Advanced Transport Phenomena

Practical session 1

Solution of transport equations including advection, diffusion and reaction in 1D using the explicit and implicit Euler methods

Alberto Cuoci

September 30, 2025

Advection-diffusion equation in 1D

1. Case description

We want to numerically solve with the Finite Difference (FDM) method the **advection-diffusion equation in 1D** on a domain with length L:

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} = \Gamma\frac{\partial^2 f}{\partial x^2}$$

with **periodic boundary conditions** $f\left(x=0,\forall t\right) = f\left(x=L,\forall t\right)$ and initial condition $f\left(t=0,\forall x\right) = A\ \sin\left(2\pi k\ x\right)$, where A (amplitude) and k (wave-number) are user-defined parameteres.

The chosen initial and boundary conditions correspond to the following analytical solution:

$$a\left(x,t\right) = A\ \exp\left(-4\pi^2 k^2 \Gamma\ t\right)\ \sin\left(2\pi k\left(x - u\ t\right)\right)$$

In particular, we are going to use the **explicit (or forward) Euler method** in time, while for spatial discretization of advective and diffusive terms we are going to use **centered, 2nd order discretizations**. We assume constant, uniform velocity u and diffusion coefficient Γ. The discretized equation becomes:

$$\frac{f_i^{n+1} - f_i^n}{\Delta t} + u\frac{f_{i+1}^n - f_{i-1}^n}{2h} = \Gamma\frac{f_{i+1}^n - 2f_i^n + f_{i-1}^n}{h^2}$$

where n is the time level and i the grid point.

We want also to monitor the error between the analytical $a\left(x,t\right)$ and the numerical solution $f\left(x,t\right)$:

$$E = \left\|a\left(x,t\right) - f\left(x,t\right)\right\|_2$$

where $\left\|\mathbf{v}\right\|_2$ indicates the Euclidean norm of vector $\mathbf{v}$.

We also monitor the squared areas below the analytical and numerical curves, to better understand the differences between the analytical and numerical curves:

$$a_{int}^2\left(t\right) = \int_0^L a^2\left(x,t\right)\mathrm{dx}$$

$$f_{int}^2\left(t\right) = \int_0^L f^2\left(x,t\right)\mathrm{dx}$$

From the theory, we know that we can have stable solutions only if the following constraints on the Courant number Co and the Diffusion number Di are satisfied:

$$\begin{array}{c}\mathrm{Co} = \frac{u\Delta t}{h} < 1\\ \mathrm{Di} = \frac{\Gamma\Delta t}{h^2} < \frac{1}{2}\end{array}$$

This means that the time step we can consider must satisfy the following expression:

$$\Delta t < \min\left(1\frac{h}{u}, \frac{1}{2}\frac{h^2}{\Gamma}\right)$$

2. Implementation

Numerical setup

We start with the definition of the input data: n (number of grid points), n_{step} (number of time steps), Δt (time step), u (velocity), L (length), Γ (diffusion equation):

```
np=21;               % number of grid points
nstep=100;           % number of time steps
L=2.0;               % domain length [m]
dt=0.05;             % time step [s]
u=1;                 % velocity [m/s]
D=0.05;              % diffusion coefficient [m2/s]
```

We also define the 2 parameters for the initial condition:

```
A=0.5;               % amplitude of initial solution
k=1;                 % wave number [1/m]
```

We calculate now the spatial step h:

```
h=L/(np-1);          % grid step [m]
```

In order to ensure a good computational efficiency, it is convenient to pre-allocate memory for the relevant vectors y (new solution at time $n+1$),f(old solution at time n) and a (the analytical solution):

```
fo=zeros(np,1);      % temporary numerical solution
f=zeros(np,1);       % current numerical solution
a=zeros(np,1);       % exact solution
```

Let's define the initial solution:

```
for i=1:np
        f(i)=A*sin(2*pi*k*h*(i-1));
end
```

We can now plot the initial solution:

```
plot(0:h:L, f);
```

It is convenient to check if the stability constraints are satisfied:

```
Co = u*dt/h;                            % Courant number
Di = D*dt/h^2;                          % Diffusion number
dt_max = min(1*h/u, 0.5*h*h/D);         % Maximum allowed time step
fprintf('Co=%f, Di=%f, dt=%f, dt(max)=%f\n', Co, Di, dt, dt_max);
```

b. Advancing the solution in time

We can now proceed with the numerical solution, advancing in time, starting from $t = 0$. Let's apply the forward Euler method over all the internal points:

$$f_i^{n+1} = f_i^n - \frac{u\Delta t}{2h}\left(f_{i+1}^n - f_{i-1}^n\right) + \frac{\Gamma\Delta t}{h^2}\left(f_{i+1}^n - 2f_i^n + f_{i-1}^n\right)$$

The corresponding code is reported in the following:

```
fo=f;
for i=2:np-1
 f(i) = fo(i)-(u*dt/2/h)*(fo(i+1)-fo(i-1))+...      % advection
     D*(dt/h^2)*(fo(i+1)-2*fo(i)+fo(i-1));   % diffusion
end
```

Let's update the boundary conditions. In particular, we apply the same discretization formula above on the last point of the grid, by considering that the point on the right side is point 2, because of periodicity:

$$f_N^{n+1} = f_N^n - \frac{u\Delta t}{2h}\left(f_2^n - f_{N-1}^n\right) + \frac{\Gamma\Delta t}{h^2}\left(f_2^n - 2f_N^n + f_{N-1}^n\right)$$

The corresponding code is reported in the following:

```
f(np) = fo(np)-(u*dt/2/h)*(fo(2)-fo(np-1)) + D*(dt/h^2)*(fo(2)-2*fo(np)+fo(np-1));
f(1)  = f(np);
```

Let's update the analytical solution:

```
for i=1:np
 a(i) = A*exp(-4*pi*pi*k*k*D*t)*sin(2*pi*k*(h*(i-1)-u*t));
end
```

In order to compare the analytical and numerical solutions it is convenient to define some integral quantities, for example the squared area below the curves (which is a measure of the energy of the system):

```
a2_int = 0.;
f2_int = 0.;
for i=1:np-1
     a2_int = a2_int + h/2*(a(i)^2+a(i+1)^2);
     f2_int = f2_int + h/2*(f(i)^2+f(i+1)^2);
end
```

We also want to compare the error between the numerical and analytical solution through the norm-2 of their difference:

```
E = 0;
for i=1:np
     E = E + (f(i)-a(i))^2;
end
E = h*sqrt(E);
```

