Advanced Transport Phenomena

Practical session 2

Solution of convection-diffusion transport equation in 2D and iterative methods for solving the Poisson equation

Alberto Cuoci

October 6, 2025

Advection-diffusion equation in 2D

1. Case description

We want to numerically solve with the Finite Difference (FDM) method the advection-diffusion equation in 2D, over a rectangular domain with size $L_x \times L_y$:

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} + v\frac{\partial f}{\partial y} = \Gamma\left(\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}\right)$$

Along the north ($y = L_y$), south ($y = 0$), and west ($x = 0$) sides, zero gradient boundary conditions (i.e. Neumann's boundary conditions) are prescribed:

$$\frac{\partial f}{\partial n} = 0$$

where n is the normal direction to the boundary. Along the east side ($x = L_x$), the following Dirichlet boundary conditions are prescribed:

$$f\left(x = L_x; y \in \left[\frac{L_y}{3}; \frac{2L_y}{3}\right]\right) = f_{\text{inlet}}$$

$$f\left(x = L_x; y \in \left[0; \frac{L_y}{3}\right] \cup \left[\frac{2L_y}{3}; L_y\right]\right) = 0$$

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

$$\frac{f_{i,j}^{n+1} - f_{i,j}^{n}}{\Delta t} + u\frac{f_{i+1,j}^{n} - f_{i-1,j}^{n}}{2h_x} + v\frac{f_{i,j+1}^{n} - f_{i,j-1}^{n}}{2h_y} = \Gamma\frac{f_{i+1,j}^{n} - 2f_{i,j}^{n} + f_{i-1,j}^{n}}{h_x^2} + \Gamma\frac{f_{i,j+1}^{n} - 2f_{i,j}^{n} + f_{i,j-1}^{n}}{h_y^2}$$

The image below reports the typical solution after a sufficiently large time on a squared grid. Notice the Dirichlet boundary conditions along the east side ($x = L_x$).

2. Implementation

a. Numerical setup

As a first example we consider a regular 2D squared grid ($L_x = L_y$), with same number of points along the two directions ($n_x = n_y$). We also assign the diffusion coefficient (constant and uniform) and the velocity fiels, i.e. the 2 components u and v. In particular, we impose $u = -1$ and $v = 0$, which physically correspond to a uniform and constant velocity field parallel to the x axis, from east to west side. The inlet value of f variable is also fixed as f_{inlet}.

```
nx=33;                  % number of grid points along x
ny=33;                  % number of grid points along y
nstep=150;              % number of time steps
lengthx=2.0;            % domain length along x [m]
lengthy=2.0;            % domain length along y [m]
```

```
D=0.025;                % diffusion coefficient [m2/s]
u=-1;                   % velocity along x [m/s]
v= 0;                   % velocity along y [m/s]
fin=1;                  % inlet value of f
```

From the physical point of view, we are studying the advection and diffusion of a scalar f which enters from the central section of the east side (see boundary conditions reported above). It is transported along the horizontal (i.e. parallel to x axis) direction because of advection and diffuses along both directions because of diffusion.

We calculate now the spatial steps h_x and h_y:

```
hx=lengthx/(nx-1);      % grid step along x [m]
hy=lengthy/(ny-1);      % grid step along y [m]
```

The time step is chosen according to the stability conditions on advection and diffusion, including a safety coefficient σ:

$$\Delta t \leq \sigma \ \min\left(\frac{4\Gamma}{u^2+v^2}, \frac{1}{4}\frac{h_x h_y}{\Gamma}\right)$$

```
sigma = 0.75;                     % safety coefficient
dt_diff  = 1/4*hx*hy/D;           % diffusion [s]
dt_conv = 4*D/(u^2+v^2);          % convection [s]
dt = sigma*min(dt_diff, dt_conv); % time step [s]
```

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

```
f=zeros(nx,ny);     % current numerical solution
fo=zeros(nx,ny);    % previous numerical solution
```

Notice that with the instructions above not only we allocated memory, but we also set the two fields equal to zero.

