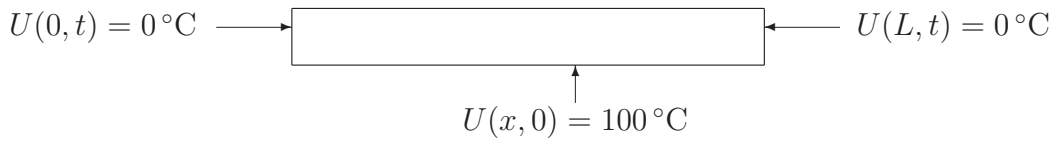
Chapter 2

Explicit methods for 1-D heat or diffusion equation

We will focus on the heat or diffusion equation for the next few chapters. This is an example of a parabolic equation.

2.1 Analytic solution: Separation of variables

First we will derive an analtical solution to the 1-D heat equation. Consider the temperature $U(x,t)$ in a bar where the temperature is governed by the heat equation, $U_t = \beta U_{xx}$. The ends of the bar are cooled to 0 °C and the initial temperature of the bar is 100 °C.



We want to solve $U_t = \beta U_{xx}$ using separation of variables. We assume that the solution can be written as the product of a function of x and a function of t, ie. $U(x,t) = X(x)T(t)$ then:

$$\begin{aligned} U_t &= \frac{\partial T}{\partial t} X = \beta T \frac{\partial^2 X}{\partial x^2} = \beta U_{xx} \Rightarrow \text{ divide by XT} \\ \underbrace{\frac{1}{\beta}\frac{T'(t)}{T(t)}}_{\text{function of } t \text{ only}} &= \underbrace{\frac{X''(x)}{X(x)}}_{\text{function of } x \text{ only}} = \underbrace{-\lambda^2}_{\text{constant}} \end{aligned} \tag{2.1}$$

The only way the LHS and RHS of equation 2.1 can be a function of t and x respectively is if they are both equal to a constant which we define to be $-\lambda^2$ for convenience.

$$T' + \lambda^2 \beta T = 0 \quad \Rightarrow \quad T = e^{-\lambda^2 \beta t}$$
$$X'' + \lambda^2 X = 0 \quad \Rightarrow \quad X = A \sin \lambda x + B \cos \lambda x$$

Use boundary conditions $U(t,0) = 0 \quad \Rightarrow \quad X(0) = 0 = B$

$$U(t,L) = 0 \quad \Rightarrow \quad X(L) = 0 = A \sin \lambda$$
$$t \Rightarrow \quad \lambda = \lambda_n = \frac{n\pi}{L}, \quad n = 1, 2, \ldots$$

$$\begin{aligned} U(x,t) &= X(x)T(t) \\ &= \sum_{n=1}^{\infty} An \sin(\frac{n\pi x}{L}) e^{-\lambda^2 \beta t} \end{aligned}$$

$e^{-\lambda^2 \beta t}$ is a transient solution and decays in time to boundary conditions.
Use initial conditions $U(0,x) = 100\,^{\circ}\mathrm{C}$ to find A_n :

$$U(0,x) = T_0 = \sum_{n=1}^{\infty} A_n \sin(n\pi x/L)$$

Use orthogonality: $\int_0^L \sin(\frac{n\pi x}{L}) \sin(\frac{m\pi x}{L}) dx = \delta_{nm}$
and $\cos(m\pi) - 1 = 0$, for $m = 0, 2, 4, \ldots$
and $\cos(m\pi) - 1 = -2$, for $m = 1, 3, 5, \ldots$

$$\begin{aligned} \Rightarrow A_m &= T_0 \left[-L/m\pi(\cos(m\pi) - 1)\right] \\ &= \frac{-2L}{m\pi} T_0 \end{aligned}$$

for m=1,3,5,...