We are now ready to move to the next time level:

```
t = t+dt;
```

c. Graphical output

Before starting the calculation, we prepare and open a video stream to register the evolution of the solution in time:

```
video_name = 'advection_diffusion_1d.mp4';
videompg4 = VideoWriter(video_name, 'MPEG-4');
open(videompg4);
```

At each time step we can update the plots showing the current numerical solution (compared with the analytical profile):

```
message = sprintf('time=%d\na^2(int)=%d\ny^2(int)=%d', t, a2_int, f2_int);
hold off; plot([0:h:L],f,'linewidth',2); axis([0 L -1, 1]); % plot num.
hold on; plot([0:h:L],a,'r','linewidth',2);                 % plot exact
hold on; legend('numerical', 'exact');                      % legend
xlabel('spatial coordinate [m]');
ylabel('solution');
time = annotation('textbox',[0.15 0.8 0.1 0.1],'String',message,'EdgeColor','none');
frame = getframe(gcf);
writeVideo(v,frame);
delete(time);
```

At the end of the calculations we have to close the video stream:

```
close(videompg4);
```

3. Bringing it all together

Let's bring it all together.

```
% Cleaning the MATLAB environment
%-----------------------------------------------------------------------%
close all;
clear variables;

% User-defined data
%-----------------------------------------------------------------------%
np=21;              % number of grid points
nstep=100;          % number of time steps
L=2.0;              % domain length [m]
dt=0.05;            % time step [s]
u=1;                % velocity [m/s]
D=0.05;             % diffusion coefficient [m2/s]
A=0.5;              % amplitude of initial solution
k=1;                % wave number [1/m]

% Pre-processing of user-defined data
%-----------------------------------------------------------------------%
% Grid step calculation
h=L/(np-1);         % grid step [m]

% Memory allocation
fo=zeros(np,1);     % temporary numerical solution
```

```
f=zeros(np,1);       % current numerical solution
a=zeros(np,1);       % exact solution

% Initial solution
for i=1:np
        f(i)=A*sin(2*pi*k*h*(i-1));
end

% Check the stability conditions on time step
Co = u*dt/h;                          % Courant number
Di = D*dt/h^2;                        % Diffusion number
dt_max = min(1*h/u, 0.5*h*h/D);      % Maximum allowed time step
fprintf('Co=%f, Di=%f, dt=%f, dt(max)=%f\n', Co, Di, dt, dt_max);

% Video setup
%-----------------------------------------------------------------------%
video_name = 'advection_diffusion_1d.mp4';
videompg4 = VideoWriter(video_name, 'MPEG-4');
open(videompg4);

% Advancing in time
%-----------------------------------------------------------------------%
t = 0.;
for m=1:nstep

    % Update the analytical solution
    for i=1:np
                a(i) = ...
                   A*exp(-4*pi*pi*k*k*D*t)*sin(2*pi*k*(h*(i-1)-u*t));
    end

    % Squared areas below the analytical and numerical solutions
    a2_int = 0.;
    f2_int = 0.;
    for i=1:np-1
         a2_int = a2_int + h/2*(a(i)^2+a(i+1)^2);
         f2_int = f2_int + h/2*(f(i)^2+f(i+1)^2);
    end

    % Graphical output
    message = sprintf('time=%d\na^2(int)=%d\ny^2(int)=%d', t, a2_int, ...
       f2_int);
    hold off; plot(0:h:L,f,'linewidth',2); axis([0 L -1, 1]); % plot ...
       num.
    hold on; plot(0:h:L,a,'r','linewidth',2);                 % plot ...
       exact
    hold on; legend('numerical', 'exact');                    % legend
    xlabel('spatial coordinate [m]');
    ylabel('solution');
    time = annotation('textbox',[0.15 0.8 0.1 ...
       0.1],'String',message,'EdgeColor','none');
    frame = getframe(gcf);
    writeVideo(videompg4,frame);
    delete(time);
    % Forward Euler method
    fo=f;
    for i=2:np-1
                f(i) = fo(i)-(u*dt/2/h)*(fo(i+1)-fo(i-1))+...      % ...
                    advection
                            D*(dt/h^2)*(fo(i+1)-2*fo(i)+fo(i-1));   % ...
                                diffusion
    end

    % Periodic boundary condition
    f(np) = fo(np)-(u*dt/2/h)*(fo(2)-fo(np-1))+...
            D*(dt/h^2)*(fo(2)-2*fo(np)+fo(np-1));
    f(1)  = f(np);

    % Update the error between numerical and analytical solution
    E = 0;
    for i=1:np
        E = E + (f(i)-a(i))^2;
    end
    E = h*sqrt(E);

    % New time step
    t=t+dt;

    % Print the current time (every 25 steps)
    if (mod(m,25)==1), fprintf('time=%d E=%e\n', t, E); end
end

% Closing the video stream
close(videompg4);
```

Play the video plotting the solution

```
implay(video_name);
```

4. Exercises

1. Analyze the behavior of the solution by considering higher Courant and/or Diffusion numbers. What happens when they do not satisfy the stability conditions?
2. Demonstrate (on a numerical basis) that the spatial discretization error decreases quadratically (order of accuracy $n = 2$) with the spatial step h. In order to this, run the simulations using always the same time interval, which must be chosen sufficiently small ($\Delta t = 0.0005$ for example), and vary the number of grid points (for example using the sequence $11, 21, 41, 61, 81, 101, 121$). Be sure that the error is evaluated always at the same final time ($t = 0.50$ for example).
3. Demonstrate (on a numerical basis) that the time discretization error decreases linearly (order of accuracy $n = 1$) with the time step Δt adopting the same procedure used for studying the spatial accuracy.

Advection-diffusion-reaction equation in 1D

1. Case description

We want to numerically solve with the Finite Difference (FDM) method the **advection-diffusion-reaction equation in 1D** on a domain with length L:

$$\frac{\partial C}{\partial t} + u\frac{\partial C}{\partial x} = \Gamma\frac{\partial^2 C}{\partial x^2} - \mathrm{kC}$$

with the **boundary conditions** $C\left(x = 0, \forall t\right) = C_{\mathrm{in}}$ and $\mathrm{zero} - \mathrm{diffusion}$ at the outlet and initial condition $C\left(t = 0, \forall x\right) = C_0$.

a. Explicit (forward) Euler method

We start using the **explicit (or forward) Euler method** in time, while for spatial discretization of advective and diffusive terms we use **centered, 2nd order discretizations**. We assume constant, uniform velocity u and diffusion coefficient Γ. The discretized equation becomes:

$$\frac{C_i^{n+1} - C_i^n}{\Delta t} + u\frac{C_{i+1}^n - C_{i-1}^n}{2h} = \Gamma\frac{C_{i+1}^n - 2C_i^n + C_{i-1}^n}{h^2} - \mathrm{kC}_i^n$$

where n is the time level and i the grid point.

