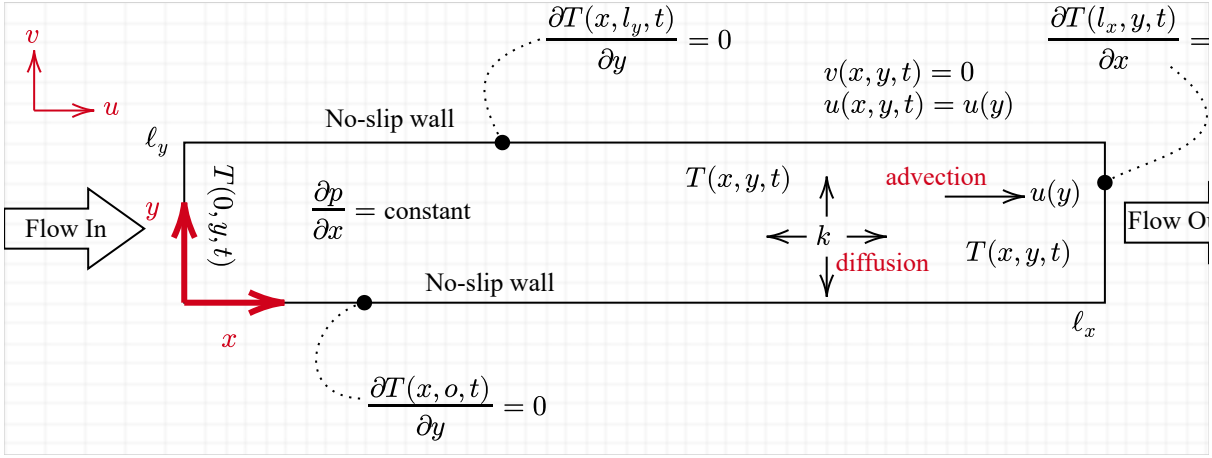
The system in (81) is a 2D Poisson PDE, which we can solve it by using the five-point difference method.

5. Navier-Stokes with Heat Transfer

In this section, we will model a cross-current heat exchanger by using the famous Navier-Stokes equation for fluid flows with heat transfer. We will start with one flow first. After that, we will implement both flows and the occurring heat transfer between them.

5.1. Channel Flow with Heat Equation

We will model the flow of the fluid as a two-dimensional flow. The flow is driven by pressure gradient (planar Poiseuille flow).



Drawing 14: Heat transfer in a two-dimensional channel flow

Conservation of mass:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{82}$$

Conservation of linear momentum:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{83}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{84}$$

Conservation of energy:

$$\rho c_p\left(\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y}\right) = k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) \tag{85}$$

where:
$u(x,y,t)$ is the x-velocity at location (x,y) at time t
$v(x,y,t)$ is the y-velocity at location (x,y) at time t
$p(x,y,t)$ is the pressure at location (x,y) at time t
$T(x,y,t)$ is the temperature at location (x,y) at time t
ν is the fluid viscosity coefficient.
c_p is the specific heat at constant
ρ is the fluid density
k is the thermal diffusivity rate

In planar Poiseuille flow, the flow is only in x-direction. u becomes a function of only y. Therefore: $v = 0$, $\frac{\partial v}{\partial t} = 0$, $\frac{\partial v}{\partial y} = 0$, $\frac{\partial^2 v}{\partial y^2} = 0$, $\frac{\partial u}{\partial t} = 0$, $\frac{\partial u}{\partial x} = 0$, and $\frac{\partial p}{\partial x} = 0$. As a result, (82) to (84) can be simplified as follows:

$$\frac{1}{\rho}\left(\frac{\partial p}{\partial x}\right) = \nu\frac{\partial^2 u}{\partial y^2} \tag{86}$$

Since $\frac{\partial p}{\partial x}$ is known, we are left with only finding $u(y)$ as the solution to (86). Thus, now we have an ODE instead of a PDE. As an ODE, we can directly solve $\frac{1}{\rho}\left(\frac{dp}{dx}\right) = \nu\frac{d^2u}{dy^2}$ as follows:

$$u(y) = \frac{1}{\rho\nu}\left(\frac{dp}{dx}\right)\frac{y^2}{2} + C_1 y + C_2$$

At no-slip condition, $u(0) = 0$ and $u(l_y) = 0$. By using these conditions, we can find C_1 and C_2. Let us the first boundary condition:

$$u(0) = 0 = C_2$$

Next, let us take the second boundary:

$$u(l_y) = 0 = \frac{1}{\rho\nu}\left(\frac{dp}{dx}\right)\frac{l_y^2}{2} + C_1 l_y$$