2.2 Numerical solution of 1-D heat equation

2.2.1 Difference Approximations for Derivative Terms in PDEs

We consider $U(x,t)$ for $0 \leq x \leq a, \qquad 0 \leq t \leq T$
Discretise time and spatial variable x:

$$\Delta t = \frac{T}{m}, \qquad \Delta x = \frac{a}{n+1},$$

$$t_k = k\Delta t, \quad 0 \leq k \leq m \quad x_j = j\Delta x, \quad 0 \leq j \leq n+1$$

Let $U_j^k = U(x_j, t_k)$
Consider Taylor series expansion for U_j^{k+1}:

$$U_j^{k+1} = U_j^k + \Delta t \frac{\partial U_j^k}{\partial t} + \frac{\Delta t^2}{2} \frac{\partial^2 U_j^k}{\partial t^2} + \mathrm{O}(\Delta t^3) \tag{2.2}$$

If we only consider $\mathrm{O}(\Delta t)$ terms in equation 2.2 then we arrive at the forward difference in time approximation for U_t:

$$\frac{\partial U_j^k}{\partial t} = \frac{U_j^{k+1} - U_j^k}{\Delta t} + \mathrm{O}(\Delta t)$$

We can also derive a higher order approximation for U_t if we consider the Taylor series expansion for U_j^{k-1} as well:

$$U_j^{k-1} = U_j^k - \Delta t \frac{\partial U_j^k}{\partial t} + \frac{\Delta t^2}{2} \frac{\partial^2 U_j^k}{\partial t^2} + \mathrm{O}(\Delta t^3) \tag{2.3}$$

$2.2 - 2.3 \Rightarrow \dfrac{\partial U_j^k}{\partial t} = \dfrac{U_j^{k+1} - U_j^{k-1}}{2\Delta t} + \mathrm{O}(\Delta t^2) \Rightarrow$ leap-frog (or centred difference) in time.

This gives higher order accuracy than forward difference.

We can also perform similar manipulations to arrive at approximations for the second derivative U_{tt}:

$$2.2 + 2.3 - 2U_j^k \Rightarrow \frac{\partial^2 U_j^k}{\partial t^2} = \frac{U_j^{k+1} + U_j^{k-1} - 2U_j^k}{\Delta t^2} + \mathrm{O}(\Delta t^2) \Rightarrow \text{central difference}$$

The finite difference method makes use of the above approximations to solve PDEs numerically.

2.2.2 Numerical solution of 1-D heat equation using the finite difference method

$$U_t = \beta U_{xx}$$

Initial conditions

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

Types of boundary conditions

Neumann boundary conditions

$$\begin{aligned} U_x(t, 0) &= g_1(t) \\ U_x(t, a) &= g_2(t) \end{aligned}$$

Dirichlet boundary conditions

$$\begin{aligned} U(t, 0) &= g_1(t) \\ U(t, a) &= g_2(t) \end{aligned}$$

Mixed boundary conditions

$$\begin{aligned} U(t, 0) &= g_1(t) \\ U_x(t, a) &= g_2(t) \end{aligned}$$

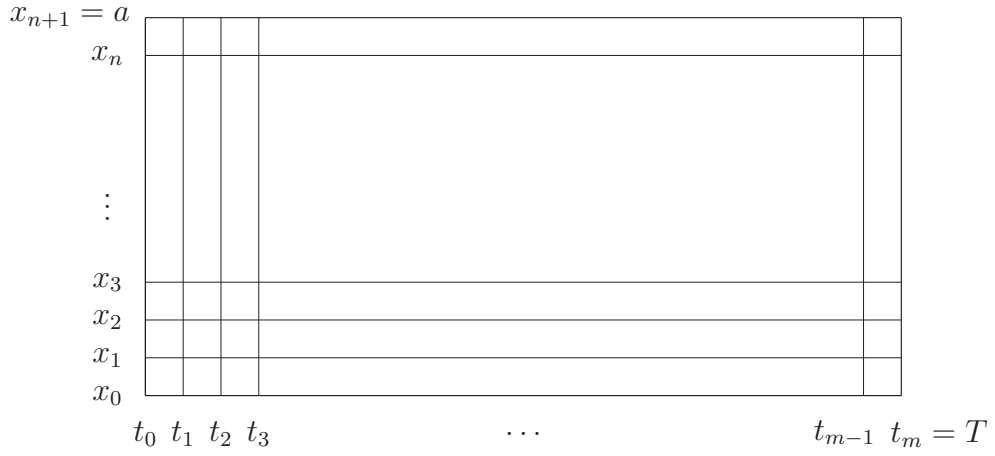
2.2.3 Explicit Forward Euler method

or FTCS (Foward Time Centred Space)

We want to solve the 1-D heat equation:

$$U_t = \beta U_{xx}. \tag{2.4}$$

We solve this PDE for points on a grid using the finite difference method where we discretise in x and t for $0 \leq x \leq a$ and $0 \leq t \leq T$:



We discretise in time with time step: $\Delta t = T/m$ and in space with grid spacing: $\Delta x = a/(n+1)$, and let $t_k = k\Delta t$ where $0 \leq k \leq m$ and $x_j = j\Delta x$ where $0 \leq j \leq n+1$.

