The University of Manchester

COMP36212
Week 8 – Implicit vs Explicit

Dr Oliver Rhodes
oliver.rhodes@manchester.ac.uk

Overview

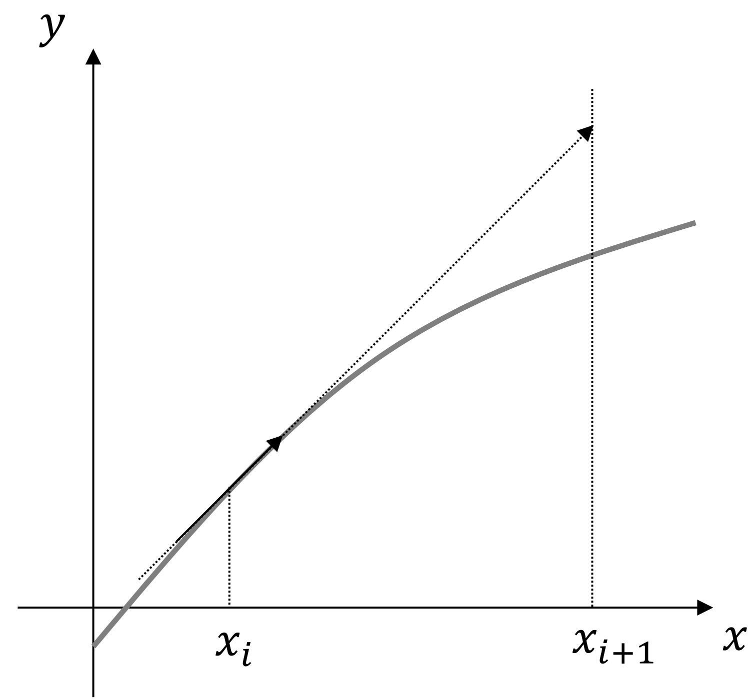
- Recap asynchronous content:
  - Implicit methods for stiff ODEs
  - Implicit solution of PDEs
- Solving tridiagonal systems
  - Thomas algorithm
- Example: solving the hyperbolic wave equation

Explicit methods for ODEs

- The methods explored previously are all classified as *explicit* techniques.
  - Euler's method
  - Midpoint method
  - Heun's method

- They all use information at the beginning of an interval to predict the value at the end of the interval
  - Efficient, and easy to implement
  - Well-suited to initial value problems
  - Accuracy has been controlled by step size

- Do all ODEs respond the same to numerical solution?



Example: Explicit Euler

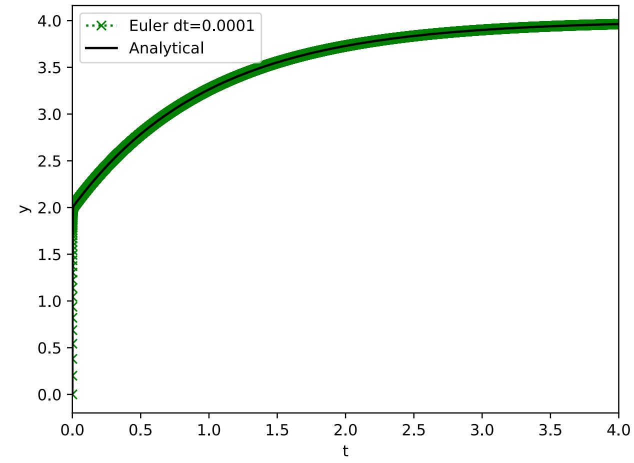
- Consider the ODE:

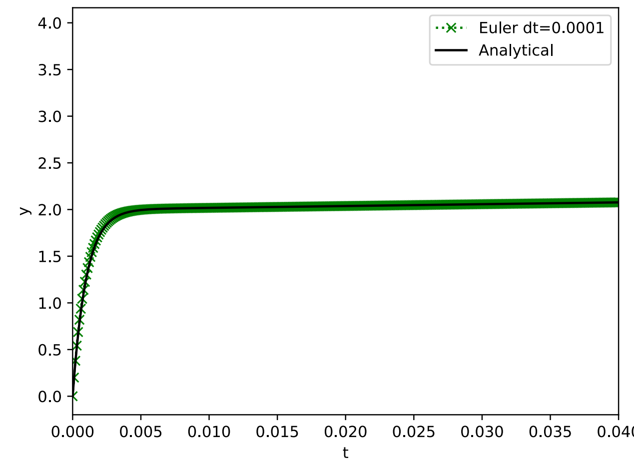
$$\frac{dy}{dt} = -1000y + 4000 - 2000\mathrm{e}^{-t}$$

- Using Euler's method the solution across interval Δt is given by:

$$y_{i+1} = y_i + (-1000y_i + 4000 - 2000\mathrm{e}^{-t_i})\Delta t$$

- Solving with $\Delta t = 0.0001,$ over the interval $0 < t \leq 4.0$, with initial condition: $y(t_0 = 0.0) = 0.0$





Example: Explicit Euler

- Consider the ODE:

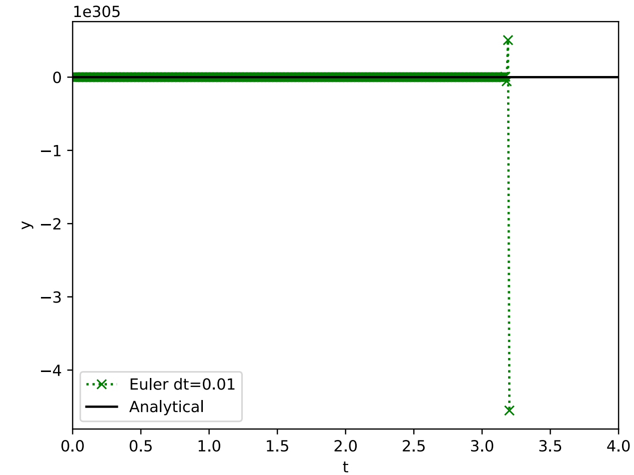
$$\frac{dy}{dt} = -1000y + 4000 - 2000e^{-t}$$

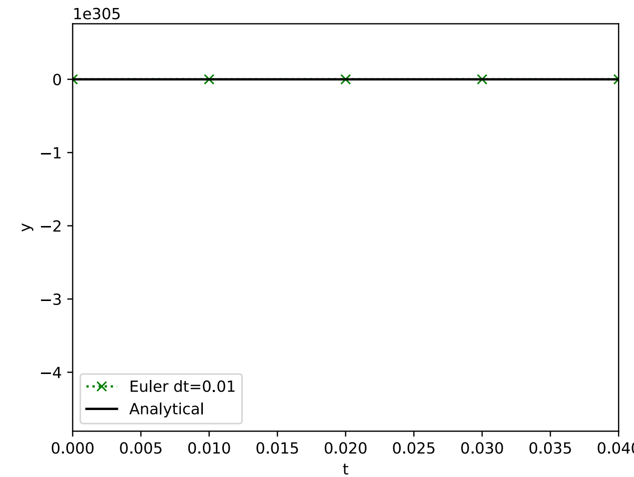
- Using Euler's method the solution across interval Δt is given by:

$$y_{i+1} = y_i + (-1000y_i + 4000 - 2000e^{-t_i})\Delta t$$

- Solving with $\color{red}{\Delta t = 0.01}$, over the interval $0 < t \leq 4.0$, with initial condition: $y(t_0 = 0.0) = 0.0$

- Increasing the step size hasn't just made the solution *inaccurate*, it has made it *unstable*





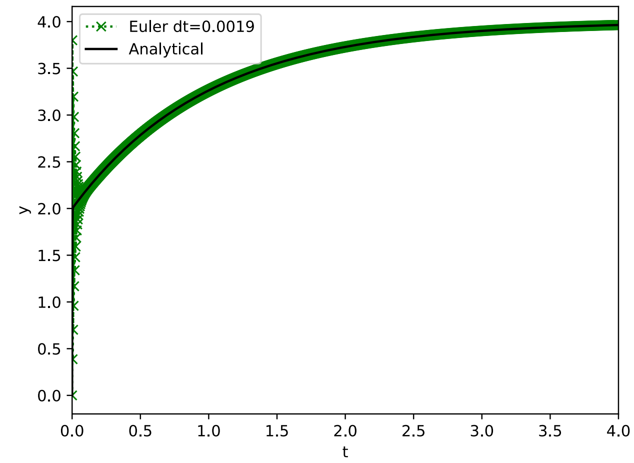
Stability

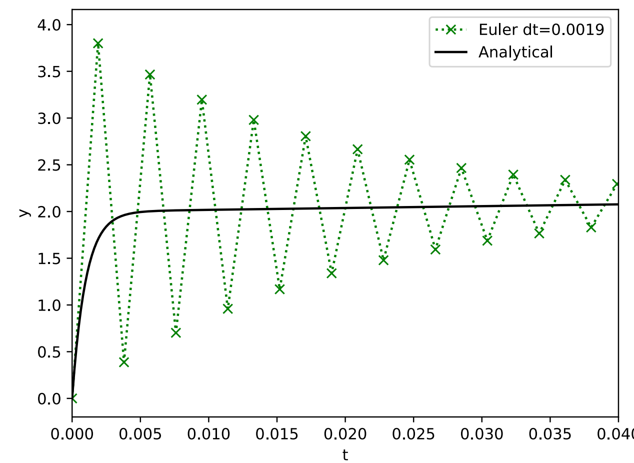
- The condition $h < 0.002$, enables a bound to be put on the step size to ensure stability