$$C_1 = -\frac{1}{\rho\nu}\left(\frac{dp}{dx}\right)\frac{l_y}{2}$$

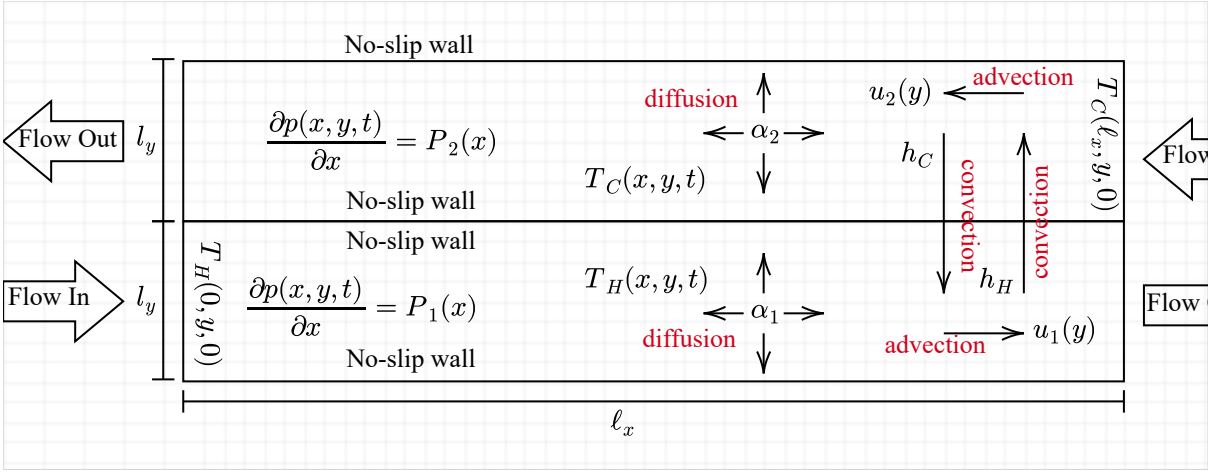
Hence:

$$u(y) = \frac{1}{\rho\nu}\left(\frac{dp}{dx}\right)\frac{y^2}{2} + \underbrace{\left(-\frac{1}{\rho\nu}\left(\frac{dp}{dx}\right)\frac{l_y}{2}\right)}_{C_1} y$$

$$= \frac{1}{\rho\nu}\left(\frac{dp}{dx}\right)\frac{y}{2}(y - l_y)$$

5.2. Two Counter-Direction Channel Flows with Heat Equations

Given below a simple two-dimensional model of a cross-current heat exchanger. The model is compose of two channel flows. Between the two channel, heat transfer occurs.



Drawing 15: Heat transfer in a Two-dimensional model of a cross-current heat exchanger

Conservation of mass:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{87}$$

Conservation of linear momentum:

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}=-\frac{1}{\rho}\frac{\partial p}{\partial x}+\nu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right) \tag{88}$$

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}=-\frac{1}{\rho}\frac{\partial p}{\partial y}+\nu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right) \tag{89}$$

Conservation of energy:

$$\rho c_p\left(\frac{\partial T_H}{\partial t}+u\frac{\partial T_H}{\partial x}+v\frac{\partial T_H}{\partial y}\right)=\alpha_1\left(\frac{\partial^2 T_H}{\partial x^2}+\frac{\partial^2 T_H}{\partial y^2}\right) \tag{90}$$

$$\rho c_p\left(\frac{\partial T_C}{\partial t}+u\frac{\partial T_C}{\partial x}+v\frac{\partial T_C}{\partial y}\right)=\alpha_2\left(\frac{\partial^2 T_C}{\partial x^2}+\frac{\partial^2 T_C}{\partial y^2}\right) \tag{91}$$

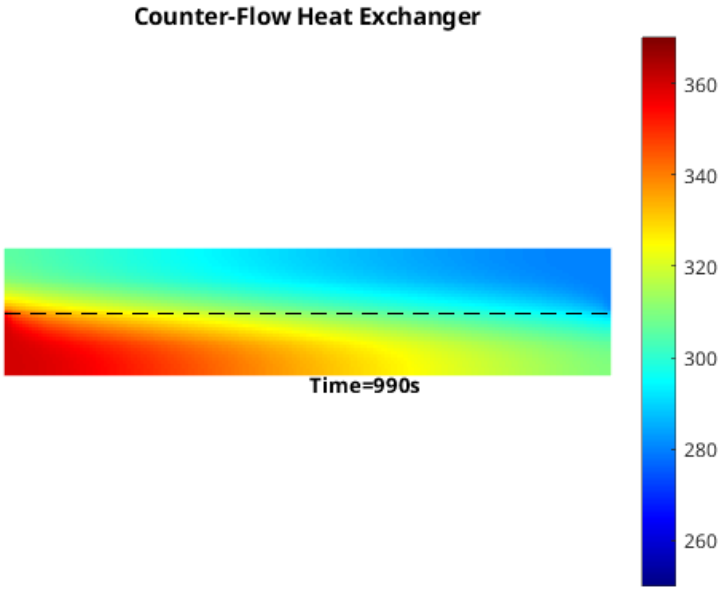
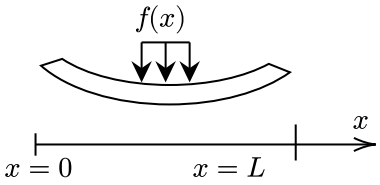


Figure 11:

6. Steady Euler-Bernoulli Beam

6.1. Mathematical Modeling

The Euler–Bernoulli equation describes the relationship between the beam's deflection and the applied load.