Let $U_j^k = U(x_j, t_k)$ then the finite difference approximations for equation (2.4) are given by:

$$\begin{aligned}
\frac{\partial U(x_j,t_k)}{\partial t} &= \frac{U_j^{k+1} - U_j^k}{\Delta t} + \mathrm{O}(\Delta t), \text{ Forward Euler method for time derivative,} \\
\frac{\partial^2 U(x_j,t_k)}{\partial x^2} &= \frac{U_{j+1}^k - 2U_j^k + U_{j-1}^k}{\Delta x^2} + \mathrm{O}(\Delta x^2), \\
&\quad \text{Central difference method for spatial derivative.}
\end{aligned}$$

Our discretised PDE (equation (2.4)) becomes:

$$\begin{aligned}
\frac{U_j^{k+1} - U_j^k}{\Delta t} &= \frac{\beta}{\Delta x^2}\left(U_{j+1}^k - 2U_j^k + U_{j-1}^k\right), \\
\text{or } U_j^{k+1} &= s\left(U_{j+1}^k + U_{j-1}^k\right) + (1-2s)U_j^k, \\
\text{where } s &= \frac{\beta \Delta t}{\Delta x^2}.
\end{aligned}$$

U_j^{k+1} is the solution for the temperature at the next time step.

Suppose we have initial conditions $U(x,0) = U_j^0 = f(x_j)$, and mixed boundary conditions:

$$\begin{aligned} \text{Dirichlet boundary conditions at } x=0\text{: } U(0,t) &= U_0^k = g_1(t_k), \\ \text{Neumann boundary conditions at } x=a\text{: } U_x(a,t) &= \frac{\partial U_{n+1}^k}{\partial x} = g_2(t_k), \end{aligned}$$

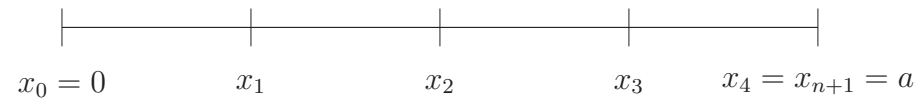
Numerical implementation of Explicit Forward Euler method

Solving equation (2.4): $U_t = \beta U_{xx}$ with:
initial conditions: $U_j^0 = f(x_j) = U(x, t=0)$,
Dirichlet boundary conditions at $x = 0$: $U(x=0,t) = U_0^k = g_1(t_k)$ and Neumann boundary conditions at $x = a$: $\partial U(a,t)/\partial x = \partial U_{n+1}^k/\partial x = g_2(t_k)$.

To solve using the Neumann boundary condition we need an extra step:

$$\begin{aligned} \frac{\partial U_{n+1}^k}{\partial x} &\approx \frac{U_{n+1}^k - U_n^k}{\Delta x} = g_2(t_k), \\ \text{or } U_{n+1}^k &= \Delta x g_2(t_k) + U_n^k. \end{aligned} \tag{2.5}$$

We can write out the matrix system of equations we will solve numerically for the temperature U. Suppose we use 5 grid points $x_0, x_1, x_2, x_3, x_4 = x_{n+1}$, ie. $n = 3$ in this example:



We let:

$$\vec{U}^k = \begin{pmatrix} U_1^k \\ U_2^k \\ U_3^k \end{pmatrix}, \text{ solution for temperature vector } \vec{U}^k \text{ at time } t_k.$$

The boundary conditions give $U_0^k = U(x=0,t_k)$ and $U_{n+1}^k = U_4^k = U(x = a, t_k)$.