Let's fix the boundary conditions along the inlet section:

```
f(nx, ny*1/3:ny*2/3) = fin;
```

There is no need to set explicitly the second BC, since the $f$field has been automatically set equal to 0 everywhere.

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:

$$f_{i,j}^{n+1} = f_{i,j}^n - \frac{u\Delta t}{2h_x}\left(f_{i+1,j}^n - f_{i-1,j}^n\right) - \frac{v\Delta t}{2h_y}\left(f_{i,j+1}^n - f_{i,j-1}^n\right) + \frac{\Gamma\Delta t}{h_x^2}\left(f_{i+1,j}^n - 2f_{i,j}^n + f_{i-1,j}^n\right) + \frac{\Gamma\Delta t}{h_y^2}\left(f_{i,j+1}^n - 2f_{i,j}^n + f_{i,j-1}^n\right)$$

The corresponding code is reported in the following:

```
fo=f;
for i=2:nx-1
     for j=2:ny-1
            f(i,j) = fo(i,j)...
                    -(0.5*dt*u/hx)*(fo(i+1,j)-fo(i-1,j))...
                    -(0.5*dt*v/hy)*(fo(i,j+1)-fo(i,j-1))...
                    +(D*dt/hx^2)*(fo(i+1,j)-2*fo(i,j)+fo(i-1,j))...
                    +(D*dt/hy^2)*(fo(i,j+1)-2*fo(i,j)+fo(i,j-1));
     end
end
```

We then apply the Neumann's boundary conditions along the south, north, and west sides:

```
f(1:nx,1)=f(1:nx,2);
f(1:nx,ny)=f(1:nx,ny-1);
f(1,1:ny)=f(2,1:ny);
```

There is no need to apply explicitly the boundary conditions along the east side, because they have been already fixed (they are steady-state Dirichlet boundary conditions). Only in case of Dirichlet boundary conditions changing in time, it would be necessary update them.

We can now advance 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_2d.mp4'
videompg4 = VideoWriter(video_name, 'MPEG-4');
open(videompg4);
```

For graphical purposes only, it is convenient to define two vectors corresponding to the grid points in the two directions:

```
xaxis = 0:hx:lengthx;
yaxis = 0:hy:lengthy;
```

At each time step it is convenient to plot the surface corresponding to the current solution:

```
% Graphics only
hold off;
mesh(xaxis, yaxis, f');
axis([0 lengthx 0 lengthy 0 1.25]);
xlabel('x'); ylabel('y'); zlabel('f');
message = sprintf('time=%d\n', t);
time = annotation('textbox',[0.15 0.8 0.15 0.15],'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
%-------------------------------------------------------------------------%
nx=33;                  % number of grid points along x
ny=33;                  % number of grid points along y
nstep=150;              % number of time steps
lengthx=2.0;            % domain length along x [m]
lengthy=2.0;            % domain length along y [m]
D=0.025;                % diffusion coefficient [m2/s]
u=-1;                   % velocity along x [m/s]
v= 0;                   % velocity along y [m/s]
fin=1;                  % inlet value of f

% Pre-processing of user-defined data
%-------------------------------------------------------------------------%
% Calculate grid steps
hx=lengthx/(nx-1);      % grid step along x [m]
hy=lengthy/(ny-1);      % grid step along y [m]

% Numerical setup: time step (stability conditions)
sigma = 0.75;                       % safety coefficient
dt_diff  = 1/4*hx*hy/D;             % diffusion [s]
dt_conv = 4*D/(u^2+v^2);            % convection [s]
dt = sigma*min(dt_diff, dt_conv); % time step [s]

% Memory allocation
f=zeros(nx,ny);     % current numerical solution
fo=zeros(nx,ny);    % previous numerical solution

% Dirichlet boundary conditions along the east side
f(nx, ny*1/3:ny*2/3) = fin;

% Definition of rectangular mesh (graphical purposes only)
xaxis = 0:hx:lengthx;
yaxis = 0:hy:lengthy;