Drawing 16: An Euler-Bernoulli Beam

For a one-dimensional elastic beam with a length of L, its deflection is governed by the Euler-Bernoulli equation as follows:

$$\frac{d^2}{dx^2}\left[EI\,\frac{d^2w(x)}{dx^2}\right] = f(x) \tag{92}$$

In (92), $w(x)$ and $f(x)$ are the beam's deflection and the external load at location x, respectively. E is the Young's modulus, and I is the area moment of inertia. We start by assuming that E and I are constants. Hence, (92) can be rewritten as:

$$\frac{d^4w(x)}{dx^4} = \frac{1}{EI}\,f(x) \tag{93}$$

The initial conditions to (93) are made by:

$$\frac{d^2w(x)}{dx^2} = \frac{1}{EI}\,M(x) \qquad \text{Internal moment at } x$$
$$\frac{d^3w(x)}{dx^3} = \frac{1}{EI}\,F(x) \qquad \text{Internal shear force at } x$$

In a central difference method we know that:

$$\begin{aligned}
\frac{dw(x)}{dx}\bigg|_{x_i} &\approx \frac{w_{i-1} - w_{i+1}}{2\Delta x} \\
\frac{d^2w(x)}{dx^2}\bigg|_{x_i} &\approx \frac{w_{i-1} - 2w_i + w_{i+1}}{\Delta x^2} \\
\frac{d^3w(x)}{x^3}\bigg|_{x_i} &\approx \frac{-w_{i-2} + 2w_{i-1} - 2w_{i+1} + w_{i+2}}{2\Delta x^3} \\
\frac{d^4w(x)}{dx}\bigg|_{x_i} &\approx \frac{w_{i-2} - 4w_{i-1} + 6w_i - 4w_{i+1} + w_{i+2}}{\Delta x^4}
\end{aligned} \tag{94}$$

where $w_i = w(i\Delta x)$ and Δx is the segment length.

Hence, we can rewrite (93) as follows"

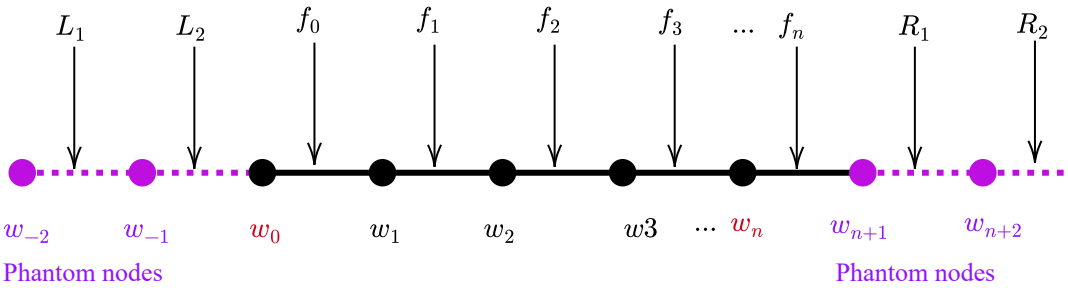
$$\begin{aligned}
\frac{d^4w(x)}{dx^4} &= \frac{1}{EI}\,f(x) \\
\sum_{i=0}^{n} \frac{w_{i-2} - 4w_{i-1} + 6w_i - 4w_{i+1} + w_{i+2}}{(\Delta x)^4} &= \frac{1}{EI}\,f_i
\end{aligned} \tag{95}$$

where $f_i = f(i\Delta x)$.

We can also express in its matrix form as follows:

$$\underbrace{\left[\begin{array}{cc|cccc|cc} 1 & 0 & & & & & \dots & 0 \\ 0 & 1 & 0 & & & & \dots & 0 \\ \hline 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & -4 & 6 & -4 & 1 \\ \hline 0 & & & & \dots & 0 & 1 & 0 \\ 0 & & & & & \dots & 0 & 1 \end{array}\right]}_{A} \underbrace{\left[\begin{array}{c} w_{-2} \\ w_{-1} \\ \hline w_0 \\ w_1 \\ \vdots \\ w_n \\ \hline w_{n+1} \\ w_{n+2} \end{array}\right]}_{W} = \underbrace{\frac{\Delta x^4}{EI}\left[\begin{array}{c} L_1 \\ L_2 \\ \hline f_0 \\ f_1 \\ \vdots \\ f_n \\ \hline R_1 \\ R_2 \end{array}\right]}_{F} \tag{96}$$

where w_{-1}, w_{-2}, w_n, w_{n+1}, L_1, L_2, R_1, and R_2 are all the properties of additional nodes that we must add . We call them as the phantom nodes (see Drawing 17).



Drawing 17: . Finite difference model of a one-dimensional problem with extra phantom nodes for the boundary conditions