We can rewrite $U_j^{k+1} = s\left(U_{j+1}^k + U_{j-1}^k\right) + (1-2s)U_j^k$ in matrix form:

$$\vec{U}^{k+1} = \begin{pmatrix} U_1^{k+1} \\ U_2^{k+1} \\ U_3^{k+1} \end{pmatrix} = \begin{pmatrix} 1-2s & s & 0 \\ s & 1-2s & s \\ 0 & s & 1-2s \end{pmatrix} \begin{pmatrix} U_1^k \\ U_2^k \\ U_3^k \end{pmatrix} + \begin{pmatrix} sU_0^k \\ 0 \\ sU_4^k \end{pmatrix} \tag{2.6}$$

Using boundary conditions: $U_0^k = g_1(t_k)$ and $U_4^k = \Delta x g_2(t_k) + U_3^k$ equation (2.6) becomes:

$$\vec{U}^{k+1} = \begin{pmatrix} U_1^{k+1} \\ U_2^{k+1} \\ U_3^{k+1} \end{pmatrix} = \underbrace{\begin{pmatrix} 1-2s & s & 0 \\ s & 1-2s & s \\ 0 & s & \underbrace{1-s}_{\text{Neumann bc}} \end{pmatrix}}_{A} \begin{pmatrix} U_1^k \\ U_2^k \\ U_3^k \end{pmatrix} + \underbrace{\begin{pmatrix} \overbrace{sg_1(t_k)}^{\text{Dirichlet bc}} \\ 0 \\ \underbrace{s\Delta x g_2(t_k)}_{\text{Neumann bc}} \end{pmatrix}}_{\vec{b}}$$

$$\text{or } \vec{U}^{k+1} = A\vec{U}^k + \vec{b} \tag{2.7}$$

The term $(1-s)$ in the matrix A above and the term $(s\Delta x g_2(t_k))$ in vector $\vec{b}$ above are from the Neumann boundary condition given using the approximation in equation (2.5).

Matlab code for Explicit Forward Euler method

The matlab code for this example is **ForwardEuler.m**.

Solving equation (2.4): $U_t = \beta U_{xx}$ with $0 \le x \le 1$, $\beta = 10^{-5}$, and $0 \le t \le 12,000$, using $m = 600$ time steps, and $n = 39$ for 41 grid points in x.

Initial conditions: $U(x, t = 0) = 2x + \sin(2\pi x) + 1$,
and Dirichlet boundary conditions at $x = 0$: $U(x = 0, t) = 1$ and Neumann boundary conditions at $x = 1$: $\partial U(x = 1, t)/\partial x = 2$.

With these boundary conditions we can check that the numerical solution approximates the steady state solution $U(x, t) = 2x + 1$ as $t \to \infty$. Your solution using the finite difference code can also be checked using Matlab's PDE solver (using `pdex1`).

Figure 2.1 shows the initial conditions in (a) and matlab solution for temperature distribution along rod with time in (b). The numerical solution matches the analytical solution reasonably well: $U(x, t) = 2x + 1$ at the final time. However the Neumann boundary condition at $x = 1$ introduces error

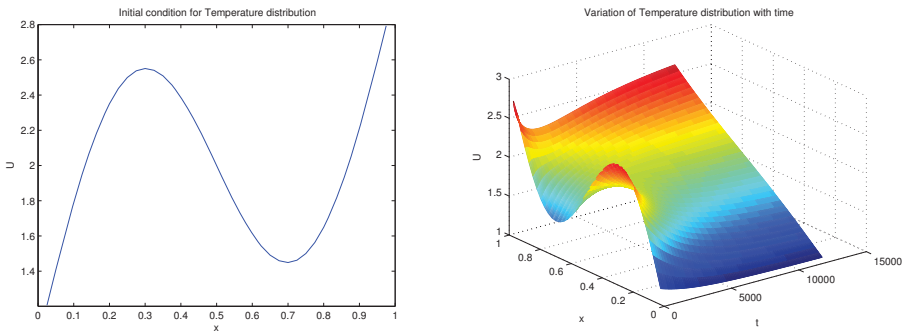


Figure 2.1: Initial conditions in (a) and matlab solution using Forward Euler method for temperature distribution along rod with time in (b)

through the approximation to the spatial derivative in this boundary condition so the numerical solution at $x = 1$ is not exactly $U = 3$ but close to it.