- In this case the solution is stable, although oscillates about the true solution

- Explicit methods require strict conditions for the step size to be bounded, but often require even smaller step sizes for accuracy
  - An accurate solution was found with $\Delta t = 0.0001$ as seen at the beginning of the example

- If we are not interested in the accuracy of the initial transient, a step size of $\Delta t = 0.001$ could be used





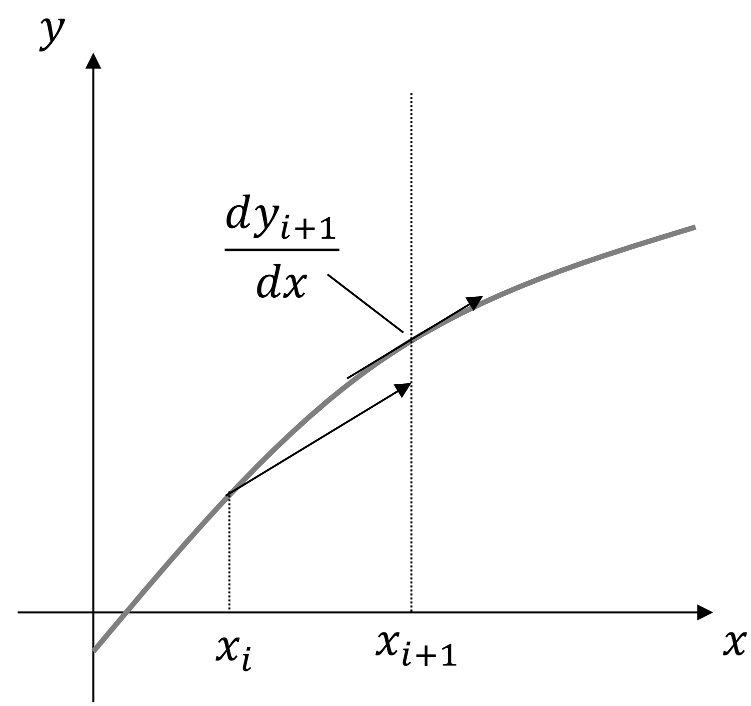
Implicit Methods

- Implicit methods offer a solution to the stability and accuracy problems.

- Compute updates based on information at end of solution step:

$$y_{i+1} = y_i + \frac{dy_{i+1}}{dx}\Delta x$$

- This is known as the *implicit Euler* method. Implicit, as the unknown appears on both sides of the update equation



Example: Implicit Euler

- Solve over the interval $0 < t \leq 4.0$, with initial condition: $y(t_0 = 0.0) = 0.0$

$$\frac{dy}{dt} = -1000y + 3000 - 2000e^{-t}$$

- An update equation based on the implicit Euler's method can be formulated as:

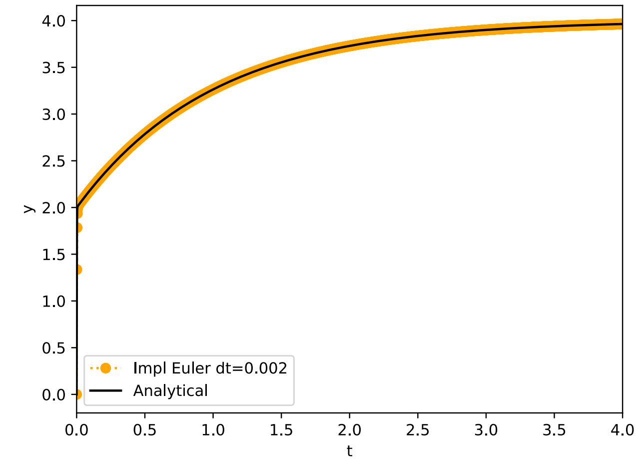
$$y_{i+1} = y_i + \frac{dy_{i+1}}{dt}\Delta t$$

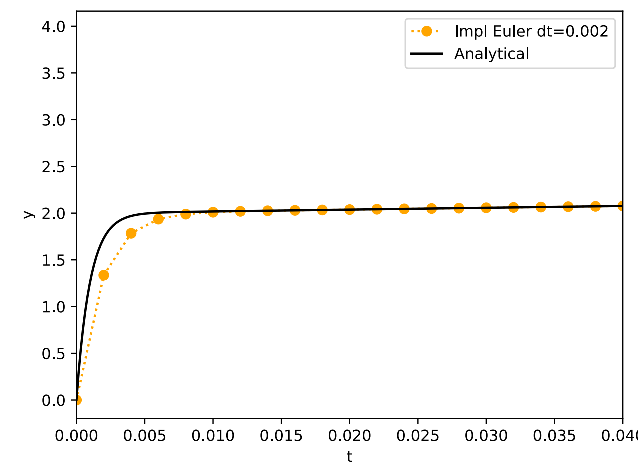
$$y_{i+1} = y_i + (-1000y_{i+1} + 3000 - 2000e^{-t_{i+1}})\Delta t$$

- As the ODE is linear in y, the update equation can be rearranged to give:

$$y_{i+1} = \frac{y_i + 3000\Delta t - 2000\Delta t e^{-t_{i+1}}}{1 + 1000\Delta t}$$

- Solving for $\Delta t = 0.002$





Example: Implicit Euler

- Solve over the interval $0 < t \leq 4.0$, with initial condition: $y(t_0 = 0.0) = 0.0$

$$\frac{dy}{dt} = -1000y + 3000 - 2000e^{-t}$$

- An update equation based on the implicit Euler's method can be formulated as:

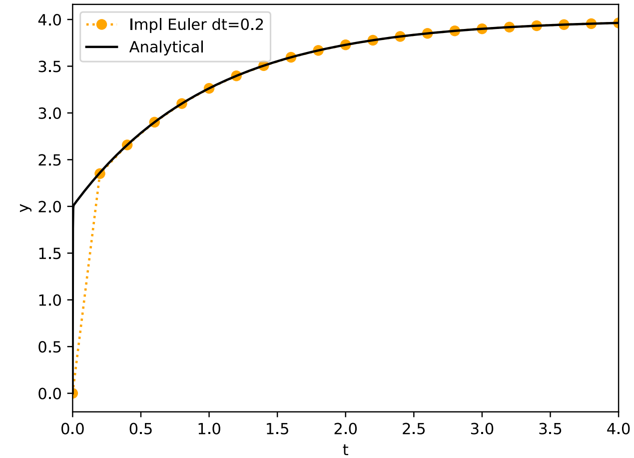
$$y_{i+1} = y_i + \frac{dy_{i+1}}{dt}\Delta t$$

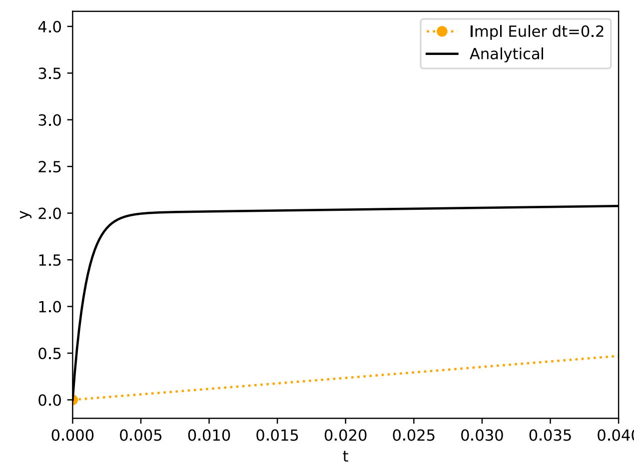
$$y_{i+1} = y_i + (-1000y_{i+1} + 3000 - 2000e^{-t_{i+1}})\Delta t$$

- As the ODE is linear in y, the update equation can be rearranged to give:

$$y_{i+1} = \frac{y_i + 3000\Delta t - 2000\Delta t e^{-t_{i+1}}}{1 + 1000\Delta t}$$

- Solving for $\Delta t = 0.2$. With a large step size the fast transient cannot be resolved, but the solution is stable!





Summary: Implicit methods

- Use implicit information, information that is unknown at the current solution step

- Can be unconditionally stable for certain problems – very useful with stiff ODEs, containing solutions with fast and slow components

- Allow greater accuracy, with larger timesteps

- Have simple update equations when the ODE is linear in the dependent variable (as seen in example)

- Are more complicated when ODE is nonlinear – require alternative soltion approaches:

$$\frac{dy}{dx} = ky^2$$

$$y_{i+1} = y_i + ky_{i+1}^2 \Delta x, \qquad y_{i+1} = \frac{y_i}{1 - ky_{i+1}\Delta x}$$

Partial Differential Equations

- Differential equations involving two or more independent variables are known as partial differential equations (PDEs)

$$\frac{\partial^2(x,t)}{\partial x^2} = \frac{1}{\kappa^2}\frac{\partial^2\phi(x,t)}{\partial t^2} \qquad \frac{\partial^2\phi(x,y)}{\partial x^2} + \frac{\partial^2\phi(x,y)}{\partial y^2} = 0$$

- The order of a PDE is defined by its highest order derivative; it is described as linear if the unknown functions and derivatives are linear (with coefficients depending only on the independent variables)

- Analytical solution is difficult - typically hard to write down a formula describing PDE solution, especially for meaningful engineering problems (i.e. anything beyond a trivial example).

