This has already been graded, go to page 5 and Appendix B.

# AMATH 481/581 - Autumn 2022 Homework #3

**Presentation skill: discussing problems from a physical perspective.**

Wietse Vaes 222448816

## Introduction

The transport of certain substances is an important aspect of life. Therefore it would be useful if this could be modelled. One way of doing this is through the one-dimensional advection equation, otherwise known as the transport equation, given by

$$u_t + c(x,t)u_x = 0, \qquad \text{for } -\infty < x < \infty, t > 0. \tag{1}$$

Given an initial condition $u(x,0) = u_0$, the function $u(x,t)$ is the specific quantity of the substance which we would like to model under the influence of a velocity field, $c$. This field can be constant or dependant on the temporal and/or spatial field. Since this is the only parameter in this equation questions can arise such as: how does this affect the solution for specific $c$?

## Results and discussion

Looking at the advection equation from a physical perspective, $u_t$ denotes the change of $u$ in time and $u_x$ the spatial change of $u$. Therefore, since

$$u_t = -cu_x,$$

the velocity field $c$ indicates the rate at which $u_t$ changes over $u_x$. Because of the minus sign indicates that it changes in the opposite direction. Another way of looking at it, is to look at a certain point $(x^*, t^*)$ where $u(x^*, t^*) = u^*$. If time increases, the spatial point where $u$ reaches $u^*$ changes at rate $c$.

Now we look at two concrete examples of the velocity field. Given that the initial condition is

$$u(x,0) = e^{-(x-5)^2},$$

a Gaussian-like function. We will consider two velocity fields:

$$c(x,t) = -0.5, \text{ and } c(x,t) = -(1 + 2\sin(5t) - H(x-4)),$$

with $H(x)$ the heaviside function.

Using fourth-order Runga-Kuttah and the second-order central difference method over a spatial grid of 200 equally spaced point for $-10 \leq x \leq 10$ and periodic boundary conditions, we can get an approximation of the solution of the transport equation (1) for $t \in [0, 10]$. For the first velocity field the solution can be seen in figure 1.

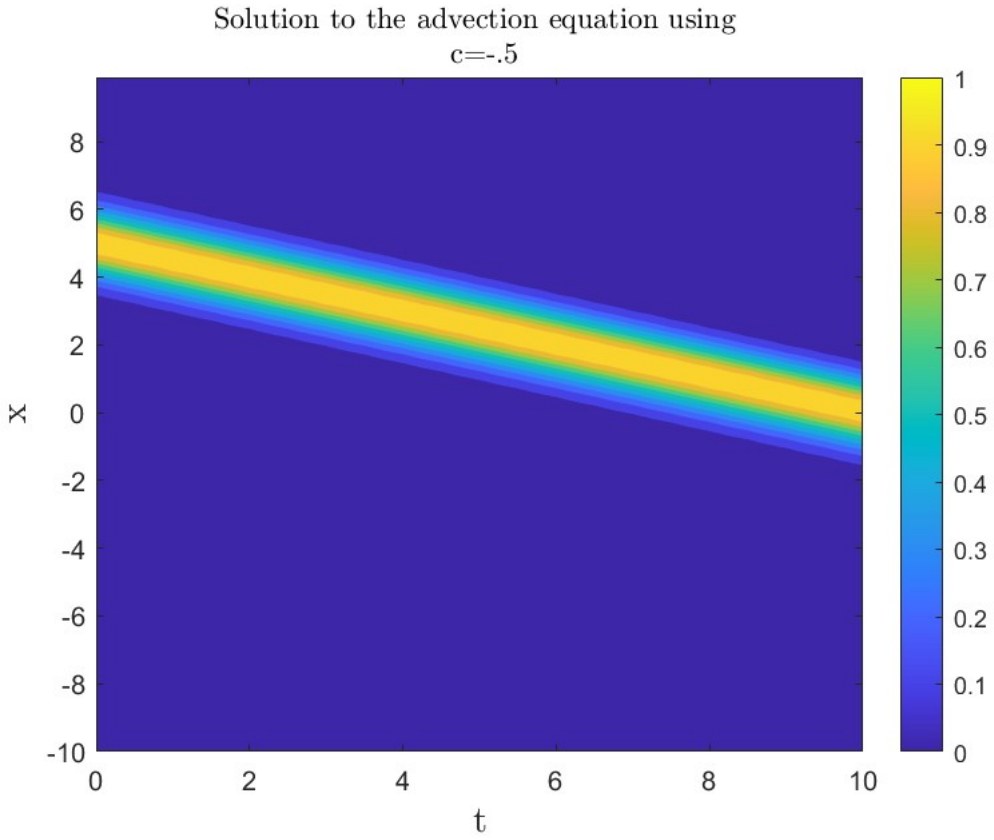


Figure 1: An approximation of the solution to the advection equation with velocity field $c = -0.5$.