6.2. Engineering Interpretation

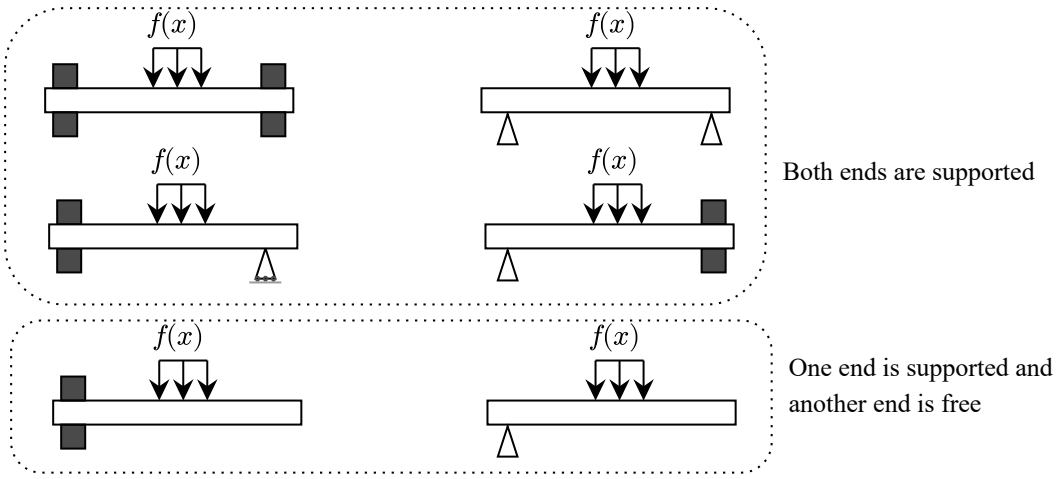
The solutions to (93) are subjected to the given initial conditions (an initial value problem). However, in engineering, it is more common to call these initial values as boundary conditions since the independent variables are spatial and not temporal. Also, (93) is actually originated from a partial differential equation.

There are three possible boundary conditions for each end, which are:

- Fixed-end (clamped), which means that there is no displacement at the current end. Additionally, at the current end, its horizontal tangent is also zero.
- Pinned-end, which means that there is no displacement at the current end. Also, at the current end, its bending moment is also zero.
- Free-end, which means that both bending moment and shear force are zero at the current end.

As additional information, bending moment is the second derivative of w with respect to x and shear force is the third derivative of w with respect to x.

The combinations of those three boundaries create nine possible configurations. However, we will focus only on 6 configuration as shown in Drawing 18.



Drawing 18: Engineering interpretation of the Euler-Bernoulli equation

Now, let us explore the boundary conditions together with their discrete forms in details.

6.2.1. Boundary Conditions at the Left End

a) Fixed-end Boundary (Clamped)

- No position displacement means that $w(0) = 0$. This then translates into:

$$w_0 = 0 \tag{97}$$

- Zero horizontal tangent means that $\frac{dw(0)}{dx} = 0$. which then translates into:

$$\frac{w_{-1} - w_1}{2\Delta x} = 0 \Longrightarrow w_{-1} = w_1 \tag{98}$$

b) Pinned-end Boundary

- No position displacement means that $w(0) = 0$, which translates to:

$$w_0 = 0 \tag{99}$$

- No bending moment means that $\frac{d^2w(0)}{dx^2} = 0$, which then translates to:

$$\frac{w_{-1} - 2w_0 + w_1}{(\Delta x)^2} = 0 \Longrightarrow w_{-1} = -w_1 \tag{100}$$

c) Free-end Boundary

- No bending moment means that $\frac{dw^2(0)}{dx^2} = 0$, which then translates into:

$$\frac{w_{-1} - 2w_0 + w_1}{(\Delta x)^2} = 0 \Longrightarrow w_{-1} = -w_1 + 2w_0 \tag{101}$$

- No shear force means that $\frac{d^3w(0)}{dx^3} = 0$, which then translate into:

$$\frac{-w_{-2} + 2w_{-1} - 2w_1 + w_2}{2(\Delta x)^3} = 0 \Longrightarrow w_{-2} = 2w_{-1} - 2w_1 + w_2 \tag{102}$$

Substituting (101) to (102) gives us:

$$\begin{aligned} w_{-2} &= 2w_{-1} - 2w_1 + w_2 \\ &= 2(2w_0 - w_1) - 2w_1 + w_2 \\ &= 4w_0 - 2w_1 - 2w_1 + w_2 \\ &= 4w_0 - 4w_1 + w_2 \end{aligned} \tag{103}$$

6.2.2. Boundary Conditions at the Right End

a) Fixed-end Boundary (Clamped)

- No position displacement means that $w(L) = 0$, which then translates into:

$$w_n = 0 \tag{104}$$

- Zero horizontal tangent means that $\frac{dw(L)}{dx} = 0$, which then translates into:

$$\frac{w_{n-1} - w_{n+1}}{2\Delta x} = 0 \Longrightarrow w_{n+1} = w_{n-1} \tag{105}$$

b) Pinned-end Boundary

- No position displacement means that $w(L) = 0$, which then translates into:

$$w_n = 0 \tag{106}$$

- No bending moment means that $\frac{d^2w(L)}{dx^2} = 0$, which then translates into:

$$\frac{w_{n-1} - 2w_n + w_{n+1}}{(\Delta x)^2} = 0 \Longrightarrow w_{n+1} = -w_{n-1} \tag{107}$$

c) Free-end Boundary

- No bending moment means that $\frac{dw^2(L)}{dx^2} = 0$, which translates into:

$$\frac{w_{n-1} - 2w_n + w_{n+1}}{(\Delta x)^2} = 0 \Longrightarrow w_{n+1} = -w_{n-1} + 2w_n \tag{108}$$

- No shear force means that $\frac{d^3w(L)}{dx^3} = 0$, which then translates into:

$$\frac{-w_{n-2} + 2w_{n-1} - 2w_{n+1} + w_{n+2}}{2(\Delta x)^3} = 0 \Longrightarrow w_{n+2} = w_{n-2} - 2w_{n-1} + 2w_{n+1} \tag{109}$$

By substituting (108) to (109), we can then acquire:

$$\begin{aligned} w_{n+2} &= w_{n-2} - 2w_{n-1} + 2w_{n+1} \\ &= w_{n-2} - 2w_{n-1} + 2(-w_{n-1} + 2w_n) \\ &= w_{n-2} - 2w_{n-1} - 2w_{n-1} + 4w_n \\ &= w_{n-2} - 4w_{n-1} + 4w_n \end{aligned} \tag{110}$$

To summarize, we put (97) to (110) into a table as follows.