% Video setup
%-------------------------------------------------------------------------%
video_name = 'advection_diffusion_2d.mp4';
videompg4 = VideoWriter(video_name, 'MPEG-4');
open(videompg4);

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

```
    % Plot the current solution
    hold off;
    mesh(xaxis, yaxis, f');
    axis([0 lengthx 0 lengthy 0 1.25]);
    xlabel('x'); ylabel('y'); zlabel('f');
    message = sprintf('time=%d\n', t);
    time = annotation('textbox',[0.15 0.8 0.15 ...
       0.15],'String',message,'EdgeColor','none');
    frame = getframe(gcf);
    writeVideo(videompg4,frame);
    delete(time);

    % Forward Euler method
    fo=f;
    for i=2:nx-1
        for j=2:ny-1
            f(i,j) = fo(i,j)...
                    -(0.5*dt*u/hx)*(fo(i+1,j)-fo(i-1,j))...
                    -(0.5*dt*v/hy)*(fo(i,j+1)-fo(i,j-1))...
                    +(D*dt/hx^2)*(fo(i+1,j)-2*fo(i,j)+fo(i-1,j))...
                    +(D*dt/hy^2)*(fo(i,j+1)-2*fo(i,j)+fo(i,j-1));
        end
    end

    % Boundary conditions (Neumann's only)
    f(1:nx,1)=f(1:nx,2);          % south
    f(1:nx,ny)=f(1:nx,ny-1);      % north
    f(1,1:ny)=f(2,1:ny);          % west

    % New time step
    t=t+dt;

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

Play the video plotting the solution

```
implay(video_name);
```

4. Exercises

1. Analyze the effect of diffusion coefficient on the solution.
2. Calculate the solution corresponding to the following new boundary conditions along the east side:

$$f\left(x = L_x; y \in \left[\frac{L_y}{5}; \frac{2L_y}{5}\right] \cup \left[\frac{3L_y}{5}; \frac{4L_y}{5}\right]\right) = f_{\text{inlet}}$$

$$f\left(x = L_x; y \in \left[0; \frac{L_y}{5}\right] \cup \left[\frac{2L_y}{5}; \frac{3L_y}{5}\right] \cup \left[\frac{4L_y}{5}; L_y\right]\right) = 0$$

Poisson equation in 2D

1. Case description

We want to numerically solve the **Poisson equation on a 2D rectangular domain** using the Successive Over-Relaxation method:

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = S\left(x, y\right)$$

Using the standard **finite differences** to discretize, we have:

$$\frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{h_x^2} + \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{h_y^2} = S_{i,j}$$

Solving for $f_{i,j}$, we have:

$$f_{i,j} = \frac{1}{2\left(h_x^2 + h_y^2\right)} \left[h_y^2 \left(f_{i+1,j} + f_{i-1,j}\right) + h_x^2 \left(f_{i,j+1} + f_{i,j-1}\right) - h_x^2 h_y^2 \; S_{i,j}\right]$$

In particular, in case of same grid step along the two directions (i.e. $h_x = h_y = h$), the equation above becomes:

$$f_{i,j} = \frac{1}{4} \left[f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1} - h^2 \; S_{i,j}\right]$$

a. The Jacobi method