From the theory, we know that we can have stable solutions only if the following constraints on the Courant number Co and the Diffusion number Di are satisfied:

$$\begin{array}{c} \mathrm{Co} = \frac{u\Delta t}{h} < 1 \\ \mathrm{Di} = \frac{\Gamma\Delta t}{h^2} < \frac{1}{2} \end{array}$$

This means that the time step we can consider must satisfy the following expression:

$$\Delta t < \min\left(1\frac{h}{u}, \frac{1}{2}\frac{h^2}{\Gamma}\right)$$

b. Implicit (backward) Euler method

We now consider the **implicit (or backward) Euler method** in time, while for spatial discretization of advective and diffusive terms we still use **centered, 2nd order discretizations**. The discretized equation becomes:

$$\frac{C_i^n - C_i^{n-1}}{\Delta t} = -u\frac{C_{i+1}^n - C_{i-1}^n}{2h} + \Gamma\frac{C_{i+1}^n - 2C_i^n + C_{i-1}^n}{h^2} - \mathrm{kC}_i^n$$

where n is the time level and i the grid point.

c. Boundary conditions

At the inlet section, a simple Dirichlet boundary condition is prescribed:

$$C_1 = C_{\text{in}}\left(t\right)$$

At the outlet section, the *zero-diffusion* boundary condition is assumed. Therefore we can replace the original transport equation with a simplified equation, without the diffusion term:

$$\frac{\partial C}{\partial t} + u\frac{\partial C}{\partial x} = -kC$$

2. Implementation

Numerical setup

We start with the definition of the input data: n (number of grid points), n_{step} (number of time steps), Δt (time step), u (velocity), L (length), Γ (diffusion equation):

```
np=21;                  % number of grid points
nstep=100;              % number of time steps
L=1.0;                  % domain length [m]
dt=0.025;               % time step [s]
u=0.5;                  % velocity [m/s]
D=0.01;                 % diffusion coefficient [m2/s]
kappa=1;                % kinetic constant [1/s]
C0=0.;                  % initial concentration [kmol/m3]
Cin=1.;                 % inlet concentration [kmol/m3]
```

We calculate now the spatial step h:

```
h=L/(np-1);             % grid step [m]
```

In order to ensure a good computational efficiency, it is convenient to pre-allocate memory for the relevant vectors y (new solution at time $n+1$),f(old solution at time n) and a (the analytical solution):

```
Co_exp=zeros(np,1);     % previous numerical solution (explicit Euler)
C_exp=zeros(np,1);      % current numerical solution (explicit Euler)
Co_imp=zeros(np,1);     % previous numerical solution (implicit Euler)
C_imp=zeros(np,1);      % current numerical solution (implicit Euler)
A=zeros(np,np);         % linear system matrix
b=zeros(np);            % linear system RHS
```

Let's define the initial solution:

```
C_exp(1)=Cin;
C_imp(1)=Cin;
for i=2:np
        C_exp(i)=C0;
    C_imp(i)=C0;
end
```

It is convenient to check if the stability constraints are satisfied (for the explicit Euler method):

```
Co = u*dt/h;                          % Courant number
Di = D*dt/h^2;                        % Diffusion number
dt_max = min(1*h/u, 0.5*h*h/D);      % Maximum allowed time step
fprintf('Co=%f, Di=%f, dt=%f, dt(max)=%f\n', Co, Di, dt, dt_max);
```

b. Advancing the solution in time (explicit Euler)

We can now proceed with the numerical solution, advancing in time, starting from $t = 0$. Let's apply the forward Euler method over all the internal points:

$$C_i^{n+1} = C_i^n - \frac{u\Delta t}{2h}\left(C_{i+1}^n - C_{i-1}^n\right) + \frac{\Gamma\Delta t}{h^2}\left(C_{i+1}^n - 2C_i^n + C_{i-1}^n\right) - \mathrm{kC}_i^n$$

The corresponding code is reported in the following:

```
for i=2:np-1

    dC_over_dx = (u/2/h)*(Co_exp(i+1)-Co_exp(i-1));
    d2C_over_dx2 = D/h^2*(Co_exp(i+1)-2*Co_exp(i)+Co_exp(i-1));

    C_exp(i) =  Co_exp(i) - dt*dC_over_dx + ...       % advection
                            dt*d2C_over_dx2 + ...     % diffusion
                            -kappa*dt*C_exp(i);       % reaction
end
```

Let's update the boundary conditions. At the inlet we simply prescribe a Dirichlet boundary condition:

```
C_exp(1) = Cin;
```

The outlet boundary condition (negligible diffusion flux) corresponds to the following discretization:

```
C_exp(np) = Co_exp(np) -(u*dt/h)*(Co_exp(np)-Co_exp(np-1))+...  % advection
                       -kappa*dt*C_exp(np);                     % reaction
```

We are now ready to move to the next time level:

```
t = t+dt;
```

b. Advancing the solution in time (implicit Euler)

We can now proceed with the numerical solution, advancing in time, starting from $t = 0$. Let's apply the forward Euler method over all the internal points:

$$C_i^n \left(1 + 2\frac{\Gamma\Delta t}{h^2} + k\Delta t\right) + C_{i+1}^n \left(\frac{u\Delta t}{2h} - \frac{\Gamma\Delta t}{h^2}\right) + C_{i-1}^n \left(-\frac{u\Delta t}{2h} - \frac{\Gamma\Delta t}{h^2}\right) = C_i^{n-1}$$

This corresponds to a system of linear equations:

```
% Matrix coefficients (for internal points only)
AP = 1. + 2*D*dt/h^2 +kappa*dt;
AE =  u*dt/2/h - D*dt/h^2;
AW = -u*dt/2/h - D*dt/h^2;

% Internal points: complete convection-diffusion-reaction equation
for i=2:np-1
    A(i,i+1) = AE;
    A(i,i)   = AP;
    A(i,i-1) = AW;
    b(i) = Co_imp(i);
end
```