- Here we'll focus on linear second order PDEs, solving for real-valued functions over a rectangular grid. This grid can be bounded in space and time, enabling description of wide range of problems from science and engineering

Partial Differential Equations

- A general form for a PDE involving a function of two real variables:

$$a(x,y)\frac{\partial^2 f}{\partial x^2}+b(x,y)\frac{\partial^2 f}{\partial x\partial y}+c(x,y)\frac{\partial^2 f}{\partial y^2}+d(x,y)\frac{\partial f}{\partial x}+e(x,y)\frac{\partial f}{\partial y}+g(x,y)=0$$

- Different systems are produced depending on the values of the constants $a, b, c, d,$ and e.

- The second order system can be can be characterised by its auxiliary quadratic equation, the roots of which enable a useful classification of the underlying system.

$$b^2-4ac$$

MANCHESTER 1824
The University of Manchester

Types of PDE

Form	$b^2 - 4ac$		Examples
Elliptic	$b^2 - 4ac < 0$ $(a = 1, b = 0, c = 1)$	Arise from steady state problems, such as potential theory or flow theory	Poisson's equation $\frac{\partial^2 \phi(x,y)}{\partial x^2} + \frac{\partial^2 \phi(x,y)}{\partial y^2} = g(x,y)$ Laplace's equation $\frac{\partial^2 \phi(x,y)}{\partial x^2} + \frac{\partial^2 \phi(x,y)}{\partial y^2} = 0$
Hyperbolic	$b^2 - 4ac > 0$ $(a = 1, b = 0, c = -\frac{1}{\kappa^2})$	Arise from vibrational and radiative problems, as described by wave mechanics	The wave equation: $\frac{\partial^2 (x,t)}{\partial x^2} = \frac{1}{\kappa^2} \frac{\partial^2 \phi(x,t)}{\partial t^2}$
Parabolic	$b^2 - 4ac = 0$ $(a = 1, b = 0, c = 0)$	Arise from transient flow problems, e.g. describing conduction	Heat conduction equation: $\frac{\partial^2 \phi(x,t)}{\partial x^2} = \frac{1}{\kappa} \frac{\partial \phi(x,t)}{\partial t}$

Solution of Elliptic PDEs

Solution steps when using finite difference method:

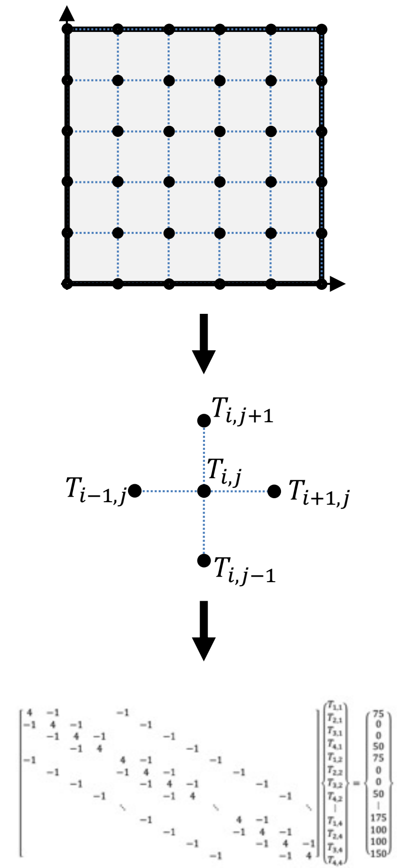
1. Create graphical representation of solution domain and draw computational molecule

2. Substitute finite difference approximations of derivatives into PDE

3. Rearrange to give known terms on right-hand side, and unknown values to be solved on left-hand side

4. Write characteristic equation for every point (or node) in the solution domain, incorporating boundary conditions where appropriate

5. Formulate the problem in matrix form $Ax = b$, and solve for x

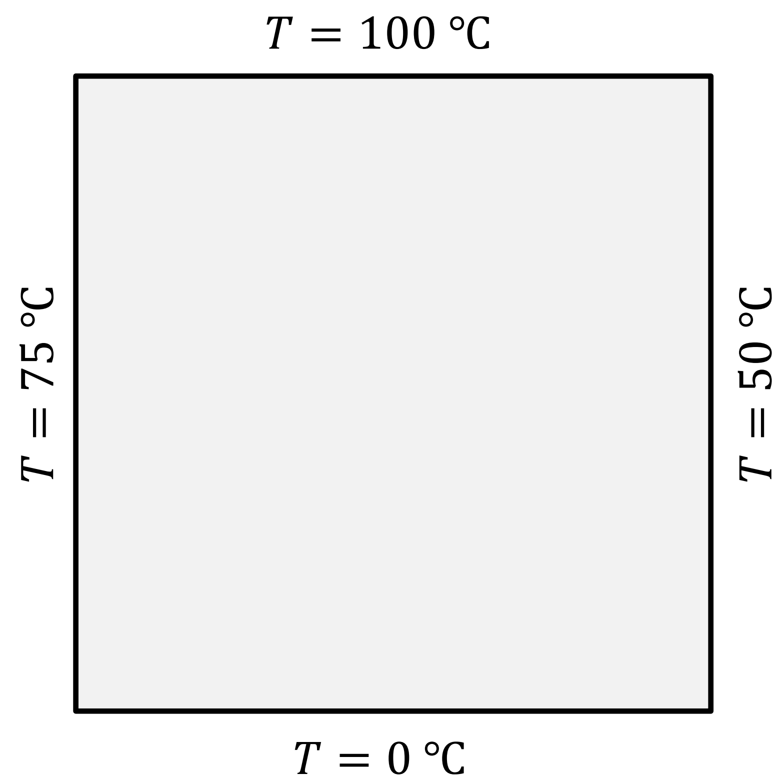


Elliptic Example

- Problem: calculate the steady state temperature distribution over a square plate, with sides held at temperatures: $100\ ^\circ\mathrm{C}, 50\ ^\circ\mathrm{C}, 0\ ^\circ\mathrm{C}\ \&\ 75\ ^\circ\mathrm{C}$

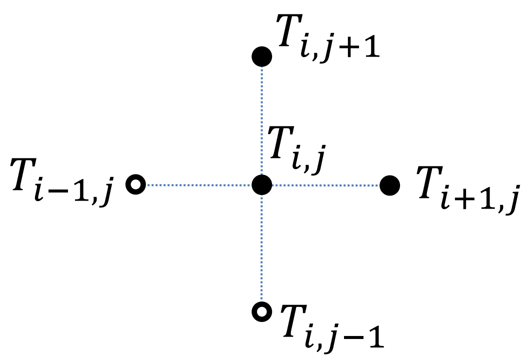
- The plate had edge length $10\ \mathrm{cm}$, with temperature distribution satisfying the elliptic Laplace equation:
$$\frac{\partial^2 T}{dx^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

- Use the finite difference approach with step sizes of $\Delta x = \Delta y = 2\ \mathrm{cm}$, to evaluate the temperature distribution $T(x, y)$

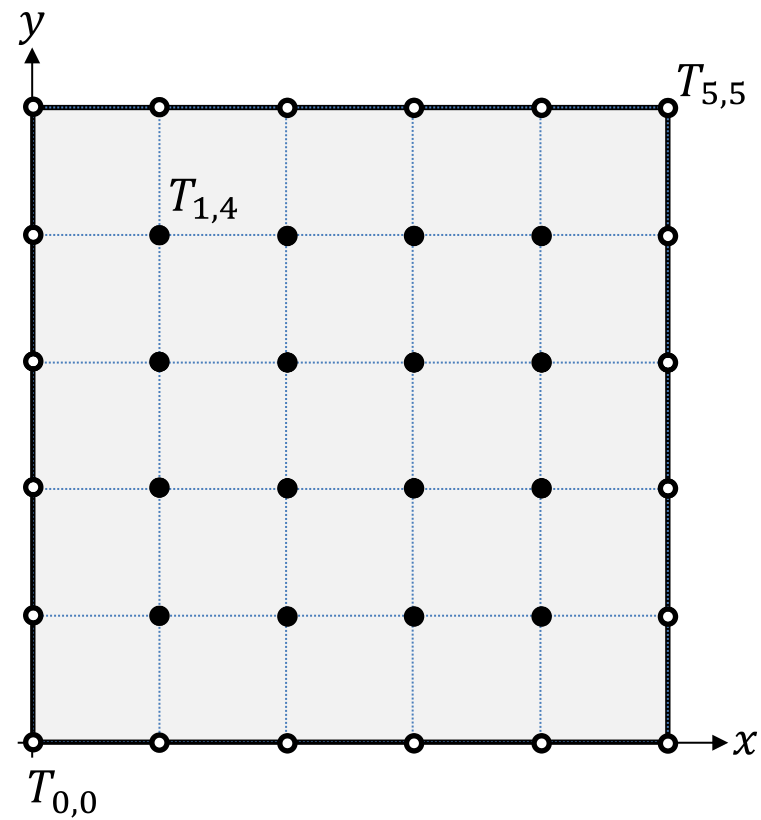


Elliptic Example

- Graphical representation of solution domain and finite difference grid
- Construct computational molecule:



- Solid nodes indicate solution variable, hollow nodes indicate node with boundary condition