According to the Jacobi method, we solve for $f_{i,j}$ and use the right hand side to compute a new value. If we denote the old values by α and the new ones with $\alpha + 1$, we have:

$$f_{i,j}^{\alpha+1} = \frac{1}{2\left(h_x^2 + h_y^2\right)} \left[h_y^2 \left(f_{i+1,j}^{\alpha} + f_{i-1,j}^{\alpha}\right) + h_x^2 \left(f_{i,j+1}^{\alpha} + f_{i,j-1}^{\alpha}\right) - h_x^2 h_y^2 \; S_{i,j}^{\alpha}\right]$$

The Jacobi method is very simple, but many iterations are required to reach an accurate solution.The iteration must be carried out until the solution is sufficiently accurate. To measure the error, define the **residual**:

$$R_{i,j}^{\alpha} = \frac{f_{i+1,j}^{\alpha} - 2f_{i,j}^{\alpha} + f_{i-1,j}^{\alpha}}{h_x^2} + \frac{f_{i,j+1}^{\alpha} - 2f_{i,j}^{\alpha} + f_{i,j-1}^{\alpha}}{h_y^2} - \; S_{i,j}^{\alpha}$$

At steady-state the residual should be zero. The pointwise residual or the average absolute residual can be used, depending on the problem. Often, simpler criteria, such as the change from one iteration to the next, are used.

b. The Gauss-Siedler method

Although the Jacobi iteration is a very robust iteration technique, it converges **very slowly**.We therefore seek a way to **accelerate** the convergence to steady-state, making use of the fact that it is only the steady-state that is of interest. Here we introduce the **Gauss-Seidler** method.

In the Gauss-Siedler method, the Jacobi iteration is improved by using new values as soon as they become available:

$$f_{i,j}^{\alpha+1} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left[h_y^2\left(f_{i+1,j}^{\alpha}+f_{i-1,j}^{\alpha+1}\right)+h_x^2\left(f_{i,j+1}^{\alpha}+f_{i,j-1}^{\alpha+1}\right)-h_x^2h_y^2\,S_{i,j}^{\alpha}\right]$$

From a programming point of view, Gauss-Seidleriteration is even simpler than Jacobi iteration since only one vector with f values is needed.

c. The Successive Over-Relaxation (SOR) method

The Gauss-Seidler iteration can be accelerated even further by various acceleration techniques. The simplest one is the Successive Over-Relaxation (SOR) iteration:

$$f_{i,j}^{\alpha+1} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left[h_y^2\left(f_{i+1,j}^{\alpha}+f_{i-1,j}^{\alpha+1}\right)+h_x^2\left(f_{i,j+1}^{\alpha}+f_{i,j-1}^{\alpha+1}\right)-h_x^2h_y^2\,S_{i,j}^{\alpha}\right]\beta+f_{i,j}^{\alpha+1}\left(1-\beta\right)$$

The SOR iteration is very simple to program, just as the Gauss-Seidler iteration. The user must select the coefficient. It must be bounded by $1<\beta<2$. $\beta=1.5$ is usually a good starting value.

2. Implementation

a. Numerical setup

We start defining the relevant data, i.e. the lengths of the sides and the number of grid points along the two directions, the maximum number of iterations, and the over-relaxation coefficient:

```
lengthx=2.0;              % domain length along x [m]
lengthy=2.0;              % domain length along y [m]
nx=39;                    % number of grid points along x
ny=39;                    % number of grid points along y
max_iterations=5000;      % max number of iterations
beta=1.;                  % SOR coefficient (1 means Gauss-Siedler)
```

We pre-process the input data (calculation of grid steps, memory allocation, etc.):

```
% Grid step calculation
hx=lengthx/(nx-1);        % grid step along x [m]
hy=lengthy/(ny-1);        % grid step along y [m]

% Memory allocation
f=zeros(nx,ny);           % current solution
S=zeros(nx,ny);           % source term
```

We can now provide the boundary conditions. In this specifc example, we are considering the following Dirichlet condition along the west side ($x = 0$), where f_{inlet} is a constant value (for example $f_{\text{inlet}} = 1$):

$$f\left(x = 0; y \in \left[\frac{L_y}{3}; \frac{2L_y}{3}\right]\right) = f_{\text{inlet}}$$

$$f\left(x = 0; y \in \left[0; \frac{L_y}{3}\right] \cup \left[\frac{2L_y}{3}; L_y\right]\right) = 0$$

The boundary condition above can be coded as:

```
% Boundary conditions (west side)
f(1, ny*1/3:ny*2/3) = 1;
```

Along the remaining sides, we simply fix the following Dirichlet boundary conditions:

$$f = 0$$

There is no need to explicitly fix the boundary conditions, because we simply ask for $f = 0$, which is automatically satisfied since we allocated and initialized f above.

The figure below better clarifies the boundary conditions we are implementing.

b. Iteration loop

We can now proceed with the numerical solution, by applying the SOR method. At each iteration, we have:

$$f_{i,j}^{\alpha+1} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left[h_y^2\left(f_{i+1,j}^{\alpha} + f_{i-1,j}^{\alpha+1}\right) + h_x^2\left(f_{i,j+1}^{\alpha} + f_{i,j-1}^{\alpha+1}\right) - h_x^2 h_y^2\, S_{i,j}^{\alpha}\right]\beta + f_{i,j}^{\alpha+1}\left(1-\beta\right)$$

The corresponding code is reported in the following (notice that only internal points are updated):

```
for i=2:nx-1
        for j=2:ny-1
             f(i,j)= beta/(2*(hx^2+hy^2))*...
                    (   (f(i+1,j)+f(i-1,j))*hy^2 ...
                      + (f(i,j+1)+f(i,j-1))*hx^2 ...
                      - hx^2*hy^2*S(i,j) ...
                    ) + ...
                    (1.0-beta)*f(i,j);
        end
end
```

In order to check the convergence, we calculate the current residual:

```
res=0;
for i=2:nx-1
     for j=2:ny-1
          res=res+abs( (f(i+1,j)-2*f(i,j)+f(i-1,j))/hx^2 + ...
                       (f(i,j+1)-2*f(i,j)+f(i,j-1))/hy^2 - S(i,j) );
     end
end
```

If the residual is below a user-defined threshold, we can stop the iterations.

c. Graphical output

At the end of iterations, we can plot the solution. For example:

```
xaxis = 0:hx:lengthx;
yaxis = 0:hy:lengthy;
contour(xaxis, yaxis, f');
```

3. Bringing it All Together

Let's bring it all together.

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

% User-defined data
%------------------------------------------------------------------------%
lengthx=2.0;              % domain length along x [m]
lengthy=2.0;              % domain length along y [m]
nx=39;                    % number of grid points along x
ny=39;                    % number of grid points along y
max_iterations=5000;      % max number of iterations
beta=1.;                  % SOR coefficient (1 means Gauss-Siedler)

% Pre-processing of user-defined data
%------------------------------------------------------------------------%
% Grid step calculation
hx=lengthx/(nx-1);        % grid step along x [m]
hy=lengthy/(ny-1);        % grid step along y [m]

% Memory allocation
f=zeros(nx,ny);           % current solution
S=zeros(nx,ny);           % source term

% Boundary conditions (west side)
f(1, ny*1/3:ny*2/3) = 1;

% Iterations
%------------------------------------------------------------------------%
for l=1:max_iterations

    for i=2:nx-1
        for j=2:ny-1
            f(i,j)= beta/(2*(hx^2+hy^2))*...
                    (   (f(i+1,j)+f(i-1,j))*hy^2 ...
                      + (f(i,j+1)+f(i,j-1))*hx^2 ...
                      - hx^2*hy^2*S(i,j) ...
                    ) + ...
                    (1.0-beta)*f(i,j);
        end
    end
```

```
    % Residual
    res=0;
    for i=2:nx-1
        for j=2:ny-1
            res=res+abs( (f(i+1,j)-2*f(i,j)+f(i-1,j))/hx^2 + ...
                         (f(i,j+1)-2*f(i,j)+f(i,j-1))/hy^2 - S(i,j) );
        end
    end

    tot_res = res/((nx-2)*(ny-2));
    fprintf('Iteration: %d - Residual: %e\n', l, tot_res);

    if (tot_res < 0.001)
        break;
    end
end
% Graphical output
%--------------------------------------------------------------------------%
xaxis = 0:hx:lengthx;
yaxis = 0:hy:lengthy;
contour(xaxis, yaxis, f');
```

4. Exercises

1. Identify the optimal value of β, i.e. the value which minimizes the number of iterations.
2. Demonstrates that the method numerical discretization adopted is 2nd order accurate in space.
3. Solve a new case, in which the boundary conditions are $f = 0$ everywhere (on the west side too) and the source term has the following expression:

$$S(x, y) = (x - 1)^2 + (y - 1)^2$$