The boundary conditions need a special treatment. At the inlet we simply prescribe a Dirichlet boundary condition:

```
% Inlet boundary: prescribed inlet concentration
A(1,1) = 1.;
b(1) = Cin;
```

The outlet boundary condition (negligible diffusion flux) corresponds to the following discretization:

```
% Outlet boundary (negligible diffusion)
A(np,np-1) = -u*dt/h;
A(np,np) = 1. + u*dt/h + kappa*dt;
b(np) = Co_imp(np);
```

Now we can solve the linear system:

```
% Solve the linear system of equations
C_imp = A\b;
```

We are now ready to move to the next time level:

```
t = t+dt;
```

d. Graphical output

Before starting the calculation, we prepare and open a video stream to register the evolution of the solution in time:

```
video_name = 'advection_diffusion_reaction_1d.mp4';
videompg4 = VideoWriter(video_name, 'MPEG-4');
open(videompg4);
```

At each time step we can update the plots showing the current numerical solution (compared with the analytical profile):

```
message = sprintf('time=%d\n', t);
hold off;
plot(0:h:L,C_exp,'linewidth',2);
hold on;
plot(0:h:L,C_imp,'linewidth',2);
axis([0 L 0 1]);
legend('explicit','implicit');
xlabel('spatial coordinate [m]');
ylabel('concentration [kmol/m3]');
time = annotation('textbox',[0.15 0.8 0.1 0.1],'String',message,'EdgeColor','none');
frame = getframe(gcf);
writeVideo(videompg4,frame);
delete(time);
```

At the end of the calculations we have to close the video stream:

```
close(videompg4);
```

3. Bringing it all together

Let's bring it all together.

```
% Cleaning the MATLAB environment
%------------------------------------------------------------------------%
close all;
clear variables;

% User-defined data
%------------------------------------------------------------------------%
np=21;              % number of grid points
nstep=100;          % number of time steps
L=1.0;              % domain length [m]
dt=0.025;           % time step [s]
u=0.5;              % velocity [m/s]
D=0.01;             % diffusion coefficient [m2/s]
kappa=1;            % kinetic constant [1/s]
C0=0.;              % initial concentration [kmol/m3]
Cin=1.;             % inlet concentration [kmol/m3]

% Pre-processing of user-defined data
%------------------------------------------------------------------------%
% Grid step calculation
h=L/(np-1);             % grid step [m]

% Memory allocation
Co_exp=zeros(np,1);     % previous numerical solution (explicit Euler)
C_exp=zeros(np,1);      % current numerical solution (explicit Euler)
Co_imp=zeros(np,1);     % previous numerical solution (implicit Euler)
C_imp=zeros(np,1);      % current numerical solution (implicit Euler)
A=zeros(np,np);         % linear system matrix
```

```
b=zeros(np);             % linear system RHS

% Initial solution
C_exp(1)=Cin;
C_imp(1)=Cin;
for i=2:np
        C_exp(i)=C0;
    C_imp(i)=C0;
end

% Check the stability conditions on time step (explicit Euler only)
Co = u*dt/h;                            % Courant number
Di = D*dt/h^2;                          % Diffusion number
dt_max = min(1*h/u, 0.5*h*h/D);         % Maximum allowed time step
fprintf('Co=%f, Di=%f, dt=%f, dt(max)=%f\n', Co, Di, dt, dt_max);

% Video setup
%-------------------------------------------------------------------------%
video_name = 'advection_diffusion_reaction_1d.mp4';
videompg4 = VideoWriter(video_name, 'MPEG-4');
open(videompg4);

% Advancing in time
%-------------------------------------------------------------------------%
t = 0.;
for m=1:nstep

    % Graphical output
    message = sprintf('time=%d\n', t);
    hold off;
    plot(0:h:L,C_exp,'linewidth',2);
    hold on;
    plot(0:h:L,C_imp,'linewidth',2);
    axis([0 L 0 1]);
    legend('explicit','implicit');
    xlabel('spatial coordinate [m]');
    ylabel('concentration [kmol/m3]');
    time = annotation('textbox',[0.15 0.8 0.1 ...
       0.1],'String',message,'EdgeColor','none');
    frame = getframe(gcf);
    writeVideo(videompg4,frame);
    delete(time);

    % Current solution
    Co_exp=C_exp;
    Co_imp=C_imp;


    % Explicit (forward) Euler
    %---------------------------------------------------------------------%
    C_exp(1) = Cin;
    for i=2:np-1
        dC_over_dx = (u/2/h)*(Co_exp(i+1)-Co_exp(i-1));
        d2C_over_dx2 = D/h^2*(Co_exp(i+1)-2*Co_exp(i)+Co_exp(i-1));
        C_exp(i) =  Co_exp(i) - dt*dC_over_dx + ...   % advection
                                dt*d2C_over_dx2 + ...              % ...
                                   diffusion
                   -kappa*dt*C_exp(i);              % reaction
    end
    C_exp(np) = Co_exp(np)-(u*dt/h)*(Co_exp(np)-Co_exp(np-1))+...  % ...
       advection
                -kappa*dt*C_exp(np);                               % ...
                   reaction



    % Implicit (backward) Euler
    %------------------------------------------------------------------%

    % Matrix coefficients (for internal points only)
    AP = 1. + 2*D*dt/h^2 +kappa*dt;
    AE =  u*dt/2/h - D*dt/h^2;
    AW = -u*dt/2/h - D*dt/h^2;

    % Inlet boundary: prescribed inlet concentration
    A(1,1) = 1.;
    b(1) = Cin;

    % Internal points: complete convection-diffusion-reaction equation
    for i=2:np-1
        A(i,i+1) = AE;
                A(i,i)   = AP;
        A(i,i-1) = AW;
        b(i) = Co_imp(i);
    end

    % Outlet boundary (negligible diffusion)
    A(np,np-1) = -u*dt/h;
    A(np,np) = 1. + u*dt/h + kappa*dt;
    b(np) = Co_imp(np);

    % Solve the linear system of equations
    C_imp = A\b;


    % New time step
    t=t+dt;

    % Print the current time (every 25 steps)
    if (mod(m,25)==1), fprintf('time=%d\n', t); end
end

% Closing the video stream
close(videompg4);
```