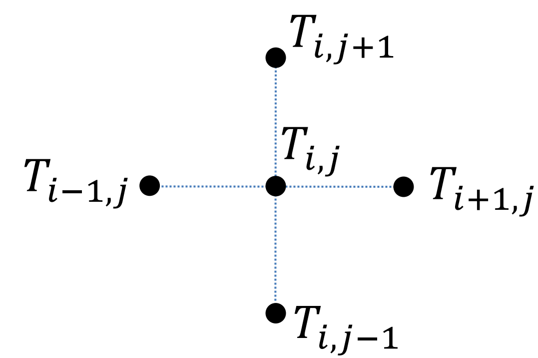


Elliptic Example

- Step 1: substitute finite difference approximations for second order partial derivatives:

$$\frac{\partial^2 T}{dx^2} = \frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{\Delta x^2}, \qquad \frac{\partial^2 T}{dy^2} = \frac{T_{i,j-1} - 2T_{i,j} + T_{i,j+1}}{\Delta y^2}$$

$$\frac{\partial^2 T}{dx^2} + \frac{\partial^2 T}{dy^2} = \frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{\Delta x^2} + \frac{T_{i,j-1} - 2T_{i,j} + T_{i,j+1}}{\Delta y^2} = 0$$



- Which provides the computational molecule description:

$$-T_{i-1,j} - T_{i+1,j} - T_{i,j-1} - T_{i,j+1} + 4T_{i,j} = 0$$

- This can now be written for all interior nodes

Elliptic Example

- For node $T_{1,1}$ (note: $T_{1,0} = 0\ ℃, T_{0,1} = 75$)

$$-T_{0,1} - T_{2,1} - T_{1,0} - T_{1,2} + 4T_{1,1} = 0$$
$$-T_{2,1} - T_{1,2} + 4T_{1,1} = 75$$

- For node $T_{2,1}$ (note: $T_{2,0} = 0\ ℃$)

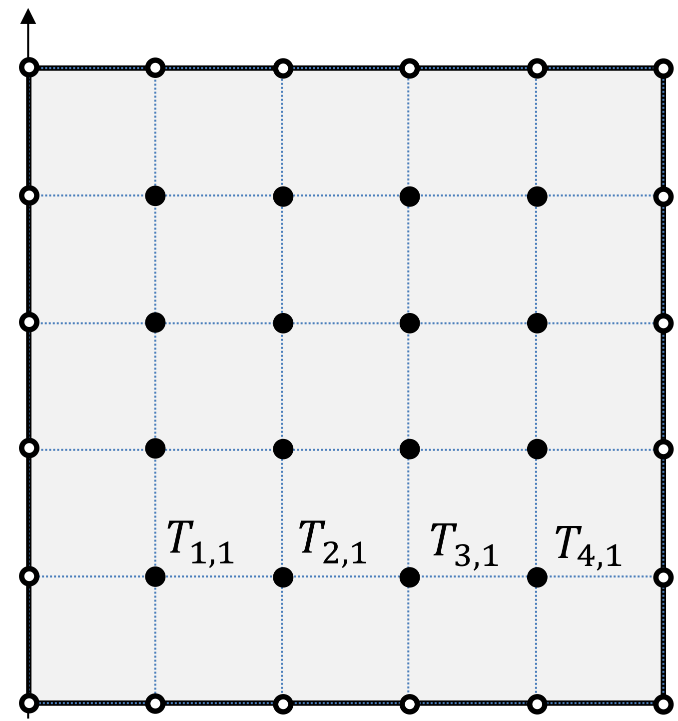
$$-T_{1,1} - T_{3,1} - T_{2,0} - T_{2,2} + 4T_{2,1} = 0$$
$$-T_{1,1} - T_{3,1} - T_{2,2} + 4T_{2,1} = 0$$

- For node $T_{3,1}$ (note: $T_{3,0} = 0\ ℃$)

$$-T_{2,1} - T_{4,1} - T_{3,0} - T_{3,2} + 4T_{3,1} = 0$$
$$-T_{2,1} - T_{4,1} - T_{3,2} + 4T_{3,1} = 0$$

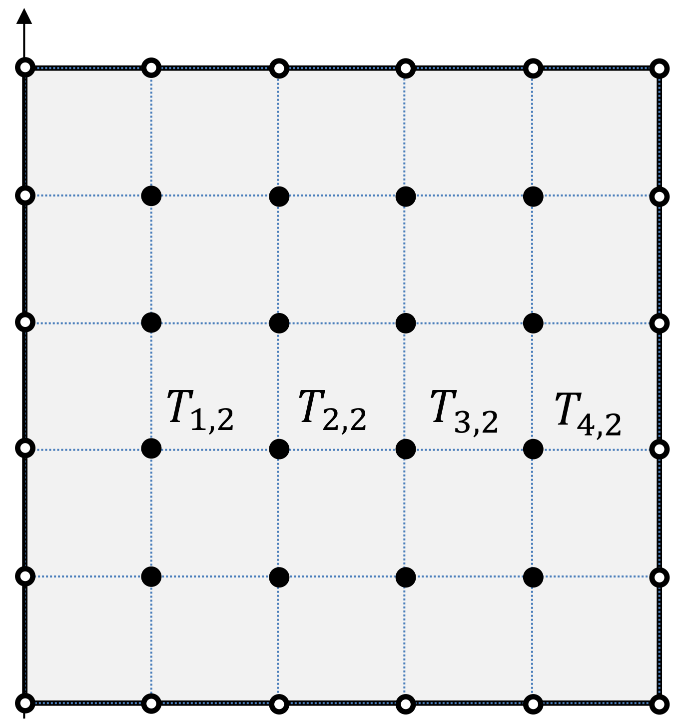
- For node $T_{4,1}$ (note: $T_{4,0} = 0\ ℃$, $T_{5,1} = 50\ ℃$)

$$-T_{3,1} - T_{5,1} - T_{i4\ 0} - T_{4,2} + 4T_{4,1} = 0$$
$$-T_{3,1} - T_{4,2} + 4T_{4,1} = 50$$



Elliptic Example

- For node $T_{1,2}$ (note: $T_{0,2} = 75\ ℃$)
  $$-T_{0,2} - T_{2,2} - T_{1,1} - T_{1,3} + 4T_{1,2} = 0$$
  $$-T_{2,2} - T_{1,1} - T_{1,3} + 4T_{1,2} = 75$$
- For node $T_{2,2}$
  $$-T_{1,2} - T_{3,2} - T_{2,1} - T_{2,3} + 4T_{2,2} = 0$$
- For node $T_{3,2}$
  $$-T_{2,2} - T_{4,2} - T_{3,1} - T_{3,3} + 4T_{3,2} = 0$$
- For node $T_{4,2}$ (note: $T_{5,2} = 50\ ℃$)
  $$-T_{3,2} - T_{5,2} - T_{4,1} - T_{4,3} + 4T_{4,2} = 0$$
  $$-T_{3,2} - T_{4,1} - T_{4,3} + 4T_{4,2} = 50$$



Elliptic Example

- For node $T_{1,3}$ (note: $T_{0,3} = 75\ ℃$)

$$-T_{0,3} - T_{2,3} - T_{1,2} - T_{1,4} + 4T_{1,3} = 0$$
$$-T_{2,3} - T_{1,2} - T_{1,4} + 4T_{1,3} = 75$$

- For node $T_{2,3}$

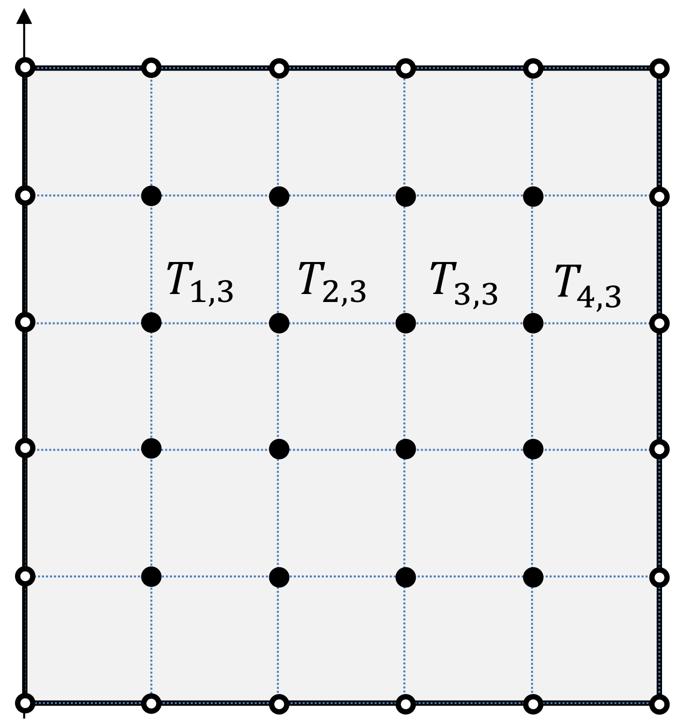
$$-T_{1,3} - T_{3,3} - T_{2,2} - T_{2,4} + 4T_{2,3} = 0$$

