Imperial College
London

AERO96014 - Introduction to Computational Fluid Dynamics

Coursework 1

Academic Responsible:	Prof. Spencer Sherwin
Department:	Department of Aeronautics
Course:	MEng Aeronautical Engineering
Module:	Computational Fluid Dynamics
Academic year:	2020/2021
Student:	Jo Wayne Tan
CID:	01327317
Personal tutor:	Dr. Francesco Montomoli
Date:	04/12/2021

Department of Aeronautics
South Kensington Campus
Imperial College London
London SW7 2AZ
U.K.

Question 1

1(a)

The Helmholtz equation, $\nabla^2 u - \lambda u = f$, can be written in 1D as:

$$u_{xx} - \lambda u = f \tag{1}$$

The u_{xx} term can be discretised using finite differences centred approximation as

$$u_{xx} = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}$$

where Δx is the distance between adjacent grid points and assuming the grid is equispaced. The Helmholtz equation becomes:

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} - \lambda u_i = f_i$$

If $\lambda = 1$,

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} - u_i = f_i \tag{2}$$

1(b)

Derivatives of the analytical solution, u, in terms of x:

$$u(x) = sin(m\pi x) \tag{3}$$

$$u_x = m\pi cos(m\pi x) \tag{4}$$

$$u_{xx} = -m^2\pi^2 sin(m\pi x) \tag{5}$$

Substitute (3) and (5) into (1) gives

$$f = -m^2\pi^2 sin(m\pi x) - \lambda sin(m\pi x) = -(\lambda + m^2\pi^2) sin(m\pi x) \tag{6}$$

1(c)

The Helmholtz equation is elliptic, which has the global characteristic of information travelling everywhere together. Hence, boundary conditions changes influences the solution everywhere instantaneously, and the equation need to have appropriate boundary conditions for it to be **well posed** in order to obtain a smooth and unique solution. If m = 3, equation (4) becomes $u(x) = sin(3\pi x)$. The Dirichlet boundary conditions would be:

$$u(0) = sin(0) = 0$$

$$u(1) = sin(3\pi) = 0$$

This means that the solution u is always 0 at the boundaries of the domain. The Neumann boundary conditions would be:

$$u_x(0) = 3\pi cos(0) = 3\pi$$

$$u_x(1) = 3\pi cos(3\pi) = -3\pi$$

1(d)

The Thomas Algorithm is used to solve the tridiagonal system resulting from discretisation using Dirichlet boundary conditions.

$$Hu = f$$

$$\begin{bmatrix} \alpha & 1 & 0 & & 0 \\ 1 & \alpha & 1 & 0 & \\ \dots & \dots & \dots & \dots & \dots \\ & 0 & 1 & \alpha & 1 \\ 0 & & 0 & 1 & \alpha \end{bmatrix} \begin{bmatrix} \mathrm{u}_1 \\ \mathrm{u}_2 \\ \dots \\ \mathrm{u}_{N-3} \\ \mathrm{u}_{N-2} \end{bmatrix} = \begin{bmatrix} \mathrm{f}_1 - \frac{u_0}{(\Delta x)^2} \\ \mathrm{f}_2 \\ \dots \\ \mathrm{f}_{N-3} \\ \mathrm{f}_{N-2} - \frac{u_{N-1}}{(\Delta x)^2} \end{bmatrix} \tag{7}$$

where $\alpha = -\frac{2}{(\Delta x)^2} - \lambda$. Equation (7) is the reduced matrix system after enforcing the Dirichlet boundary conditions at x=0 and x=1. The entries of this matrix equation (7) corresponds to:

$$a_k x_{k-1} + b_k x_k + c_k x_{k+1} = f_k \quad \text{for} \quad k = 1, ..., N-2$$

where a_k, b_k, and c_k are the lower diagonal, diagonal, and upper diagonal entries, in this case 1, α, and 1, and $a_1 = c_{N-2} = 0$. The Thomas algorithm solves this system in 2 steps, the Forward step:

$$\beta_1 = b_1 \qquad \beta_k = b_k - a_k \frac{c_{k-1}}{\beta_{k-1}} \text{ for } k = 2, ..., N-2$$

$$\gamma_1 = \frac{f_1}{\beta_1} \qquad \gamma_k = \frac{(-a_k \gamma_k + f_k)}{\beta_k} \text{ for } k = 2, ..., N-2$$

and the Backward step:

$$x_{N-2} = \gamma_{N-2} \qquad x_k = \gamma_k - x_{K+1} \frac{c_k}{\beta_k} \text{ for } k = N-3, N-4, ..., 1$$

These steps are implemented in C++ and can be found in the .cpp script attached, as *SolveThomasAlgorithm* and *SolveThomasAlgorithm2* functions, for solving symmetric and non-symmetric tridiagonal systems respectively. The numerical and exact solutions are computed and compared at every discretised point, and the maximum error for different values of Δx is shown in Table 1 below.