2.2.4 Stability criteria for forward Euler method

Suppose $U(x,0) = f(x) = \xi \cos(\pi x/\Delta x)$ where these initial conditions data oscillate with the same frequency as the grid, $\Delta x = a/n + 1$, As before we let $x_j = j\Delta x, \quad 0 \leq j \leq n+1$

$$f(x_j) = \xi \cos(\pi j) = \xi(-1)^j$$

Using finite difference discretisation of (2.4) $U_t = \beta U_{xx}$:

$$U_j^{k+1} = s(U_{j+1}^k + U_{j-1}^k) + (1-2s)U_j^k$$

At first time step ($k = 1$):

$$\begin{aligned} U_j^1 &= (1-2s)\xi(-1)^j + s\xi \underbrace{[(-1)^{j+1} + (-1)^{j-1}]}_{-2(-1)^j} \\ &= (1-4s)\xi(-1)^j \end{aligned}$$

At $k = 2$:

$$U_j^2 = (1-2s)U_j^1 + s(U_{j+1}^1 + U_{j-1}^1)$$

$$
\begin{aligned}
U_j^2 &= (1-2s)(1-4s)\xi(-1)^j + s\underbrace{[(1-4s)\xi(-1)^{j+1} + (1-4s)\xi(-1)^{j-1}]}_{-2(1-4s)\xi(-1)^j} \\
&= (1-4s)^2\xi(-1)^j
\end{aligned}
$$

Therefore at $k = n$

$$U_j^n = (1-4s)^n\xi(-1)^j$$

NB: the term $(1-4s)$ determines stability.
This solution for U_j^n will become unbounded as $n \to \infty$ if $|1-4s| \geq 1$ or $s > 1/2$.
We know the exact solution $|U(x,t)| \leq |U(x,t_0)| = \xi \quad \forall \quad x,t.$
The Forward Euler method is only stable if s (known as the gain parameter) satisfies $0 \leq s \leq 1/2$ or equivalently the time step satisfies: $\Delta t \leq \Delta x^2/2\beta$. You can check that using the matlab code ForwardEuler.m that when the time step exceeds this value the numerical solution becomes unstable.

2.3 Method of lines

There are other explicit numerical methods that can be applied to the 1-D heat or diffusion equation such as the Method of Lines which is used by Matlab and Mathematica. The trick with the Method of Lines is that it replaces all spatial derivatives with finite differences but leaves the time derivatives. It is then possible to use a stiff ordinary differential equation solver on the time derivatives in the resulting system.

2.3.1 Example

The matlab code for this example is **MethodOfLines.m** and **Uprime.m**

We are solving the same system again with the method of lines: $U_t = \beta U_{xx}$ where the initial conditions are $U(x,0) = \sin(2\pi x) + 2x + 1$
$0 \leq x \leq 1, \quad \beta = 10^{-5}, \quad 0 \leq t \leq 12,000.$
boundary conditions are $U(0,t) = 1 \quad$ and $\quad U_x(1,t) = 2$
Again we get:

$$\frac{\partial \vec{U}}{\partial t} = A\vec{U} + \vec{b}$$

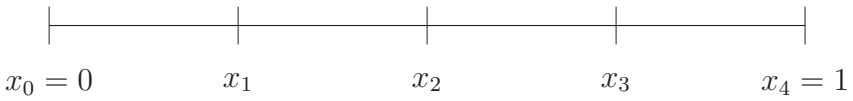
How? Replace

$$U_{xx} = \frac{U_{j+1} - 2U_j + U_{j-1}}{\Delta x^2},$$

where $U(x_j,t) = U_j(t), \quad x_j = j\Delta x, \quad 0 \le j \le n+1$
$\Delta x = a/(n+1) = 1/(n+1) \qquad (a=1)$
with boundary conditions: $\qquad U(0,t) = U_0(t) = 1$

$$\frac{\partial U}{\partial x}(1,t) = \frac{\partial U_{n+1}}{\partial x}(t) \simeq \frac{U_{n+1} - U_n}{\Delta x} = 2 \Rightarrow U_{n+1} = U_n + 2\Delta x$$

In matrix form for $n = 3$ elements:



$$\frac{\partial \vec{U}}{\partial t} = \begin{pmatrix} \dot{U}_1 \\ \dot{U}_2 \\ \dot{U}_3 \end{pmatrix} = \frac{\beta}{\Delta x^2} \begin{pmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ U_3 \end{pmatrix} + \frac{\beta}{\Delta x^2} \begin{pmatrix} 1 \\ 0 \\ 2\Delta x \end{pmatrix}$$

$$\dot{\vec{U}} = A\vec{U} + \vec{b}$$

We solve for $\vec{U}$ using **ode45** and matlab code **MethodOfLines.m** and **Uprime.m**.