- For node $T_{3,3}$

$$-T_{2,3} - T_{4,3} - T_{3,2} - T_{3,4} + 4T_{3,3} = 0$$

- For node $T_{4,3}$ (note: $T_{5,3} = 50\ ℃$)

$$-T_{3,3} - T_{5,3} - T_{4,2} - T_{4,4} + 4T_{4,3} = 0$$
$$-T_{3,3} - T_{4,2} - T_{4,4} + 4T_{4,3} = 50$$



Elliptic Example

- For node $T_{1,4}$ (note: $T_{1,5} = 100\ ℃$, $T_{0,4} = 75\ ℃$)

$$-T_{0,4} - T_{2,4} - T_{1,3} - T_{1,5} + 4T_{1,4} = 0$$
$$-T_{2,4} - T_{1,3} + 4T_{1,4} = 175$$

- For node $T_{2,4}$ (note: $T_{2,5} = 100\ ℃$)

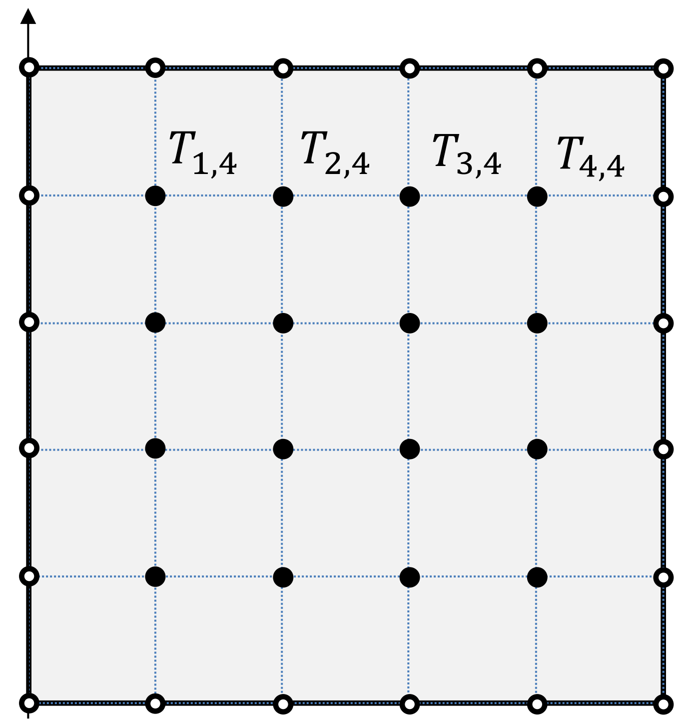
$$-T_{1,4} - T_{3,4} - T_{2,3} - T_{2,5} + 4T_{2,4} = 0$$
$$-T_{1,4} - T_{3,4} - T_{2,3} + 4T_{2,4} = 100$$

- For node $T_{3,4}$ (note: $T_{3,5} = 100\ ℃$)

$$-T_{2,4} - T_{4,4} - T_{3,3} - T_{3,5} + 4T_{3,4} = 0$$
$$-T_{2,4} - T_{4,4} - T_{3,3} + 4T_{3,4} = 100$$

- For node $T_{4,4}$ (note: $T_{5,4} = 50\ ℃$, $T_{4,5} = 100\ ℃$)

$$-T_{3,4} - T_{4,3} + 4T_{4,4} = 150$$

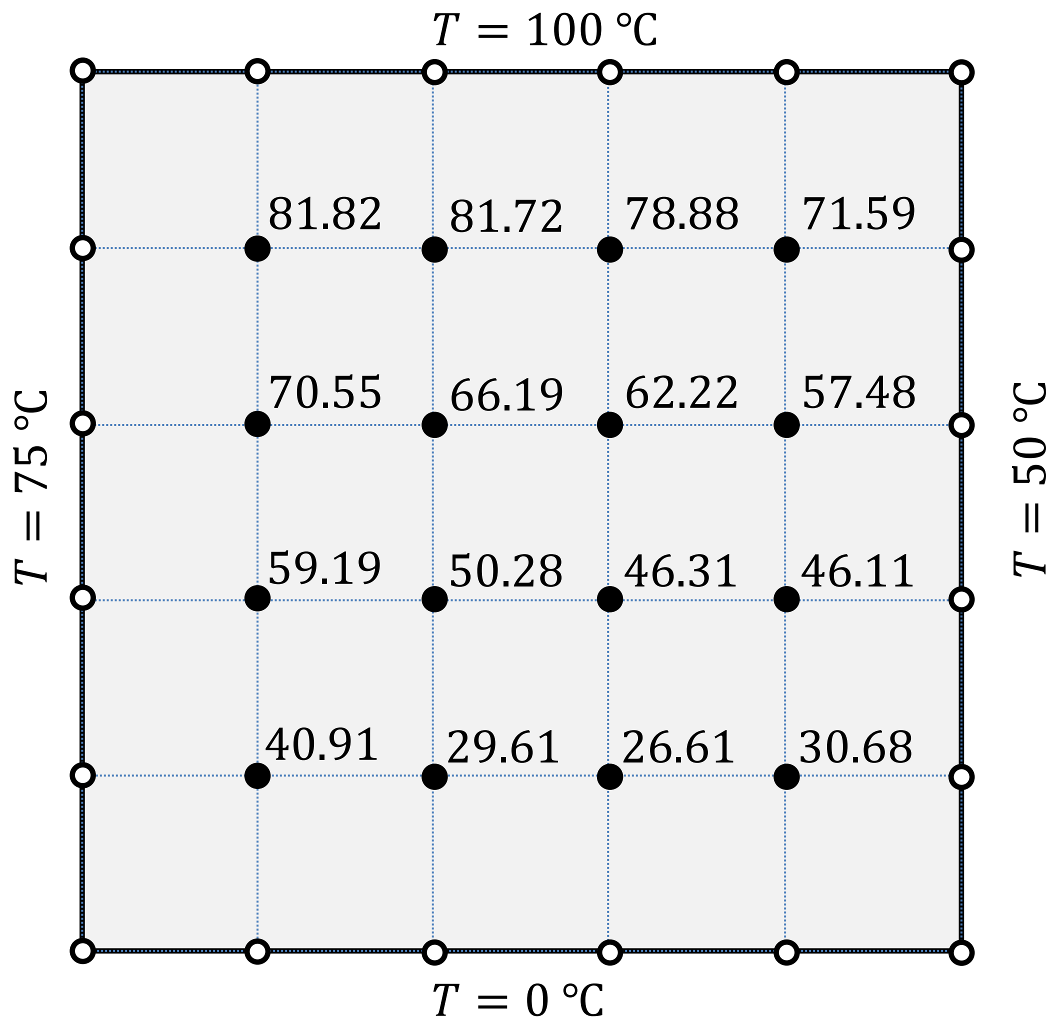


Elliptic Example

- Write in matrix form: $Ax = b$

$$
\begin{bmatrix}
4 & -1 & & & -1 & & & & & & & & \\
-1 & 4 & -1 & & & -1 & & & & & & & \\
& -1 & 4 & -1 & & & -1 & & & & & & \\
& & -1 & 4 & & & & -1 & & & & & \\
-1 & & & & 4 & -1 & & & -1 & & & & \\
& -1 & & & -1 & 4 & -1 & & & -1 & & & \\
& & -1 & & & -1 & 4 & -1 & & & -1 & & \\
& & & -1 & & & -1 & 4 & & & & -1 & \\
& & & & \ddots & & & & \ddots & & & & \ddots \\
& & & & & -1 & & & & 4 & -1 & & \\
& & & & & & -1 & & & -1 & 4 & -1 & \\
& & & & & & & -1 & & & -1 & 4 & -1 \\
& & & & & & & & -1 & & & -1 & 4
\end{bmatrix}
\begin{Bmatrix}
T_{1,1} \\ T_{2,1} \\ T_{3,1} \\ T_{4,1} \\ T_{1,2} \\ T_{2,2} \\ T_{3,2} \\ T_{4,2} \\ \vdots \\ T_{1,4} \\ T_{2,4} \\ T_{3,4} \\ T_{4,4}
\end{Bmatrix}
=
\begin{Bmatrix}
75 \\ 0 \\ 0 \\ 50 \\ 75 \\ 0 \\ 0 \\ 50 \\ \vdots \\ 175 \\ 100 \\ 100 \\ 150
\end{Bmatrix}
$$

- By ordering nodes and arrangement of equations the matrix becomes banded, and sparse.
- Can be solved here by inverting, however when the number of nodes increases this method becomes expensive
- Efficient methods exist for solving sparse matrix problems – explored in separate session



T = 100 °C
81.82
81.72
78.88
71.59
70.55
66.19
62.22
57.48
59.19
50.28
46.31
46.11
40.91
29.61
26.61
30.68
T = 75 °C
T = 50 °C
T = 0 °C