Δx	1/10	1/20	1/50	1/100	1/200	1/500
ϵ_{max}^{direct} (Δx)	7.65E-2	1.85E-2	2.93E-3	7.32E-4	1.83E-4	2.93E-5

Table 1: Maximum error using Thomas algorithm.

1(e)

The steepest descent method is an iterative algorithm for solving a system of linear equations of the form $Ax = b$, where **A** is a symmetric positive-definite matrix. Hence, it can be used to solve the problem presented in 1(d), as a tridiagonal system fulfills this condition. An iterative method is equivalent to minimising the function:

$$f(x) = \frac{1}{2} x^T A x - x^T b$$

The residual at the k^{th} step is then equivalent to $r_k = Ax_k - f$. This algorithm is implemented in C++ as the *SolveSteepestDescent* function using banded storage and BLAS routines. The maximum error for different values of Δx are computed and presented in Table 2 below.

Δx	1/10	1/20	1/50	1/100	1/200	1/500
ϵ_{max}^{iter} (Δx)	7.65E-2	1.85E-2	2.93E-3	7.32E-4	1.83E-4	2.93E-5

Table 2: Maximum error using Steepest Descent Iterative algorithm.

1(f)

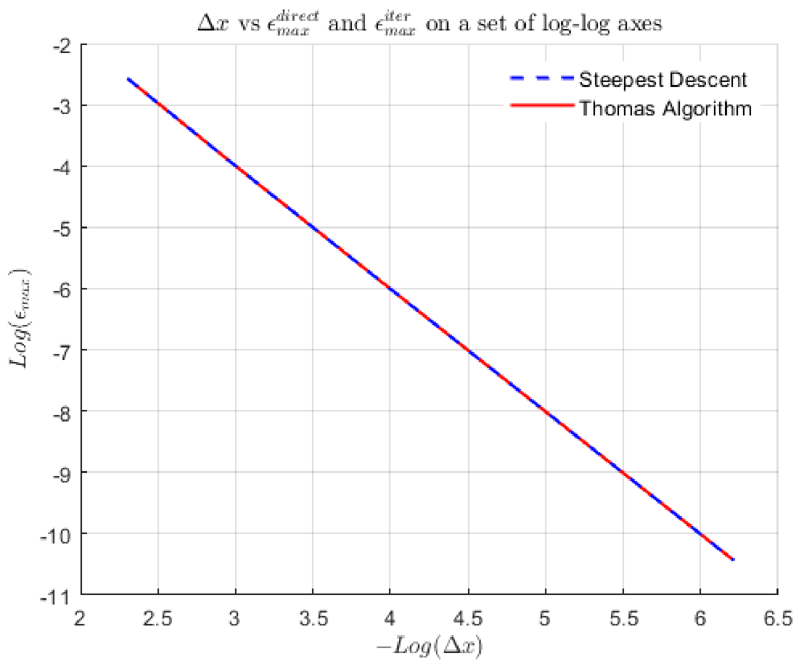

Figure 1: Plot for question 1(f).

The maximum error values of Thomas algorithm and Steepest descent are exactly the same up to more than 5 significant figures overall, for all Δx values. This means that the direct and iterative methods produced results of same accuracy and precision for this problem, but also because both algorithms used the same discretised Δx points to generate the error outputs. A $-log(\Delta x)$ axes is used to show that the maximum error decreases as Δx decreases from left to right, as shown in Figure 1, because the grid gets finer and finer. The reason for plotting the variables as a log-log graph is because it shows the linear relation of their respective powers. This can be shown by the gradient of the plots in Figure 1. They are $\simeq -2$, which corresponds to the second order accurate in space scheme (O(Δx^2)) used here, as the maximum error decreases quadratically with the decrease in Δx. More precisely, the decrease in orders of magnitude of maximum error is **twice** the decrease in orders of magnitude of Δx. If we look at the data provided in Table 1 and 2, when any Δx is reduced to a tenth of its original value (ex. 1/50 to 1/500), its corresponding maximum error is reduced to a hundredth (square of a tenth) of the original value (2.93E-3 to 2.93E-5).