It shows exactly what we expected. The function, a sort of wave, moves down, $x$ decreases, as time increases. Since $c$ is negative, $x$ decreases. The speed and direction of change does not change since $c$ is constant.

A more interesting case of $c$ is when $c(x,t) = -(1 + 2\sin(5t) - H(x-4))$, since it is time and space dependant. The solution to this can be seen in Figure 2.

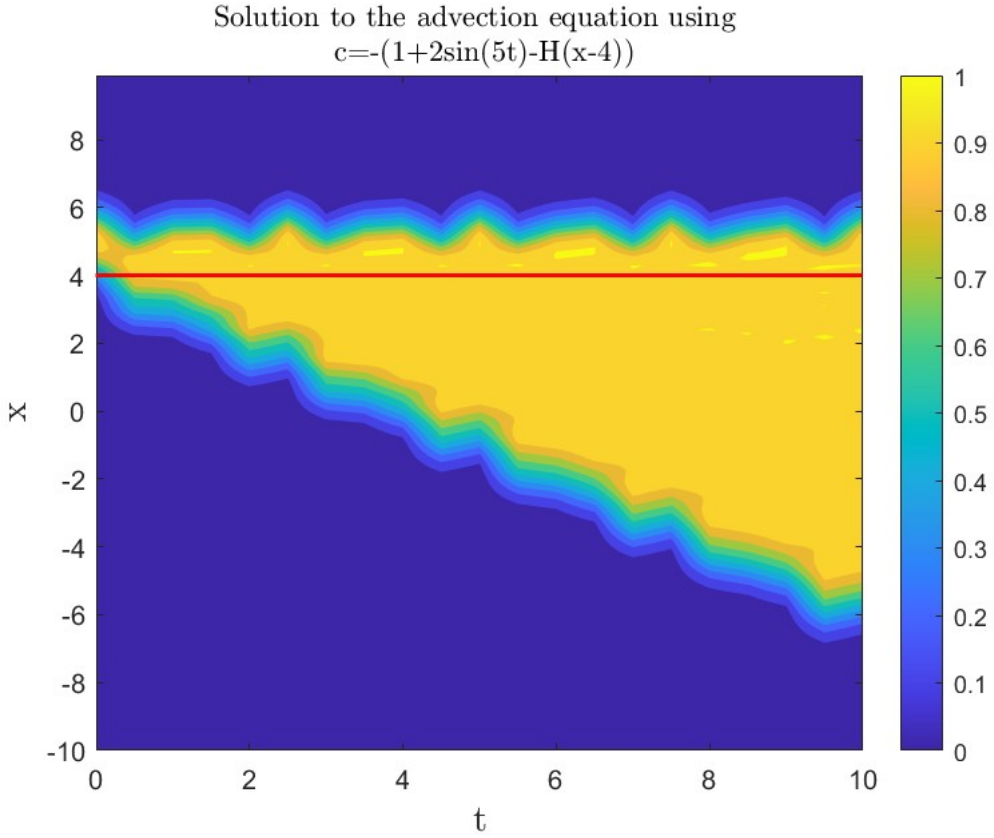


Figure 2: An approximation of the solution to the advection equation with velocity field $c = -(1 + 2\sin(5t) - H(x-4))$. A red line is made at $x = 4$ to signify where the Heaviside function changes from 0 to 1.

The red line on the figure is where $x = 4$. This signals where the Heaviside function starts contributing to the velocity field. Looking at the time before the non-zero part of the function hits the red line, a clear wave can be seen. Once it hit the red line it fragments into a fan, the only explanation for this behavior is the discontinuity in $c$. The time dependence of the velocity field is also clearly visible. Defining the "edges" of a function as the place where a lot of change happens, a sort of periodic behavior can be seen on the edges of the function. Since $-1 \leq \sin(x) \leq 1$, the velocity field can be both positive and negative. This means that the function can move up and down the $x-$axis, which can clearly be seen in the wavy pattern at the edges.

# Conclusion

The advection equation (1) tries to model the transport of substances over time. Here the velocity field $c$ influences the solution of the equation a great deal. A point, where $u(x,t) = u^*$, changes spatially at rate $c$ when time changes. Next to it being able to be constant, it can also be dependent on it's spatial and temporal location. Clear influences can be seen when approximating and plotting the solution. This analysis was, however, hand wavy. A more correct approach to knowing how $c$ will influence the solution is to look at it in a more mathematical standpoint. Other velocity fields could be considered, but this would result in another hand wavy analysis.