Play the video plotting the solution

```
implay(video_name);
```

4. Exercises

1. Analyze the behavior of the solution by considering higher Courant and/or Diffusion numbers. What happens when they do not satisfy the stability conditions?
2. Compare the solution replacing the outlet boundary condition with a direct Neumann condition. What are the differences with respect to the boundary condition corresponding to negligible diffusion flux?
3. Implement a dynamic boundary condition at the inlet section, for example a sinusoidal fluctuation of the concentration with a given amplitude and frequency: $C\left(x=0\right)=C_{\text{in}}\left(1+A\ \sin\left(2\pi \text{ft}\right)\right)$.

$$\text{Co} = \tfrac{u\Delta t}{h} < 1$$
$$\text{Di} = \tfrac{\Gamma\Delta t}{h^2} < \tfrac{1}{2}$$

This means that the time step we can consider must satisfy the following expression:

$$\Delta t < \min\left(1\frac{h}{u}, \frac{1}{2}\frac{h^2}{\Gamma}\right)$$

b. Implicit (backward) Euler method

We now consider the **implicit (or backward) Euler method** in time, while for spatial discretization of advective and diffusive terms we still use **centered, 2nd order discretizations**. The discretized equation becomes:

$$\frac{C_i^n - C_i^{n-1}}{\Delta t} = -u\frac{C_{i+1}^n - C_{i-1}^n}{2h} + \Gamma\frac{C_{i+1}^n - 2C_i^n + C_{i-1}^n}{h^2} - \text{kC}_i^n$$

where n is the time level and i the grid point.

c. Boundary conditions

At the inlet section, a simple Dirichlet boundary condition is prescribed:

$$C_1 = C_{\text{in}}\left(t\right)$$

At the outlet section, the *zero-diffusion* boundary condition is assumed. Therefore we can replace the original transport equation with a simplified equation, without the diffusion term:

$$\frac{\partial C}{\partial t} + u\frac{\partial C}{\partial x} = -kC$$

2. Implementation

Numerical setup

We start with the definition of the input data: n (number of grid points), n_{step} (number of time steps), Δt (time step), u (velocity), L (length), Γ (diffusion equation):

```
np=21;                  % number of grid points
nstep=100;              % number of time steps
L=1.0;                  % domain length [m]
dt=0.025;               % time step [s]
u=0.5;                  % velocity [m/s]
D=0.01;                 % diffusion coefficient [m2/s]
kappa=1;                % kinetic constant [1/s]
C0=0.;                  % initial concentration [kmol/m3]
Cin=1.;                 % inlet concentration [kmol/m3]
```

We calculate now the spatial step h:

```
h=L/(np-1);             % grid step [m]
```

In order to ensure a good computational efficiency, it is convenient to pre-allocate memory for the relevant vectors y (new solution at time $n+1$),f(old solution at time n) and a (the analytical solution):

```
Co_exp=zeros(np,1);     % previous numerical solution (explicit Euler)
C_exp=zeros(np,1);      % current numerical solution (explicit Euler)
Co_imp=zeros(np,1);     % previous numerical solution (implicit Euler)
C_imp=zeros(np,1);      % current numerical solution (implicit Euler)
A=zeros(np,np);         % linear system matrix
b=zeros(np);            % linear system RHS
```

Let's define the initial solution:

```
C_exp(1)=Cin;
C_imp(1)=Cin;
for i=2:np
        C_exp(i)=C0;
    C_imp(i)=C0;
end
```

It is convenient to check if the stability constraints are satisfied (for the explicit Euler method):

```
Co = u*dt/h;                          % Courant number
Di = D*dt/h^2;                        % Diffusion number
dt_max = min(1*h/u, 0.5*h*h/D);       % Maximum allowed time step
fprintf('Co=%f, Di=%f, dt=%f, dt(max)=%f\n', Co, Di, dt, dt_max);
```

b. Advancing the solution in time (explicit Euler)

We can now proceed with the numerical solution, advancing in time, starting from $t = 0$. Let's apply the forward Euler method over all the internal points:

$$C_i^{n+1} = C_i^n - \frac{u\Delta t}{2h}\left(C_{i+1}^n - C_{i-1}^n\right) + \frac{\Gamma\Delta t}{h^2}\left(C_{i+1}^n - 2C_i^n + C_{i-1}^n\right) - \mathrm{k}\mathrm{C}_i^n$$

The corresponding code is reported in the following:

```
for i=2:np-1

    dC_over_dx = (u/2/h)*(Co_exp(i+1)-Co_exp(i-1));
    d2C_over_dx2 = D/h^2*(Co_exp(i+1)-2*Co_exp(i)+Co_exp(i-1));

    C_exp(i) =  Co_exp(i) - dt*dC_over_dx + ...       % advection
                            dt*d2C_over_dx2 + ...     % diffusion
                            -kappa*dt*C_exp(i);       % reaction
end
```

Let's update the boundary conditions. At the inlet we simply prescribe a Dirichlet boundary condition:

```
C_exp(1) = Cin;
```

The outlet boundary condition (negligible diffusion flux) corresponds to the following discretization:

```
C_exp(np) = Co_exp(np) -(u*dt/h)*(Co_exp(np)-Co_exp(np-1))+...  % advection
                       -kappa*dt*C_exp(np);                     % reaction
```

We are now ready to move to the next time level:

```
t = t+dt;
```

b. Advancing the solution in time (implicit Euler)

We can now proceed with the numerical solution, advancing in time, starting from $t = 0$. Let's apply the forward Euler method over all the internal points:

$$C_i^n\left(1 + 2\frac{\Gamma\Delta t}{h^2} + k\Delta t\right) + C_{i+1}^n\left(\frac{u\Delta t}{2h} - \frac{\Gamma\Delta t}{h^2}\right) + C_{i-1}^n\left(-\frac{u\Delta t}{2h} - \frac{\Gamma\Delta t}{h^2}\right) = C_i^{n-1}$$

This corresponds to a system of linear equations:

```
% Matrix coefficients (for internal points only)
AP = 1. + 2*D*dt/h^2 +kappa*dt;
AE =  u*dt/2/h - D*dt/h^2;
AW = -u*dt/2/h - D*dt/h^2;

% Internal points: complete convection-diffusion-reaction equation
for i=2:np-1
    A(i,i+1) = AE;
    A(i,i)   = AP;
    A(i,i-1) = AW;
    b(i) = Co_imp(i);
end
```

The boundary conditions need a special treatment. At the inlet we simply prescribe a Dirichlet boundary condition:

```
% Inlet boundary: prescribed inlet concentration
A(1,1) = 1.;
b(1) = Cin;
```