Question 2

2(a)

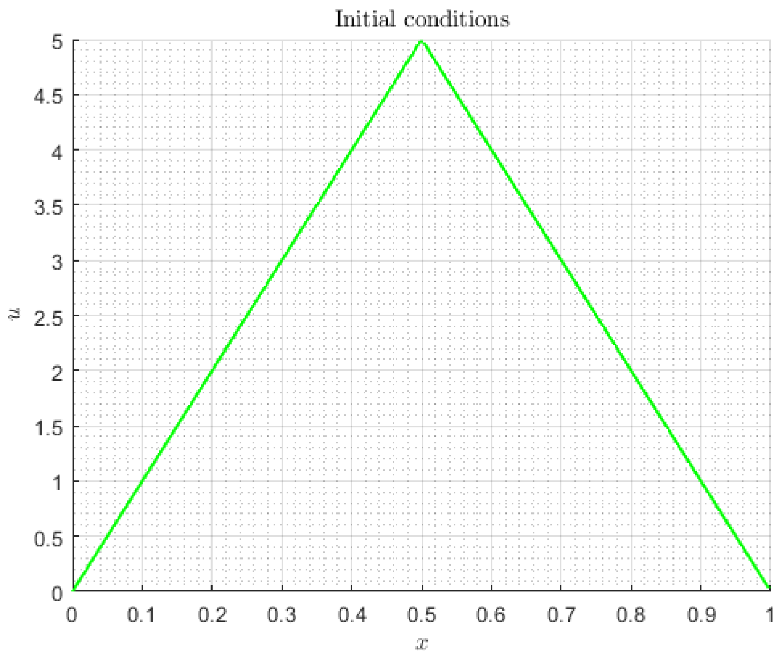

Figure 2: Initial Conditions plot, u(x, 0).

2(b)

The Neumann boundary condition at x=1 is:

$$u_x(1,t) = \sum_{m=0}^{m=50} (-1)^m \frac{40}{(2m+1)^2\pi^2} e^{-\sigma(2m+1)^2\pi^2 t}(2m+1)\pi cos[(2m+1)\pi] \tag{8}$$

2(c)

Using Equation (2) from the **problem sheet** and an implicit centred approximation to the 2nd derivative, we get:

$$\frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2} - \frac{1}{\sigma\Delta t}u_i^{n+1} = -\frac{1}{\sigma\Delta t}u_i^n$$

Multiplying $\sigma\Delta t$ to both the LHS and RHS of this equation and rearranging, the equation above becomes:

$$-\lambda u_{i+1}^{n+1} + (1+2\lambda)u_i^{n+1} - \lambda u_{i-1}^{n+1} = u_i^n$$

where $\lambda = \frac{\sigma\Delta t}{\Delta x^2}$ is the gain factor. To impose the Dirichlet and Neumann conditions, $u(0,t) = \alpha = 0$ and $u_x(1,t) = \beta$ (see Equation (8)), we use a backward difference formula to approximate the derivative u_x,

$$u_x|_{N-1} \approx \frac{u_{N-1} - u_{N-2}}{\Delta x} \ , \ -u_{N-2} + u_{N-1} = \Delta x \ u_x|_{N-1} = \Delta x \ \beta$$

where β is the value of $u_x(1,(n+1)\Delta t)$ (using Equation (8)) at each time step. Then the approximate tridiagonal matrix system would be:

$$Hu^{n+1} = u^n$$

$$\begin{bmatrix} (1+2\lambda) & -\lambda & & & & \\ -\lambda & (1+2\lambda) & -\lambda & & & \\ & \dots & \dots & \dots & & \\ & & -\lambda & (1+2\lambda) & -\lambda & \\ & & & -\lambda & (1+2\lambda) & -\lambda \\ & & & & -1/\Delta x & 1/\Delta x \end{bmatrix} \begin{bmatrix} u_1 \\ \dots \\ u_{N-2} \\ u_{N-1} \end{bmatrix}^{n+1} = \begin{bmatrix} u_1 - \alpha \\ u_2 \\ \dots \\ u_{N-3} \\ u_{N-2} \\ \beta \end{bmatrix}^{n} \tag{9}$$

where $u_{N-1} = \beta$. The finite difference implementation of the Neumann condition destroyed the symmetry of the tridiagonal matrix as shown above, hence this problem is solved using the *SolveThomasAlgorithm2* function in C++. The problem is solved using a for loop iterating through time $t = \Delta t$ to $t = T$, starting with number of time steps, n = 0, where u^0 equals to the Initial Condition in question 2(a), to solve for u^1 and so on, replacing u^n with u^{n+1} after every time step. The maximum error for the same values of Δx as 1(d) are computed and shown in Table 3 below.

Δx	1/10	1/20	1/50	1/100	1/200	1/500
$\epsilon_{max}^{direct}(\Delta x)$	1.02E-2	2.64E-3	8.61E-4	5.90E-4	6.14E-4	6.25E-4

Table 3: Maximum error using Thomas algorithm.