	Left	Right
Fixed-end	$L_1 = \times$ $L_2 = w_1$ $w_0 = 0$	$R_1 = w_{n-1}$ $R_2 = \times$ $w_n = 0$
Pinned-end	$L_1 = \times$ $L_2 = -w_1$ $w_0 = 0$	$R_1 = -w_{n-1}$ $R_2 = \times$ $w_n = 0$
Free-end	$L_1 = 4w_0 - 4w_1 + w_2$ $L_2 = -w_1 + 2w_0$	$R_1 = -w_{n-1} + 2w_n$ $R_2 = w_{n-2} - 4w_{n-1} + 4w_n$

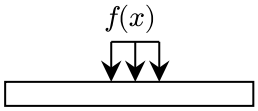
Table 1: All possible boundary conditions

Now, we can update the $L_1, L_2. R_1,$ and R_2 in (96) depending on the current boundary conditions.

6.3. System Matrices for Different Boundaries

In order to solve (96) , we must first make sure that matrix A and F contain only numbers and not variables. Currently, L_1, L_2, R_1, R_2 still contain some variables (w). This is not a desirable format since we can not apply the linear solve command to solve $Aw = F$.

➔ Free-free



Free-free boundary condition is not going to work for the Euler Bernoulli equation since the matrix that we get is not a full rank matrix. Hence, its system equation can not be solved. However, this free-free case is a a good example on how we should adapt (96) to include some boundary conditions.

By including the boundaries into (96) we get the following equation.

$$\underbrace{\left[\begin{array}{cc|cccc|cc} 1 & 0 & & & & & \dots & 0 \\ 0 & 1 & 0 & & & & \dots & 0 \\ \hline 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & -4 & 6 & -4 & 1 \\ \hline 0 & & & & \dots & 0 & 1 & 0 \\ 0 & & & & & \dots & 0 & 1 \end{array}\right]}_{A} \underbrace{\left[\begin{array}{c} w_{-2} \\ w_{-1} \\ \hline w_0 \\ w_1 \\ \vdots \\ w_n \\ \hline w_{n+1} \\ w_{n+2} \end{array}\right]}_{W} = \underbrace{\frac{\Delta x^4}{EI}\left[\begin{array}{c} 4w_0 - 4w_1 + w_2 \\ -w_1 + 2w_0 \\ \hline f_0 \\ f_1 \\ \vdots \\ f_n \\ \hline -w_{n-1} + 2w_n \\ w_{n-2} - 4w_{n-1} + 4w_n \end{array}\right]}_{F} \tag{111}$$

Starting from the left end, we write down the equations made by Row 1 to Row 4:

$$\begin{aligned} &\text{Row 1}: w_{-2} = 4w_0 - 4w_1 + w_2 \\ &\text{Row 2}: w_{-1} = -w_1 + 2w_0 \\ &\text{Row 3}: w_{-2} - 4w_{-1} + 6w_0 - 4w_1 + w_2 = f_0 \\ &\text{Row 4}: w_{-1} - 4w_0 + 6w_1 - 4w_2 + w_3 = f_0 \end{aligned} \tag{112}$$

By substituting Row 1 and Row 2 to Row 3, we get the following:

$$4w_0 - 4w_1 + w_2 - 4(-w_1 + 2w_0) + 6w_0 - 4w_1 + w_2 = f_o$$

$$2w_0 - 4w_1 + 2w_2 = f_0 \tag{113}$$

Next, by substituting Row 1 and Row 2 to Row 4, we get the following:

$$\begin{gathered} -w_1 + 2w_0 - 4w_o + 6w_1 - 4w_2 + w_3 = f_1 \\ -2w_0 + 5w_1 - 4w_2 + w_3 = f_1 \end{gathered} \tag{114}$$

As for the right end, we can do the similar steps. By substituting Row $n+1$ and Row $n+2$ to Row n, we get the following:

$$\begin{gathered} w_{n-2} - 4w_{n-1} + 6w_n - 4(-w_{n-1} + 2w_n) + w_{n-2} - 4w_{n-1} + 4w_n = f_n \\ 2w_{n-2} - 4w_{n-1} + 2w_n = f_n \end{gathered} \tag{115}$$

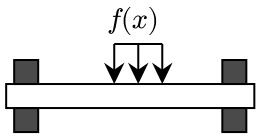
Next, by substituting Row $n+1$ and Row $n+2$ to Row $n-1$, we get the following:

$$\begin{gathered} w_{n-3} - 4w_{n-2} + 6w_{n-1} - 4w_n - w_{n-1} + 2w_n = f_{n-1} \\ w_{n-3} - 4w_{n-2} + 5w_{n-1} - 2w_n = f_{n-1} \end{gathered} \tag{116}$$