The outlet boundary condition (negligible diffusion flux) corresponds to the following discretization:

```
% Outlet boundary (negligible diffusion)
A(np,np-1) = -u*dt/h;
A(np,np) = 1. + u*dt/h + kappa*dt;
b(np) = Co_imp(np);
```

Now we can solve the linear system:

```
% Solve the linear system of equations
C_imp = A\b;
```

We are now ready to move to the next time level:

```
t = t+dt;
```

d. Graphical output

Before starting the calculation, we prepare and open a video stream to register the evolution of the solution in time:

```
video_name = 'advection_diffusion_reaction_1d.mp4';
videompg4 = VideoWriter(video_name, 'MPEG-4');
open(videompg4);
```

At each time step we can update the plots showing the current numerical solution (compared with the analytical profile):

```
message = sprintf('time=%d\n', t);
hold off;
plot(0:h:L,C_exp,'linewidth',2);
hold on;
plot(0:h:L,C_imp,'linewidth',2);
axis([0 L 0 1]);
legend('explicit','implicit');
xlabel('spatial coordinate [m]');
ylabel('concentration [kmol/m3]');
time = annotation('textbox',[0.15 0.8 0.1 0.1],'String',message,'EdgeColor','none');
frame = getframe(gcf);
writeVideo(videompg4,frame);
delete(time);
```

At the end of the calculations we have to close the video stream:

```
close(videompg4);
```

3. Bringing it all together

Let's bring it all together.

```
% Cleaning the MATLAB environment
%-------------------------------------------------------------------%
close all;
clear variables;

% User-defined data
%-------------------------------------------------------------------%
np=21;              % number of grid points
nstep=100;          % number of time steps
L=1.0;              % domain length [m]
dt=0.025;           % time step [s]
u=0.5;              % velocity [m/s]
D=0.01;             % diffusion coefficient [m2/s]
kappa=1;            % kinetic constant [1/s]
C0=0.;              % initial concentration [kmol/m3]
Cin=1.;             % inlet concentration [kmol/m3]

% Pre-processing of user-defined data
%-------------------------------------------------------------------%
% Grid step calculation
h=L/(np-1);             % grid step [m]

% Memory allocation
Co_exp=zeros(np,1);     % previous numerical solution (explicit Euler)
C_exp=zeros(np,1);      % current numerical solution (explicit Euler)
Co_imp=zeros(np,1);     % previous numerical solution (implicit Euler)
C_imp=zeros(np,1);      % current numerical solution (implicit Euler)
A=zeros(np,np);         % linear system matrix
b=zeros(np);            % linear system RHS

% Initial solution
C_exp(1)=Cin;
C_imp(1)=Cin;
for i=2:np
        C_exp(i)=C0;
    C_imp(i)=C0;
end

% Check the stability conditions on time step (explicit Euler only)
Co = u*dt/h;                        % Courant number
Di = D*dt/h^2;                      % Diffusion number
dt_max = min(1*h/u, 0.5*h*h/D);     % Maximum allowed time step
fprintf('Co=%f, Di=%f, dt=%f, dt(max)=%f\n', Co, Di, dt, dt_max);

% Video setup
%-------------------------------------------------------------------%
video_name = 'advection_diffusion_reaction_1d.mp4';
videompg4 = VideoWriter(video_name, 'MPEG-4');
open(videompg4);
```

```
% Advancing in time
%--------------------------------------------------------------------%
t = 0.;
for m=1:nstep

    % Graphical output
    message = sprintf('time=%d\n', t);
    hold off;
    plot(0:h:L,C_exp,'linewidth',2);
    hold on;
    plot(0:h:L,C_imp,'linewidth',2);
    axis([0 L 0 1]);
    legend('explicit','implicit');
    xlabel('spatial coordinate [m]');
    ylabel('concentration [kmol/m3]');
    time = annotation('textbox',[0.15 0.8 0.1 ...
       0.1],'String',message,'EdgeColor','none');
    frame = getframe(gcf);
    writeVideo(videompg4,frame);
    delete(time);

    % Current solution
    Co_exp=C_exp;
    Co_imp=C_imp;


    % Explicit (forward) Euler
    %----------------------------------------------------------------%
    C_exp(1) = Cin;
    for i=2:np-1
        dC_over_dx = (u/2/h)*(Co_exp(i+1)-Co_exp(i-1));
        d2C_over_dx2 = D/h^2*(Co_exp(i+1)-2*Co_exp(i)+Co_exp(i-1));
        C_exp(i) =  Co_exp(i) - dt*dC_over_dx + ...   % advection
                                dt*d2C_over_dx2 + ...            % ...
                                   diffusion
                    -kappa*dt*C_exp(i);                % reaction
    end
    C_exp(np) = Co_exp(np)-(u*dt/h)*(Co_exp(np)-Co_exp(np-1))+...  % ...
       advection
                -kappa*dt*C_exp(np);                               % ...
                   reaction



    % Implicit (backward) Euler
    %----------------------------------------------------------------%

    % Matrix coefficients (for internal points only)
    AP = 1. + 2*D*dt/h^2 +kappa*dt;
    AE =  u*dt/2/h - D*dt/h^2;
    AW = -u*dt/2/h - D*dt/h^2;

    % Inlet boundary: prescribed inlet concentration
    A(1,1) = 1.;
    b(1) = Cin;

    % Internal points: complete convection-diffusion-reaction equation
    for i=2:np-1
        A(i,i+1) = AE;
                A(i,i)    = AP;
        A(i,i-1) = AW;
        b(i) = Co_imp(i);
    end

    % Outlet boundary (negligible diffusion)
    A(np,np-1) = -u*dt/h;
    A(np,np) = 1. + u*dt/h + kappa*dt;
    b(np) = Co_imp(np);

    % Solve the linear system of equations
    C_imp = A\b;


    % New time step
    t=t+dt;

    % Print the current time (every 25 steps)
    if (mod(m,25)==1), fprintf('time=%d\n', t); end
end

% Closing the video stream
close(videompg4);
```

Play the video plotting the solution

```
implay(video_name);
```

4. Exercises

1. Analyze the behavior of the solution by considering higher Courant and/or Diffusion numbers. What happens when they do not satisfy the stability conditions?
2. Compare the solution replacing the outlet boundary condition with a direct Neumann condition. What are the differences with respect to the boundary condition corresponding to negligible diffusion flux?
3. Implement a dynamic boundary condition at the inlet section, for example a sinusoidal fluctuation of the concentration with a given amplitude and frequency: $C\left(x=0\right) = C_{\text{in}}\left(1 + A\ \sin\left(2\pi \text{ft}\right)\right)$.