# AMATH 481/581 - Autumn 2022 Homework #3

**Presentation skill: discussing problems from a mathematical perspective (retake).**

Wietse Vaes 222448816

## Introduction

Approximating the solution to a partial differential equation using a computer takes time. The amount of time depends on multiple factors, for example, the specific PDE, the number of variables, the grid size of the discretization, the methods used, your computer's characteristics,.... Here we will look at how long it takes to approximate the vorticity $\omega(x, y, t)$ and stream function $\psi(x, y, t)$ using the governing equations

$$\omega_t + [\psi, \omega] = \nu^2 \omega,$$

with $\nu$ the diffusion parameter and where $[\psi, \omega] = \psi_x \omega_y - \psi_x \omega_x$ and $\nabla^2 = \partial_x^2 + \partial_y^2$. Furthermore, the streamfunction satisfies

$$\nabla^2 \psi = \omega.$$

Here the domain is discritized and the derivative operators are approximated using second-order central difference formula. Using a built-in $RK4$ function in MATLAB we approximate the solutions to these PDEs. Say that we approximate $\nabla^2$ by matrix $A$, the streamfunction then satisfies

$$A\psi = \omega.$$

We thus need to solve a system to get the voriticity $\omega$. This can be done using Gaussian Elimination or the LU decomposition, both take a different amount of time to compute and this is what we will focus on.

## Results and discussion

Say that we have a matrix $A$ of dimension $N \times N$ for the system $Ax = b$. The theoretical time it takes to solve this system, after calculating all the matrices, for the Gaussian Elimination method and LU decomposition are, respectively, $\mathcal{O}(N^3)$ and $\mathcal{O}(N^2)$. Thus,

we would assume that the LU decomposition would be faster. This can be empirically tested. We can also find the empirical order of the time spend relative to the matrix size. Say that we have two grid sizes: $N$ and $\tilde{N} = \frac{N}{a}$, with $a$ a positive non-zero number. The time $T$ spent on a method with order $n$ is then

$$T = CN^n, \text{ and } \tilde{T} = C\tilde{N}^n,$$

with $C$ a constant. Therefore, we have that,

$$\log_a \left( \frac{T}{\tilde{T}} \right) = n.$$

With this, we can find the empirical order of time spent on the method. Using code, which can be found in appendix B, we approximated the solution for 64 grid points in $x$ and $y$ direction and 128 grid points. This caused our matrix $A$ to have a dimension of, respectively, $64^2 \times 64^2$ and $128^2 \times 128^2$. Thus, our $N = 128^2$ and $\tilde{N} = \frac{N}{4} = 64^2$. We get the following times for the Gaussian elimination method (GE) and the LU decomposition (LU):

| | 64 | 128 |
|---|---|---|
| GE | 1,44 | 63,95 |
| LU | 0,49 | 31,91 |

Table 1: Time spent (in seconds) for approximating a solution using the Gaussian Elimination method (GE) and LU decomposition (LU) (rows) using a set amount of grid points in the $x$ and $y$ directions (collumns).

Here we clearly see that the LU decomposition was considerably faster, which we expected. Using this, we calculated the order of the methods. Respectively, for the Gaussian elimination method and LU decomposition, we get an order of $2,74$ and $3,01$. The GE method did not give a strange result, but the LU decomposition does, especially the fact that the order of the first method is lower than that of the second.

# Conclusion

Using the LU decomposition results in faster code than using the Gaussian elimination method. This can clearly be seen in the example stated above, however this is but one setting where this is the fact. Empirically finding the order of the speed relative to the

size of matrix $A$ gave us strange results. The order of the LU decomposition was larger than that of the Gaussian elimination method. Once again, we need to note that this was just one setting where this was the case. Solving other systems might result in different orders, where it might agree more with our theoretical findings. In order to truly say which is faster and has a better order, one must use more test cases. Furthermore, there are other methods which might give even faster results. For example, one can look at the QR factorisation. Either way, the LU decomposition did work faster, which is what we expected.