Now, we can remove Row 1, Row 2, Row $n+1$, and Row $n+2$. These rows are all made by the phantom nodes and must first be removed. As a result, we can update (111) into:

$$\underbrace{\begin{bmatrix} 2 & -4 & 2 & 0 & & & & \dots & 0 \\ -2 & 5 & -4 & 1 & 0 & & & \dots & 0 \\ 1 & -4 & 6 & -4 & 1 & 0 & & \dots & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ 0 & & \dots & 0 & 1 & -4 & 6 & -4 & 1 \\ 0 & & & \dots & 0 & 1 & -4 & 5 & -2 \\ 0 & & & & \dots & 0 & 2 & -4 & 2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{n-2} \\ w_{n-1} \\ w_n \end{bmatrix}}_{W} = \underbrace{\frac{\Delta x^4}{EI} \begin{bmatrix} f_0 \\ \vdots \\ f_n \end{bmatrix}}_{F} \tag{117}$$

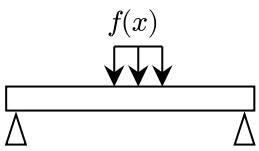
➔ Fixed-fixed



For a case where both ends are fixed, we use the following equations.

$$\underbrace{\begin{bmatrix} 7 & -4 & 1 & 0 & & & & \dots & 0 \\ -4 & 6 & -4 & 1 & 0 & & & \dots & 0 \\ 1 & -4 & 6 & -4 & 1 & 0 & & \dots & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ 0 & & & \dots & 0 & 1 & -4 & 6 & -4 \\ 0 & & & & \dots & 0 & 1 & -4 & 7 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n-2} \\ w_{n-1} \end{bmatrix}}_{W} = \underbrace{\frac{\Delta x^4}{EI} \begin{bmatrix} f_1 \\ \vdots \\ f_{n-1} \end{bmatrix}}_{F} \tag{118}$$

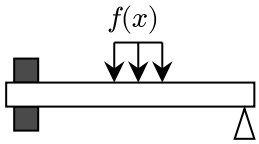
➔ Pinned-pinned



For a case where both ends are pinned, we use the following equations.

$$\underbrace{\begin{bmatrix} 5 & -4 & 1 & 0 & & & \dots & 0 \\ -4 & 6 & -4 & 1 & 0 & & \dots & 0 \\ 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ 0 & & & \dots & 0 & 1 & -4 & 6 & -4 \\ 0 & & & & \dots & 0 & 1 & -4 & 5 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n-2} \\ w_{n-1} \end{bmatrix}}_{W} = \underbrace{\frac{\Delta x^4}{EI} \overline{\begin{bmatrix} f_1 \\ \vdots \\ f_{n-1} \end{bmatrix}}}_{F} \tag{119}$$

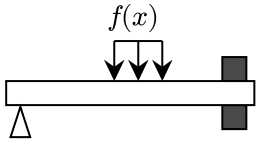
➔ Fixed-pinned



For a case where the left end is fixed while the right end is pinned, we use the following equations.

$$\underbrace{\begin{bmatrix} 7 & -4 & 1 & 0 & & & \dots & 0 \\ -4 & 6 & -4 & 1 & 0 & & \dots & 0 \\ 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ 0 & & & \dots & 0 & 1 & -4 & 6 & -4 \\ 0 & & & & \dots & 0 & 1 & -4 & 5 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n-2} \\ w_{n-1} \end{bmatrix}}_{W} = \underbrace{\frac{\Delta x^4}{EI} \overline{\begin{bmatrix} f_1 \\ \vdots \\ f_{n-1} \end{bmatrix}}}_{F} \tag{120}$$

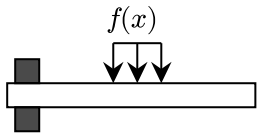
➔ Pinned-fixed



For a case where the left end is pinned while the right end is fixed, we use the following equations.

$$\underbrace{\begin{bmatrix} 5 & -4 & 1 & 0 & & & \dots & 0 \\ -4 & 6 & -4 & 1 & 0 & & \dots & 0 \\ 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ 0 & & & \dots & 0 & 1 & -4 & 6 & -4 \\ 0 & & & & \dots & 0 & 1 & -4 & 7 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n-2} \\ w_{n-1} \end{bmatrix}}_{W} = \underbrace{\frac{\Delta x^4}{EI} \overline{\begin{bmatrix} f_1 \\ \vdots \\ f_{n-1} \end{bmatrix}}}_{F} \tag{121}$$

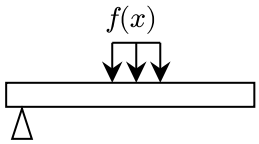
➔ Fixed-free



For a case where the left end is fixed while the right end is free, we use the following equations.

$$\underbrace{\begin{bmatrix} 7 & -4 & 1 & 0 & & & \dots & 0 \\ -4 & 6 & -4 & 1 & 0 & & \dots & 0 \\ 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ & \dots & 0 & 1 & -4 & 6 & -4 & 1 & 0 \\ & & \dots & 0 & 1 & -4 & 6 & -4 & 1 \\ \vdots & & & \dots & 0 & 1 & -4 & 5 & -2 \\ 0 & & & & \dots & 0 & 2 & -4 & 2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n-1} \\ w_n \end{bmatrix}}_{W} = \underbrace{\frac{\Delta x^4}{EI}\begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}}_{F} \tag{122}$$