$$\text{Co} = \tfrac{u\Delta t}{h} < 1$$
$$\text{Di} = \tfrac{\Gamma\Delta t}{h^2} < \tfrac{1}{2}$$

This means that the time step we can consider must satisfy the following expression:

$$\Delta t < \min\left(1\frac{h}{u}, \frac{1}{2}\frac{h^2}{\Gamma}\right)$$

b. Implicit (backward) Euler method

We now consider the **implicit (or backward) Euler method** in time, while for spatial discretization of advective and diffusive terms we still use **centered, 2nd order discretizations**. The discretized equation becomes:

$$\frac{C_i^n - C_i^{n-1}}{\Delta t} = -u\frac{C_{i+1}^n - C_{i-1}^n}{2h} + \Gamma\frac{C_{i+1}^n - 2C_i^n + C_{i-1}^n}{h^2} - kC_i^n$$

where n is the time level and i the grid point.

c. Boundary conditions

At the inlet section, a simple Dirichlet boundary condition is prescribed:

$$C_1 = C_{\text{in}}(t)$$

At the outlet section, the zero-diffusion boundary condition is assumed. Therefore we can replace the original transport equation with a simplified equation, without the diffusion term:

$$\frac{\partial C}{\partial t} + u\frac{\partial C}{\partial x} = -kC$$

2. Implementation

Numerical setup

We start with the definition of the input data: n (number of grid points), n_{step} (number of time steps), Δt (time step), u (velocity), L (length), Γ (diffusion equation):

```
np=21;                % number of grid points
nstep=100;            % number of time steps
L=1.0;                % domain length [m]
dt=0.025;             % time step [s]
u=0.5;                % velocity [m/s]
D=0.01;               % diffusion coefficient [m2/s]
kappa=1;              % kinetic constant [1/s]
C0=0.;                % initial concentration [kmol/m3]
Cin=1.;               % inlet concentration [kmol/m3]
```

We calculate now the spatial step h:

```
h=L/(np-1);           % grid step [m]
```

In order to ensure a good computational efficiency, it is convenient to pre-allocate memory for the relevant vectors y (new solution at time $n+1$),f(old solution at time n) and a (the analytical solution):

```
Co_exp=zeros(np,1);      % previous numerical solution (explicit Euler)
C_exp=zeros(np,1);       % current numerical solution (explicit Euler)
Co_imp=zeros(np,1);      % previous numerical solution (implicit Euler)
C_imp=zeros(np,1);       % current numerical solution (implicit Euler)
A=zeros(np,np);          % linear system matrix
b=zeros(np);             % linear system RHS
```

Let's define the initial solution:

```
C_exp(1)=Cin;
C_imp(1)=Cin;
for i=2:np
        C_exp(i)=C0;
    C_imp(i)=C0;
end
```

It is convenient to check if the stability constraints are satisfied (for the explicit Euler method):

```
Co = u*dt/h;                          % Courant number
Di = D*dt/h^2;                        % Diffusion number
dt_max = min(1*h/u, 0.5*h*h/D);      % Maximum allowed time step
fprintf('Co=%f, Di=%f, dt=%f, dt(max)=%f\n', Co, Di, dt, dt_max);
```

b. Advancing the solution in time (explicit Euler)

We can now proceed with the numerical solution, advancing in time, starting from $t = 0$. Let's apply the forward Euler method over all the internal points:

$$C_i^{n+1} = C_i^n - \frac{u\Delta t}{2h}\left(C_{i+1}^n - C_{i-1}^n\right) + \frac{\Gamma\Delta t}{h^2}\left(C_{i+1}^n - 2C_i^n + C_{i-1}^n\right) - \mathrm{k}C_i^n$$

The corresponding code is reported in the following:

```
for i=2:np-1

    dC_over_dx = (u/2/h)*(Co_exp(i+1)-Co_exp(i-1));
    d2C_over_dx2 = D/h^2*(Co_exp(i+1)-2*Co_exp(i)+Co_exp(i-1));

    C_exp(i) =  Co_exp(i) - dt*dC_over_dx + ...       % advection
                            dt*d2C_over_dx2 + ...     % diffusion
                            -kappa*dt*C_exp(i);       % reaction
end
```

Let's update the boundary conditions. At the inlet we simply prescribe a Dirichlet boundary condition:

```
C_exp(1) = Cin;
```

The outlet boundary condition (negligible diffusion flux) corresponds to the following discretization:

```
C_exp(np) = Co_exp(np) -(u*dt/h)*(Co_exp(np)-Co_exp(np-1))+...  % advection
                       -kappa*dt*C_exp(np);                     % reaction
```

We are now ready to move to the next time level:

```
t = t+dt;
```

b. Advancing the solution in time (implicit Euler)

We can now proceed with the numerical solution, advancing in time, starting from $t = 0$. Let's apply the forward Euler method over all the internal points:

$$C_i^n \left(1 + 2\frac{\Gamma\Delta t}{h^2} + k\Delta t\right) + C_{i+1}^n \left(\frac{u\Delta t}{2h} - \frac{\Gamma\Delta t}{h^2} + k\Delta t\right) + C_{i-1}^n \left(-\frac{u\Delta t}{2h} - \frac{\Gamma\Delta t}{h^2} + k\Delta t\right) = C_i^{n-1}$$

This corresponds to a system of linear equations:

```
% Matrix coefficients (for internal points only)
AP = 1. + 2*D*dt/h^2 +kappa*dt;
AE =  u*dt/2/h - D*dt/h^2;
AW = -u*dt/2/h - D*dt/h^2;

% Internal points: complete convection-diffusion-reaction equation
for i=2:np-1
    A(i,i+1) = AE;
A(i,i)   = AP;
    A(i,i-1) = AW;
    b(i) = Co_imp(i);
end
```

The boundary conditions need a special treatment. At the inlet we simply prescribe a Dirichlet boundary condition:

```
% Inlet boundary: prescribed inlet concentration
A(1,1) = 1.;
b(1) = Cin;
```

The outlet boundary condition (negligible diffusion flux) corresponds to the following discretization:

```
% Outlet boundary (negligible diffusion)
A(np,np-1) = -u*dt/h;
A(np,np) = 1. + u*dt/h + kappa*dt;
b(np) = Co_imp(np);
```