MANCHESTER 1824
The University of Manchester

Time based PDEs

- Parabolic PDEs: e.g. heat conductance equation

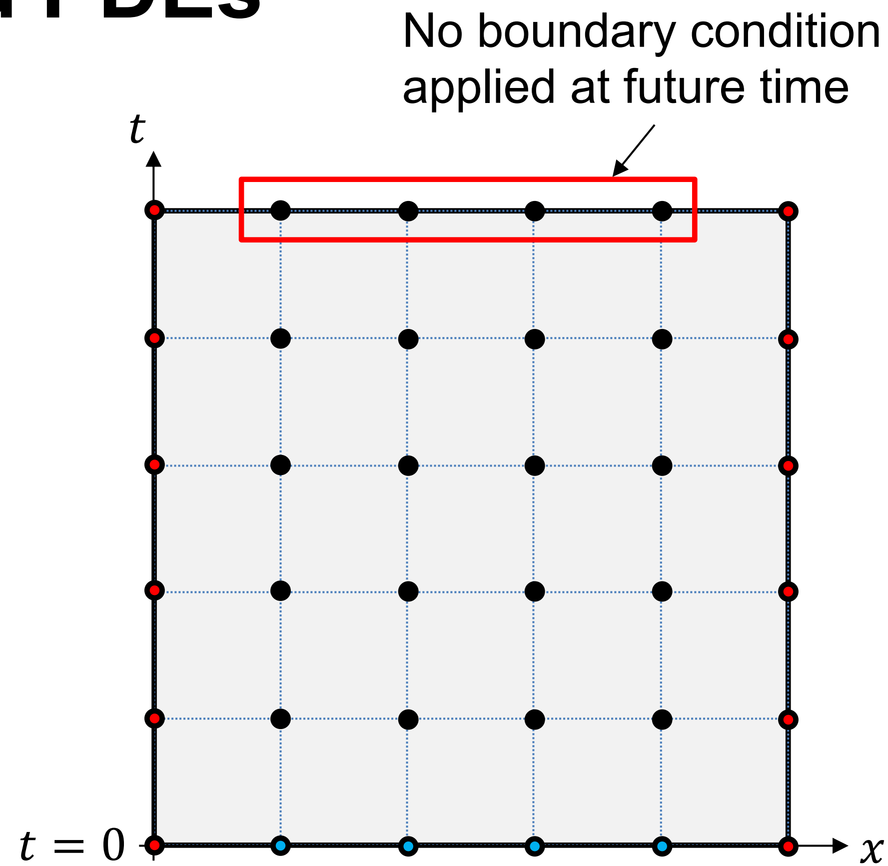
$$\frac{\partial^2 \phi(x,t)}{\partial x^2} = \frac{1}{\kappa} \frac{\partial \phi(x,t)}{\partial t}$$

- Hyperbolic PDEs: e.g. wave equation

$$\frac{\partial^2 (x,t)}{\partial x^2} = \frac{1}{\kappa^2} \frac{\partial^2 \phi(x,t)}{\partial t^2}$$

- Boundary conditions specified to constrain solution in spatial dimension

- Initial conditions specified at $t = 0$ to constrain temporal solution



Example: Parabolic PDE

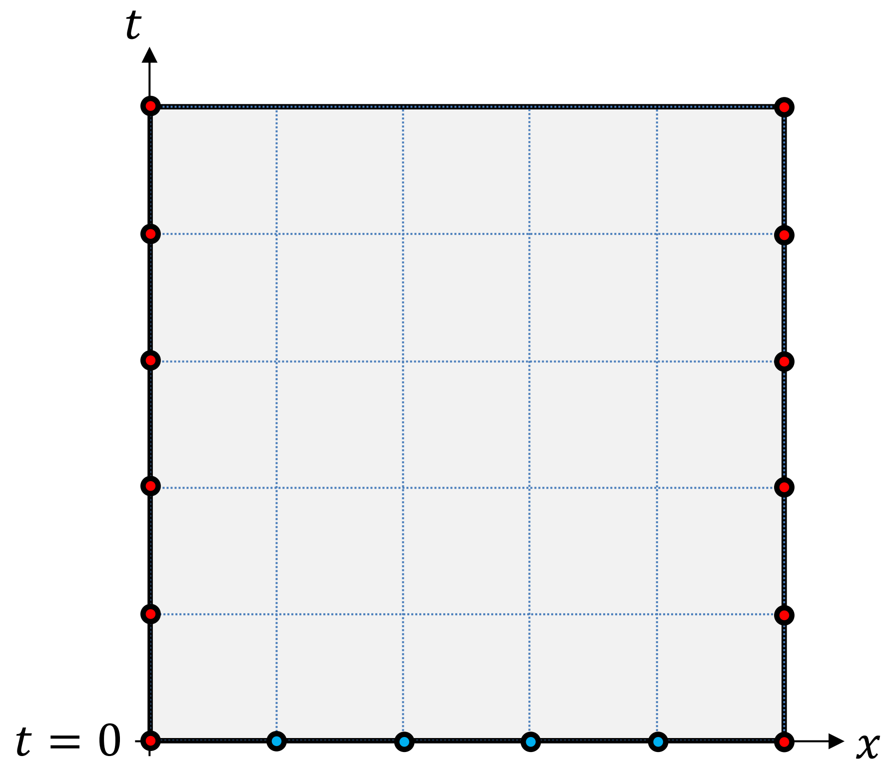
- Solve the numerically the PDE to compute $\phi(x,t)$, over the intervals $0 < x < 2$, and $0 < t < 10$; with step sizes: $\Delta x = 0.4$, $\Delta t = 1$

$$\frac{\partial^2 \phi(x,t)}{\partial x^2} = \frac{1}{\kappa}\frac{\partial \phi(x,t)}{\partial t}$$

- The following auxiliary conditions are known:
  - $\phi(x=0,t) = 10.0$
  - $\phi(x=2,t) = 10.0$
  - $\phi(x,t=0) = 25.0$

  Boundary conditions (first two); Initial condition (third)

- Step 1: Initialise computational grid, and apply boundary conditions to define solution at known points



Parabolic PDE: Explicit Method

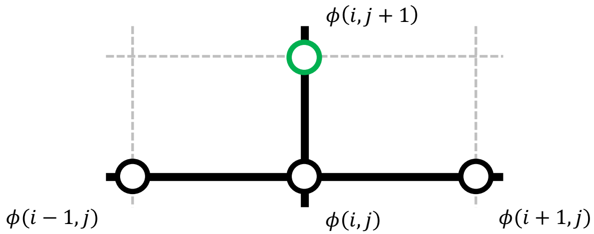
The University of Manchester

- Step 2: Substitute finite difference approximations for partial derivatives at node i, j:

$$\frac{\partial^2 \phi(x,t)}{\partial x^2} = \frac{\phi_{i-1,j} - 2\phi_{i,j} + \phi_{i+1,j}}{\Delta x^2}$$

$$\frac{\partial \phi(x,t)}{\partial t} = \frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta t}$$

$$\frac{\phi_{i-1,j} - 2\phi_{i,j} + \phi_{i+1,j}}{\Delta x^2} = \frac{1}{\kappa}\frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta t}$$



- Rearranging, gives an expression for $\phi_{i,j+1}$ based on values at time j:

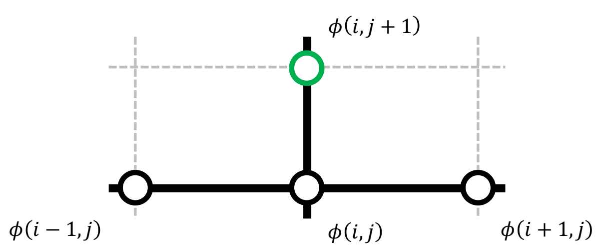
$$\phi_{i,j+1} = \frac{\kappa \Delta t}{\Delta x^2}\left(\phi_{i-1,j} + \phi_{i,j}\left(\frac{\Delta x^2}{\kappa \Delta t} - 2\right) + \phi_{i+1,j}\right)$$

Explicit update scheme, as new value is calculated from known information

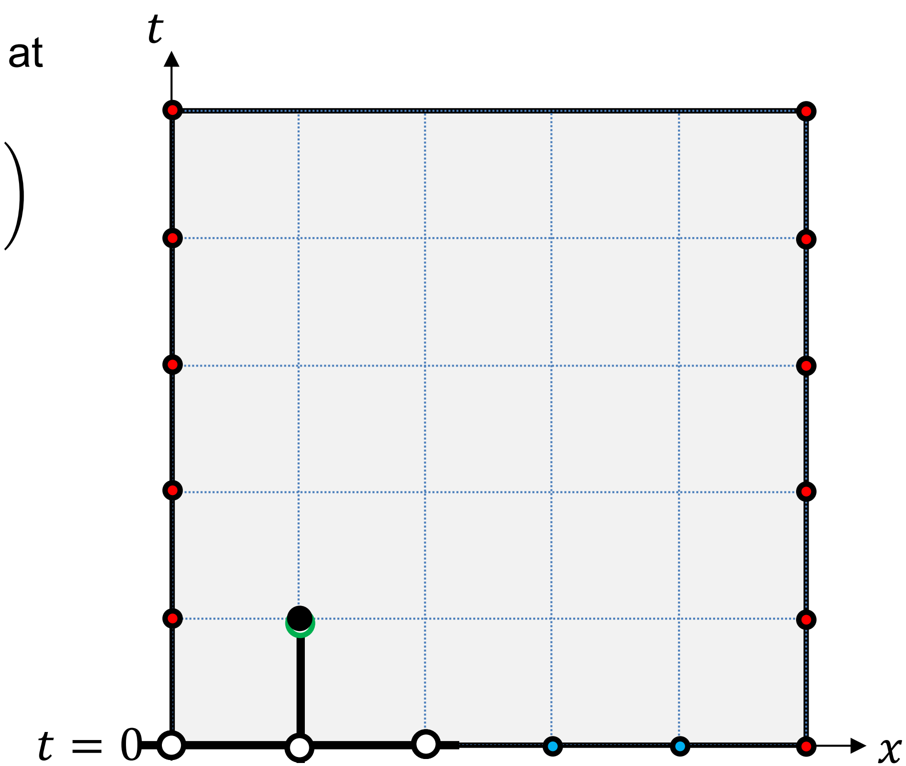
Parabolic PDE: Explicit Method

- Explicit update for $\phi_{i,j+1}$ based on values at time j:

$$\phi_{i,j+1} = \frac{\kappa \Delta t}{\Delta x^2}\left(\phi_{i-1,j} + \phi_{i,j}\left(\frac{\Delta x^2}{\kappa \Delta t} - 2\right) + \phi_{i+1,j}\right)$$