2(d)

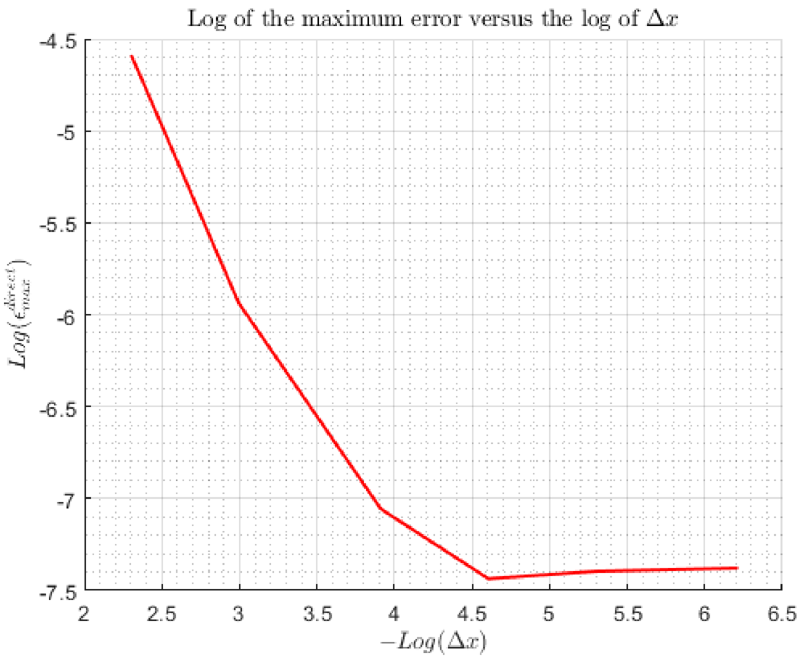

Figure 3: Plot for question 2(d).

Figure 3 shows the maximum error between the numerical (Thomas algorithm) and exact solution of the diffusion equation for values of Δx, as shown in Table 3. A $-log(\Delta x)$ axes is used to show the increase in grid fineness/quality from left to right. The direct maximum error decreases rapidly from $\Delta x = 1/10$ to 1/100, but increases slightly from this point onwards, for 1/200 and 1/500. The error was expected to decrease in a linear fashion just like the plot in answer 1(f), because the truncation error is of second order in space as well. However, the gradient of this error output plot in Figure 3 increases as Δx decreases, and actually becomes positive pass the 1/100 point. This is because the maximum error of a finite difference scheme does not consist purely of truncation error (although by definition it literally means the difference between numerical and exact solution), but also actually contains the rounding error. This rounding error is the characteristic feature of floating-point computation, due to approximate representation of values with a finite number of bits. This physically induced limitation/error becomes very significant as number of computation increases exponentially due to the increase in number of grid points multiplied by solving at each and every time step. At the $\Delta x = 1/100$ point the gain in rounding error begins to overtake the advantage of using a finer grid - reduction of truncation error - hence the overall/total error starts to increase.

2(e)

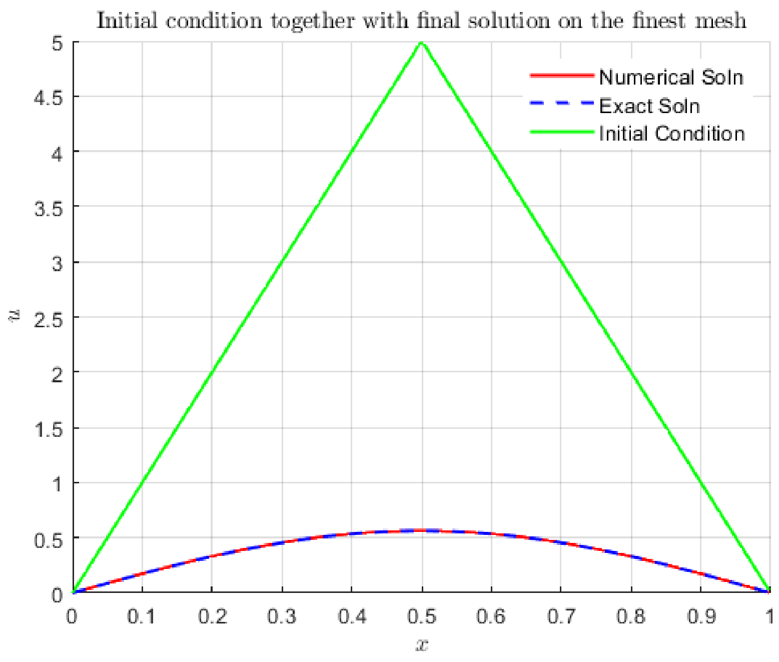

Figure 4: Plot of final solution.

Figure 4 shows a physical interpretation of diffusive decay in our diffusion equation, which is an elliptic Helmholtz equation. This is a feature of a time-dependent solution, because σ in our problem is 4 (> 0 indicating positive diffusion), hence the solution decays to a steady state over time. This physically represents smoothing, which stabilises the problem as shown in our plot.