➔ Pinned-free

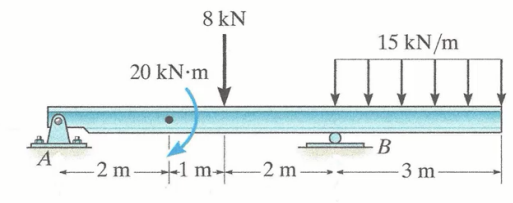


.For a case where the left end is pinned while the right end is free, we use the following equations.

$$\underbrace{\begin{bmatrix} 5 & -4 & 1 & 0 & & & \dots & 0 \\ -4 & 6 & -4 & 1 & 0 & & \dots & 0 \\ 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ & \dots & 0 & 1 & -4 & 6 & -4 & 1 & 0 \\ & & \dots & 0 & 1 & -4 & 6 & -4 & 1 \\ \vdots & & \dots & 0 & 1 & -4 & 5 & 1 & -2 \\ 0 & & & & \dots & 0 & 2 & -4 & 2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n-1} \\ w_n \end{bmatrix}}_{W} = \underbrace{\frac{\Delta x^4}{EI}\begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}}_{F} \tag{123}$$

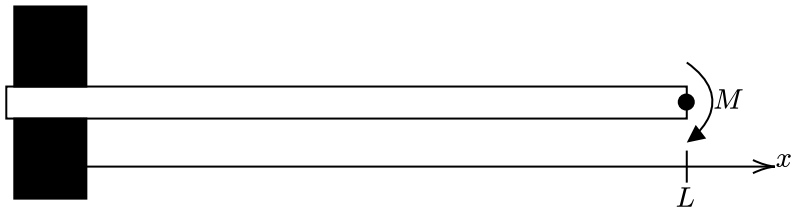
6.4. Mathematical Interpretation of Explicit Moments in Beams

In engineering, we are often given the problem that is similar to the picture below:



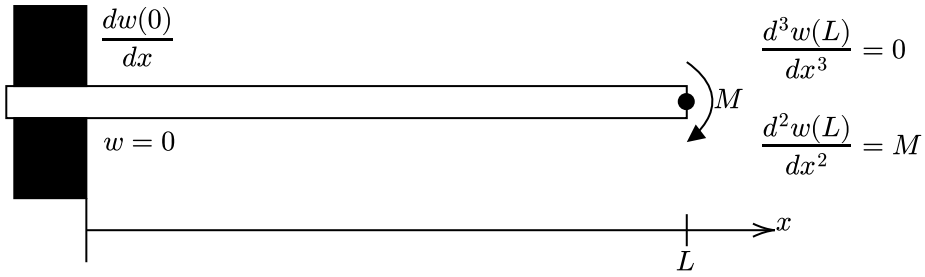
How we should mathematically interpret a case where an explicit moment is applied to a beam knowing that Euler-Bernoulli equation only has force and not torque in its formula. More importantly, direction of forces in an Euler-Bernoulli equation can only be upward or downward. Even if we apply a couple (two equal, opposite and parallel non-collinear forces), we have to maintain the directions of the two forces so that they are always perpendicular to the beam segment between the two force locations as the beam is bending.

To understand this, let us start with a simple case as follows. We are given a beam, with fixed/clamped left end and free right end. Our task is to find the deflection of the beam. An explicit moment (M) is applied to beam's right end.



First, let us write down the boundary condition of the beam above.

- The left end is fixed. Hence, no position displacement occurs $(w(0) = 0)$ and its tangent is zero $\left(\frac{dw(0)}{dx} = 0\right)$.
- An explicit moment is applied at the right end. Since moment is the second derivative of w with respect to x, we can then write: $\frac{d^2w(L)}{dx^2} = M$. We also must consider that applying a pure torque at a certain point does not generate shear force at that point. Hence, we also have $\frac{d^3w(L)}{dx^3} = 0$ as the boundary condition since shear force is the third derivative of w with respect to x.



Now that we have all the knowledge we need, we can start solving the problem.

Left end is fixed (clamped):

- No position displacement means that $w(0) = 0$. This then translates into:

$$w_0 = 0 \tag{124}$$

- Zero horizontal tangent means that $\frac{dw(0)}{dx} = 0$. which then translates into:

$$\frac{w_{-1} - w_1}{2\Delta x} = 0 \Longrightarrow w_{-1} = w_1 \tag{125}$$

A pure torque that causes bending is applied at the right end:

- A manually-defined constant bending moment of M or $\frac{dw^2(L)}{dx^2} = M$. This then translates into:

$$\frac{w_{n-1}-2w_n+w_{n+1}}{\Delta x^2}=M \Longrightarrow w_{n+1}=M\Delta x^2-w_{n-1}+2w_n \tag{126}$$

- No shear force means that $\dfrac{d^3w(L)}{dx^3}=0$, which then translates into:

$$\frac{-w_{n-2}+2w_{n-1}-2w_{n+1}+w_{n+2}}{2\ \Delta x^3}=0 \Longrightarrow w_{n+2}=w_{n-2}-2w_{n-1}+2w_{n+1} \tag{127}$$