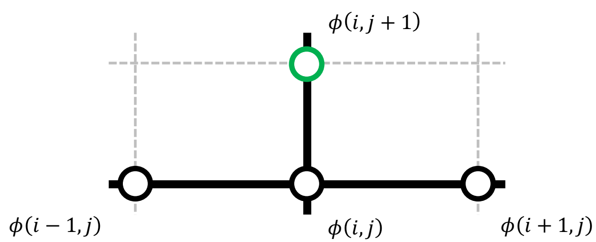
- Step 3: apply update molecule iteratively over grid nodes



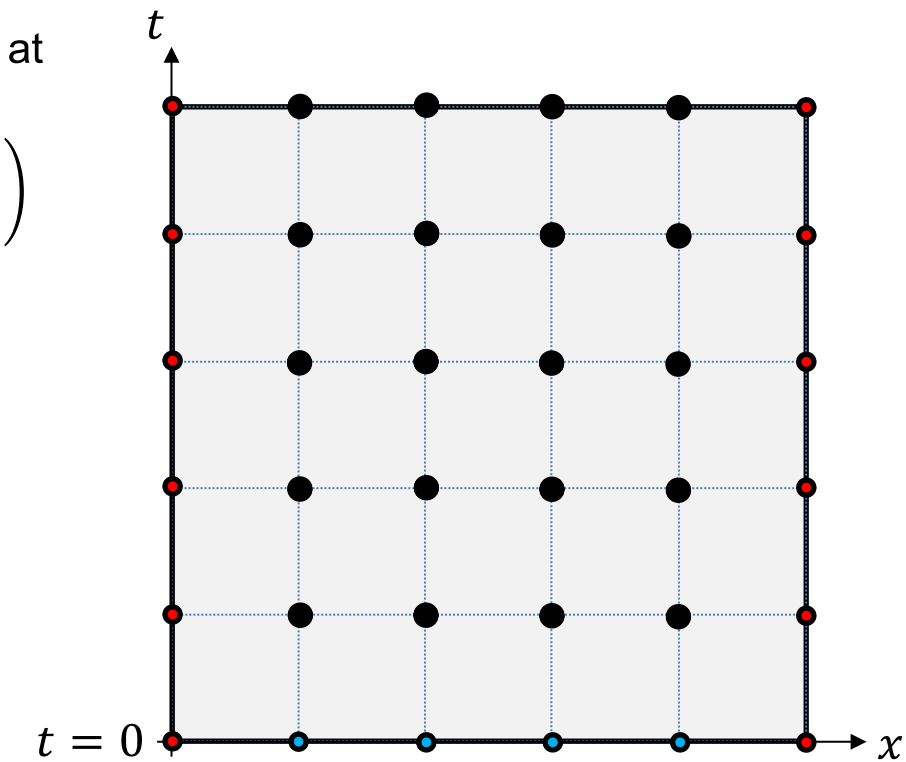
Parabolic PDE: Explicit Method

- Explicit update for $\phi_{i,j+1}$ based on values at time j:

$$\phi_{i,j+1} = \frac{\kappa \Delta t}{\Delta x^2}\left(\phi_{i-1,j} + \phi_{i,j}\left(\frac{\Delta x^2}{\kappa \Delta t} - 2\right) + \phi_{i+1,j}\right)$$



- Step 3: apply update molecule iteratively over grid nodes



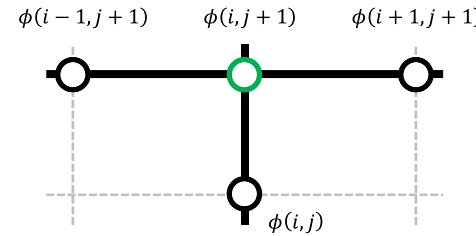
Implicit FD Approximations

- An implicit solution based on finite difference approximations can be developed by using approximations based at point $(i, j+1)$

- Repeating Step 2 from before, the spatial derivative becomes:

$$\frac{\partial^2 \phi(x,t)}{\partial x^2} = \frac{\phi_{i-1,j+1} - 2\phi_{i,j+1} + \phi_{i+1,j+1}}{\Delta x^2}$$

$$\frac{\partial \phi(x,t)}{\partial t} = \frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta t}$$



$$\frac{\phi_{i-1,j+1} - 2\phi_{i,j+1} + \phi_{i+1,j+1}}{\Delta x^2} = \frac{1}{\kappa} \frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta t}$$

- Rearranging provides an update scheme with known values on the right hand side, and unknowns on the left:

$$\frac{\kappa \Delta t}{\Delta x^2}\left(-\phi_{i-1,j+1} + \phi_{i,j+1}\left(2 + \frac{\Delta x^2}{\kappa \Delta t}\right) - \phi_{i+1,j+1}\right) = \phi_{i,j}$$

Implicit update scheme – assemble update equation for all nodes at time j, and solve simultaneously for time $j+1$

Implicit FD Solution

Apply computational molecule to all nodes at time $j = 0$:

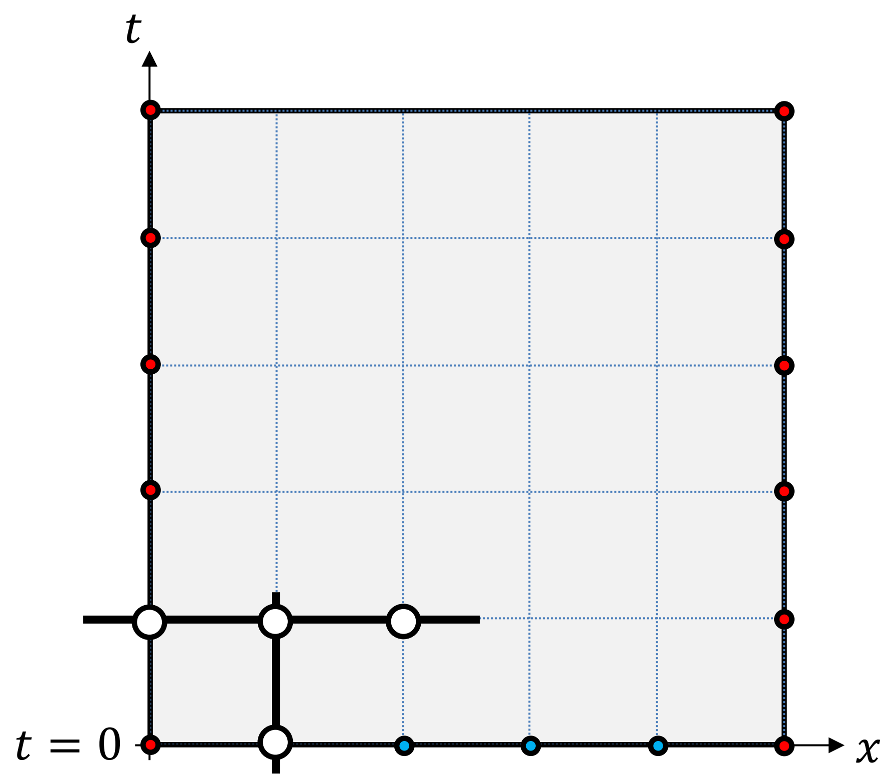
Node 1:

$$\frac{\kappa\Delta t}{\Delta x^2}\left(-\phi_{0,j+1} + \phi_{1,j+1}\left(2 + \frac{\Delta x^2}{\kappa\Delta t}\right) - \phi_{2,j+1}\right) = \phi_{1,j}$$

$$\frac{\kappa\Delta t}{\Delta x^2}\left(-10 + \phi_{1,j+1}\left(2 + \frac{\Delta x^2}{\kappa\Delta t}\right) - \phi_{2,j+1}\right) = 25$$

Collect known terms on RHS:

$$\left(2\frac{\kappa\Delta t}{\Delta x^2} + 1\right)\phi_{1,j+1} - \frac{\kappa\Delta t}{\Delta x^2}\phi_{2,j+1} = 25 + \frac{10\kappa\Delta t}{\Delta x^2}$$



Implicit FD Solution

- Collect nodal equations and formulate as matrix problem: $A\phi = b$. (Letting $\gamma = \frac{\kappa \Delta t}{\Delta x^2}$)

$$\begin{bmatrix} (2\gamma+1) & -\lambda & & \\ -\lambda & (2\gamma+1) & -\lambda & \\ & -\lambda & (2\gamma+1) & -\lambda \\ & & -\lambda & (2\gamma+1) \end{bmatrix} \begin{Bmatrix} \phi_{1,j+1} \\ \phi_{2,j+1} \\ \phi_{3,j+1} \\ \phi_{4,j+1} \end{Bmatrix} = \begin{Bmatrix} 25 + \frac{10\kappa\Delta t}{\Delta x^2} \\ 25 \\ 25 \\ 25 + \frac{10\kappa\Delta t}{\Delta x^2} \end{Bmatrix}$$