Now we can solve the linear system:

```
% Solve the linear system of equations
C_imp = A\b;
```

We are now ready to move to the next time level:

```
t = t+dt;
```

d. Graphical output

Before starting the calculation, we prepare and open a video stream to register the evolution of the solution in time:

```
video_name = 'advection_diffusion_reaction_1d.mp4';
videompg4 = VideoWriter(video_name, 'MPEG-4');
open(videompg4);
```

At each time step we can update the plots showing the current numerical solution (compared with the analytical profile):

```
message = sprintf('time=%d\n', t);
hold off;
plot(0:h:L,C_exp,'linewidth',2);
hold on;
plot(0:h:L,C_imp,'linewidth',2);
axis([0 L 0 1]);
legend('explicit','implicit');
xlabel('spatial coordinate [m]');
ylabel('concentration [kmol/m3]');
time = annotation('textbox',[0.15 0.8 0.1 0.1],'String',message,'EdgeColor','none');
frame = getframe(gcf);
writeVideo(videompg4,frame);
delete(time);
```

At the end of the calculations we have to close the video stream:

```
close(videompg4);
```

3. Bringing it all together

Let's bring it all together.

```
% Cleaning the MATLAB environment
%----------------------------------------------------------------------%
close all;
clear variables;

% User-defined data
%----------------------------------------------------------------------%
np=21;              % number of grid points
nstep=100;          % number of time steps
L=1.0;              % domain length [m]
dt=0.025;           % time step [s]
u=0.5;              % velocity [m/s]
D=0.01;             % diffusion coefficient [m2/s]
kappa=1;            % kinetic constant [1/s]
C0=0.;              % initial concentration [kmol/m3]
Cin=1.;             % inlet concentration [kmol/m3]

% Pre-processing of user-defined data
%----------------------------------------------------------------------%
% Grid step calculation
h=L/(np-1);             % grid step [m]

% Memory allocation
Co_exp=zeros(np,1);     % previous numerical solution (explicit Euler)
C_exp=zeros(np,1);      % current numerical solution (explicit Euler)
Co_imp=zeros(np,1);     % previous numerical solution (implicit Euler)
C_imp=zeros(np,1);      % current numerical solution (implicit Euler)
A=zeros(np,np);         % linear system matrix
```

```
b=zeros(np);             % linear system RHS

% Initial solution
C_exp(1)=Cin;
C_imp(1)=Cin;
for i=2:np
        C_exp(i)=C0;
    C_imp(i)=C0;
end

% Check the stability conditions on time step (explicit Euler only)
Co = u*dt/h;                            % Courant number
Di = D*dt/h^2;                          % Diffusion number
dt_max = min(1*h/u, 0.5*h*h/D);        % Maximum allowed time step
fprintf('Co=%f, Di=%f, dt=%f, dt(max)=%f\n', Co, Di, dt, dt_max);

% Video setup
%-------------------------------------------------------------------------%
video_name = 'advection_diffusion_reaction_1d.mp4';
videompg4 = VideoWriter(video_name, 'MPEG-4');
open(videompg4);

% Advancing in time
%-------------------------------------------------------------------------%
t = 0.;
for m=1:nstep

    % Graphical output
    message = sprintf('time=%d\n', t);
    hold off;
    plot(0:h:L,C_exp,'linewidth',2);
    hold on;
    plot(0:h:L,C_imp,'linewidth',2);
    axis([0 L 0 1]);
    legend('explicit','implicit');
    xlabel('spatial coordinate [m]');
    ylabel('concentration [kmol/m3]');
    time = annotation('textbox',[0.15 0.8 0.1 ...
       0.1],'String',message,'EdgeColor','none');
    frame = getframe(gcf);
    writeVideo(videompg4,frame);
    delete(time);

    % Current solution
    Co_exp=C_exp;
    Co_imp=C_imp;


    % Explicit (forward) Euler
    %---------------------------------------------------------------------%
    C_exp(1) = Cin;
    for i=2:np-1
        dC_over_dx = (u/2/h)*(Co_exp(i+1)-Co_exp(i-1));
        d2C_over_dx2 = D/h^2*(Co_exp(i+1)-2*Co_exp(i)+Co_exp(i-1));
        C_exp(i) =  Co_exp(i) - dt*dC_over_dx + ...   % advection
                                dt*d2C_over_dx2 + ...              % ...
                                    diffusion
                       -kappa*dt*C_exp(i);                % reaction
    end
    C_exp(np) = Co_exp(np)-(u*dt/h)*(Co_exp(np)-Co_exp(np-1))+...  % ...
        advection
                  -kappa*dt*C_exp(np);                               % ...
                      reaction



    % Implicit (backward) Euler
    %-------------------------------------------------------------------%

    % Matrix coefficients (for internal points only)
    AP = 1. + 2*D*dt/h^2 +kappa*dt;
    AE =  u*dt/2/h - D*dt/h^2;
    AW = -u*dt/2/h - D*dt/h^2;

    % Inlet boundary: prescribed inlet concentration
    A(1,1) = 1.;
    b(1) = Cin;

    % Internal points: complete convection-diffusion-reaction equation
    for i=2:np-1
        A(i,i+1) = AE;
                A(i,i)   = AP;
        A(i,i-1) = AW;
        b(i) = Co_imp(i);
    end

    % Outlet boundary (negligible diffusion)
    A(np,np-1) = -u*dt/h;
    A(np,np) = 1. + u*dt/h + kappa*dt;
    b(np) = Co_imp(np);

    % Solve the linear system of equations
    C_imp = A\b;


    % New time step
    t=t+dt;

    % Print the current time (every 25 steps)
    if (mod(m,25)==1), fprintf('time=%d\n', t); end
end

% Closing the video stream
close(videompg4);
```

Play the video plotting the solution

```
implay(video_name);
```

4. Exercises

1. Analyze the behavior of the solution by considering higher Courant and/or Diffusion numbers. What happens when they do not satisfy the stability conditions?
2. Compare the solution replacing the outlet boundary condition with a direct Neumann condition. What are the differences with respect to the boundary condition corresponding to negligible diffusion flux?
3. Implement a dynamic boundary condition at the inlet section, for example a sinusoidal fluctuation of the concentration with a given amplitude and frequency: $C\left(x=0\right)=C_{\mathrm{in}}\left(1+A\ \sin\left(2\pi \mathrm{ft}\right)\right)$.