By substituting to (126), to (128) we can then acquire:

$$\begin{aligned} w_{n+2} &= w_{n-2}-2w_{n-1}+2w_{n+1} \\ &= w_{n-2}-2w_{n-1}+2\left(M\Delta x^2-w_{n-1}+2w_n\right) \\ &= w_{n-2}-2w_{n-1}-2w_{n-1}+4w_n+2M\Delta x^2 \\ &= w_{n-2}-4w_{n-1}+4w_n \end{aligned} \tag{128}$$

Left Boundary (Fixed-end)	Right Boundary (Applied pure torque)
$L_1=\times$ $L_2=w_1$ $w_0=0$	$R_1=M\Delta x^2-w_{n-1}+2w_n$ $R_2=w_{n-2}-4w_{n-1}+4w_n$

Discretizing (93) gives us:

$$\underbrace{\left[\begin{array}{cc|cccccc|cc} 1 & 0 & & & & & \dots & 0 \\ 0 & 1 & 0 & & & & \dots & 0 \\ \hline 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & -4 & 6 & -4 & 1 \\ \hline 0 & & & & \dots & 0 & 1 & 0 \\ 0 & & & & & \dots & 0 & 1 \end{array}\right]}_{A}\underbrace{\left[\begin{array}{c} w_{-2} \\ w_{-1} \\ \hline w_0 \\ w_1 \\ \vdots \\ w_n \\ \hline w_{n+1} \\ w_{n+2} \end{array}\right]}_{W}=\frac{\Delta x^4}{EI}\underbrace{\left[\begin{array}{c} L_1 \\ L_2 \\ \hline f_0 \\ f_1 \\ \vdots \\ f_n \\ \hline R_1 \\ R_2 \end{array}\right]}_{F} \tag{129}$$

Applying the boundary conditions give us:

$$\underbrace{\left[\begin{array}{cc|cccccc|cc} 1 & 0 & & & & & \dots & 0 \\ 0 & 1 & 0 & & & & \dots & 0 \\ \hline 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & -4 & 6 & -4 & 1 \\ \hline 0 & & & & \dots & 0 & 1 & 0 \\ 0 & & & & & \dots & 0 & 1 \end{array}\right]}_{A}\underbrace{\left[\begin{array}{c} w_{-2} \\ w_{-1} \\ \hline w_0=0 \\ w_1 \\ \vdots \\ w_n \\ \hline w_{n+1} \\ w_{n+2} \end{array}\right]}_{W}=\frac{\Delta x^4}{EI}\underbrace{\left[\begin{array}{c} \times \\ w_1 \\ \hline 0 \\ 0 \\ \vdots \\ f_n \\ \hline M\Delta x^2-w_{n-1}+2w_n \\ 2M\Delta x^2+w_{n-2}-4w_{n-1}+4w_n \end{array}\right]}_{F} \tag{130}$$

Since we have already addressed the left boundary, hence, we only need to address the right boundary.
From the second-to-the last row:

$$w_{n-2}-4w_{n-1}+6w_n-4w_{n+1}+w_{n+2}=f_n$$

$$w_{n-2}-4w_{n-1}+6w_n-4\left(M\Delta x^2-w_{n-1}+2w_n\right)+\left(2M\Delta x^2+w_{n-2}-4w_{n-1}+4w_n\right)=f_n$$
$$w_{n-2}-4w_{n-1}+6w_n-2M\Delta x^2+4w_{n-1}-8w_n+w_{n-2}-4w_{n-1}+4w_n=f_n$$
$$2w_{n-2}-4w_{n-1}+2w_n=f_n+2M\Delta x^2$$

From the third-to-the last row:

$$w_{n-3}-4w_{n-2}+6w_{n-1}-4w_n+M\Delta x^2-w_{n-1}+2w_n=f_{n-1}$$
$$w_{n-3}-4w_{n-2}+5w_{n-1}-2w_n=f_{n-1}-M\Delta x^2 \tag{131}$$

$$\underbrace{\begin{bmatrix} 7 & -4 & 1 & 0 & & & \dots & 0 \\ -4 & 6 & -4 & 1 & 0 & & \dots & 0 \\ 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ & \ddots & \ddots & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & \ddots & & \\ 0 & \dots & 0 & 1 & -4 & 6 & -4 & 1 \\ 0 & & & \dots & 0 & 1 & -4 & 7 \end{bmatrix}}_{A} \tag{132}$$

$$\underbrace{\begin{bmatrix} 7 & -4 & 1 & 0 & & & & \dots & 0 \\ -4 & 6 & -4 & 1 & 0 & & & \dots & 0 \\ 1 & -4 & 6 & -4 & 1 & 0 & & \dots & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ 0 & & \dots & 0 & 1 & -4 & 6 & -4 & 1 \\ 0 & & \dots & 0 & 1 & -4 & 5 & 1 & -2 \\ 0 & & & & \dots & 0 & 2 & -4 & 2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ \vdots \\ w_{n-2} \\ w_{n-1} \\ w_n \end{bmatrix}}_{W} = \underbrace{\frac{1}{EI}\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ -M\Delta x^2 \\ 2M\Delta x^2 \end{bmatrix}}_{F} \tag{133}$$

We can then fine W by using the linear solve function in MATLAB.

Our main conclusion here is that an explicit moment in a beam is another boundary condition to our Euler-Bernoulli equation. Things will be more complicated if the explicit moment is at the center of the beam, which we will not discuss here.