# Appendix

## A Code presentation skill: physical perspective

```
%% Problem 1
clear all
%Parameters
f = @(x) exp(-(x-5).^2);
L = 10;
T = 10;
Deltax = 0.1;

%Discretisation
x = linspace(-L,L-Deltax,200);
N = length(x);

%Calculating A, could be done better with diag
A = sparse(N,N);
for index1 = 1:N
    if index1 ~= 1 && index1 ~= N
        A(index1,index1-1) = -1;A(index1,index1+1) = 1;
    elseif index1 == N
        A(index1,index1-1) = -1;A(index1,1) = 1;
    elseif index1 == 1
        A(index1,index1+1) = 1; A(index1,N) = -1;
    end
end

A = A/(Deltax*2);

%Deliverable
A1=full(A);
```

```
29
30 %Initial condition
31 y0 = f(x);
32
33 %Calculating y without for c constant
34 [t, y1] = ode45(@(t, x) advec1(t, x, A), linspace(0,T,21), y0);
35 A2 = y1';
36
37 %Calculating y without for c nonconstant
38 xx=x;
39 [t, y2] = ode45(@(t, x) advec2(t, x, xx), linspace(0,T,21), y0);
40
41 A3 = y2';
42 %Plotting
43 figure(1);
44 [XX,TT] = meshgrid(t,x);
45 contourf(XX,TT,y1','edgecolor','none'); xlabel('t','Interpreter'
      ,'Latex','FontSize',14);ylabel('x','Interpreter','Latex','
      FontSize',14); title(sprintf('Solution to the advection
      equation using \n c=-.5'),...
46           'Interpreter','Latex'); colorbar();
47 figure(2);
48 contourf(XX,TT,y2','edgecolor','none'); xlabel('t','Interpreter'
      ,'Latex','FontSize',14);ylabel('x','Interpreter','Latex','
      FontSize',14); title(sprintf('Solution to the advection
      equation using \n c=-(1+2sin(5t)-H(x-4))'),...
49           'Interpreter','Latex'); colorbar();
50 hold on; yline(4,'r-','Linewidth',1.5)
51
52 %% Functions
53 function u_t = advec1(t, x, A)
54 c = -0.5;
```

```
55 u_t = -c*A*x;
56 end
57
58 function u_t = advec2(t, x, xx)
59     N = length(xx);
60     A = sparse(N,N);
61     Deltax = 0.1;
62     for index1 = 1:N
63         c = (1-heaviside(xx(index1)-4)+2*sin(5*t));
64         if index1 ~= 1 && index1 ~= N
65             A(index1,index1-1) = -c;A(index1,index1+1) = c;
66         elseif index1 == N
67             A(index1,index1-1) = -c;A(index1,1) = c;
68         elseif index1 == 1
69             A(index1,index1+1) = c; A(index1,N) = -c;
70         end
71     end
72     A = A/(2*Deltax);
73     u_t = A*x;
74 end
75
76 function H = heaviside(x)
77     H = (x > 0) + 0.5*(x == 0);
78 end
```

# B Code presentation skill: mathematical perspective

```
%% Problem 2
for m = [64,128]
    clearvars -except m TIME1 TIME2
    n=m*m;
    e1 = ones(n,1);
    Low1 = repmat([ones(m-1, 1); 0], m, 1);
    Low2 = repmat([1; zeros(m-1, 1)], m, 1);
    Up1 = circshift(Low1, 1);
    Up2 = circshift(Low2, m-1);
    A = spdiags([e1, e1, Low2, Low1, -4*e1, Up1, Up2, e1, e1],
        ...
         [-(n-m), -m, -m+1, -1, 0, 1, m-1, m, (n-m)], n, n);

    A(1,1)=2;

    B = spdiags([e1, -e1, e1, -e1], [-(n-m), -m, m, (n-m)], n, n
        );

    C = spdiags([Low2, -Low1, Up1, -Up2], [-m+1, -1,  1, m-1], n
        , n);

    %Parameters
    nu = 0.001;
    f = @(x,y)  exp(-2*x.^2-(y.^2/20));
    L = 10;
    T = 4;
    Deltax = 20/m;
    Deltat = 0.5;

```

```
27      xx = linspace(-L,L-Deltax,m)'; yy = xx;
28
29      A = A/(Deltax^2);
30      B = B/(2*Deltax);
31      C = C/(2*Deltax);
32
33      y = repmat(yy,m,1); %Creating x and y coordinates
34      x = repmat(xx,1,m)';
35      x=reshape(x,[m^2,1]);
36
37      init = A\f(x,y); %initial condition
38      [L,U,P] = lu(A); % LU decomposition
39
40     %Solving without the LU decomposition
41      tic
42      [t, ppsi1] = ode45(@(t, x) vorticity1(t, x, A,B,C,nu), 0:
           Deltat:T, init);
43      TIME1(m/64) = toc;
44
45     %Solving with the LU decomposition
46      tic
47      [t, ppsi2] = ode45(@(t, x) vorticity2(t, x, A,B,C,nu,L,U,P),
            0:Deltat:T, init);
48      TIME2(m/64) = toc;
49  end
50  log(TIME1(2)/TIME1(1))/log(4)
51  log(TIME2(2)/TIME2(1))/log(4)
52
53
54  %% Functions
55  function ut = vorticity1(t, x, A,B,C,nu)
56      b =  nu*A^2*x+(C*x).*(B*A*x)-(B*x).*(C*A*x);
```

```
57        ut = A\b;
58    end
59    function ut = vorticity2(t, x, A,B,C,nu,L,U,P)
60        b =  nu*A^2*x+(C*x).*(B*A*x)-(B*x).*(C*A*x);
61        ut = U\(L\(P*b));
62    end
```