- This is a tri-diagonal system, which means it can be solved for ϕ efficiently using the Thomas algorithm

- Once solution at time $j+1$ has been found, the process can be repeated marching forward in time until the end of the solution domain is reached

- Implicit method enables larger timesteps to be taken for a given accuracy, and is unconditionally stable

COMP36212
Solving tridiagonal matrix problems

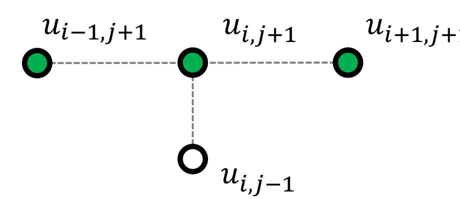
Dr Oliver Rhodes
oliver.rhodes@manchester.ac.uk

Overview

- A frequent problem in finite difference methods is the solution of systems described by a diagonal matrix A, where $Ax = b$:
  - Solving boundary value ODEs
  - Solving elliptic PDEs
  - Solving implicitly parabolic/hyperbolic PDEs

- A number of methods exist for solving such systems, including matrix inversion, Gaussian elimination, and LU decomposition, with choice of algorithm governed by:
  - Structure of matrix
  - Computational efficiency (time)
  - Memory
  - Solution accuracy

- Finite difference equations often result in tridiagonal systems:



Tridiagonal Systems

- Solve the following system for x:

$$\begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ & & -1 & 2 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} 4 \\ 2 \\ 2 \\ 10 \end{Bmatrix}$$

- System is tri-diagonal: banded with terms only on the diagonal, with maximum band width of 3. A more general form is described for a problem of size n:

$$\begin{bmatrix} \beta_1 & \gamma_1 & & \\ \alpha_2 & \beta_2 & \gamma_2 & \\ & \alpha_{\dots} & \beta_{\dots} & \gamma_{\dots} \\ & & \alpha_n & \beta_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_{\dots} \\ x_n \end{Bmatrix} = \begin{Bmatrix} s_1 \\ s_2 \\ s_{\dots} \\ s_n \end{Bmatrix}$$

- Tridiagonal systems are common in engineering and physical systems, and enable efficient storage by omitting zero terms (not on the diagonal)

MANCHESTER 1824
The University of Manchester

Thomas Algorithm

- Step 1: Decomposition of A

$$\begin{bmatrix} \beta_1 & \gamma_1 & & \\ \alpha_2 & \beta_2 & \gamma_2 & \\ & \alpha_{\dots} & \beta_{\dots} & \gamma_{\dots} \\ & & \alpha_n & \beta_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_{\dots} \\ x_n \end{Bmatrix} = \begin{Bmatrix} s_1 \\ s_2 \\ s_{\dots} \\ s_n \end{Bmatrix}$$

Decomposition pseudocode

For $i = 2, n$ do:

$\quad \alpha_i = \dfrac{\alpha_i}{\beta_{i-1}}$

$\quad \beta_i = \beta_i - \alpha_i \gamma_{i-1}$

end

- Apply to example problem:

$$\begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ & & -1 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & -1 & & \\ -0.5 & 1.5 & -1 & \\ & & & \\ & & & \end{bmatrix}$$

For $i = 2$:

$$\alpha_2 = \frac{\alpha_2}{\beta_1} = \frac{-1}{2} = -0.5$$

$$\beta_2 = \beta_2 - \alpha_2 \gamma_1$$
$$= 2 - (-0.5 \times -1) = 1.5$$

The University of Manchester

Thomas Algorithm

- Step 1: Decomposition of A

$$\begin{bmatrix} \beta_1 & \gamma_1 & & \\ \alpha_2 & \beta_2 & \gamma_2 & \\ & \alpha_{\dots} & \beta_{\dots} & \gamma_{\dots} \\ & & \alpha_n & \beta_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_{\dots} \\ x_n \end{Bmatrix} = \begin{Bmatrix} s_1 \\ s_2 \\ s_{\dots} \\ s_n \end{Bmatrix}$$

Decomposition pseudocode

For $i = 2, n$ do:

$$\alpha_i = \frac{\alpha_i}{\beta_{i-1}}$$

$$\beta_i = \beta_i - \alpha_i \gamma_{i-1}$$

end

- Apply to example problem:

$$\begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ & & -1 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & -1 & & \\ -0.5 & 1.5 & -1 & \\ & -0.667 & 1.333 & -1 \\ & & & \end{bmatrix}$$

For $i = 3$:

$$\alpha_3 = \frac{\alpha_3}{\beta_2} = \frac{-1}{1.5} = -0.667$$

$$\beta_3 = \beta_3 - \alpha_3 \gamma_2$$

$$= 2 - (-0.667 \times -1) = 1.333$$

Thomas Algorithm

- Step 1: Decomposition of A

$$\begin{bmatrix} \beta_1 & \gamma_1 & & \\ \alpha_2 & \beta_2 & \gamma_2 & \\ & \alpha_{\dots} & \beta_{\dots} & \gamma_{\dots} \\ & & \alpha_n & \beta_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_{\dots} \\ x_n \end{Bmatrix} = \begin{Bmatrix} s_1 \\ s_2 \\ s_{\dots} \\ s_n \end{Bmatrix}$$

Decomposition pseudocode

`For` $i = 2, n$ `do:`

$$\alpha_i = \frac{\alpha_i}{\beta_{i-1}}$$

$$\beta_i = \beta_i - \alpha_i \gamma_{i-1}$$

`end`

- Apply to example problem:

$$\begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ & & -1 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & -1 & & \\ -0.5 & 1.5 & -1 & \\ & -0.667 & 1.333 & -1 \\ & & -0.75 & 1.25 \end{bmatrix}$$

For $i = 4$:

$$\alpha_4 = \frac{\alpha_4}{\beta_3} = \frac{-1}{1.333} = -0.750$$

$$\beta_4 = \beta_4 - \alpha_4 \gamma_3$$
$$= 2 - (-0.75 \times -1) = 1.25$$

The University of Manchester

Thomas Algorithm

- Step 2: Forward substitution

$$\begin{bmatrix} \beta_1 & \gamma_1 & & \\ \alpha_2 & \beta_2 & \gamma_2 & \\ & \alpha_{\dots} & \beta_{\dots} & \gamma_{\dots} \\ & & \alpha_n & \beta_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_{\dots} \\ x_n \end{Bmatrix} = \begin{Bmatrix} s_1 \\ s_2 \\ s_{\dots} \\ s_n \end{Bmatrix}$$

Forward sub. pseudocode

For $i = 2, n$ do:

$s_i = s_i - \alpha_i s_{i-1}$

end

- Apply to example problem:

$$A = \begin{bmatrix} 2 & -1 & & \\ -0.5 & 1.5 & -1 & \\ & -0.667 & 1.333 & -1 \\ & & -0.75 & 1.25 \end{bmatrix}, \quad s = \begin{bmatrix} 4 \\ 2 \\ 2 \\ 10 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 \\ 4 \\ \\ \end{bmatrix}$$

For $i = 2$:

$s_2 = 2 - (-0.5 \times 4) = 4$

Thomas Algorithm

- Step 2: Forward substitution

$$\begin{bmatrix} \beta_1 & \gamma_1 & & \\ \alpha_2 & \beta_2 & \gamma_2 & \\ & \alpha_{\dots} & \beta_{\dots} & \gamma_{\dots} \\ & & \alpha_n & \beta_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_{\dots} \\ x_n \end{Bmatrix} = \begin{Bmatrix} s_1 \\ s_2 \\ s_{\dots} \\ s_n \end{Bmatrix}$$

Forward sub. pseudocode

For $i = 2, n$ do:

$\quad s_i = s_i - \alpha_i s_{i-1}$

end

- Apply to example problem:

$$A = \begin{bmatrix} 2 & -1 & & \\ -0.5 & 1.5 & -1 & \\ & -0.667 & 1.333 & -1 \\ & & -0.75 & 1.25 \end{bmatrix}, \qquad s = \begin{bmatrix} 4 \\ 2 \\ 2 \\ 10 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 \\ 4 \\ 4.667 \\ \end{bmatrix}$$

For $i = 3$:

$s_3 = 2 - (-0.667 \times 4) = 4.667$

The University of Manchester

Thomas Algorithm

- Step 2: Forward substitution

$$\begin{bmatrix} \beta_1 & \gamma_1 & & \\ \alpha_2 & \beta_2 & \gamma_2 & \\ & \alpha_{\dots} & \beta_{\dots} & \gamma_{\dots} \\ & & \alpha_n & \beta_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_{\dots} \\ x_n \end{Bmatrix} = \begin{Bmatrix} s_1 \\ s_2 \\ s_{\dots} \\ s_n \end{Bmatrix}$$

Forward sub. pseudocode

For $i = 2, n$ do:

$s_i = s_i - \alpha_i s_{i-1}$

end

- Apply to example problem:

$$A = \begin{bmatrix} 2 & -1 & & \\ -0.5 & 1.5 & -1 & \\ & -0.667 & 1.333 & -1 \\ & & -0.75 & 1.25 \end{bmatrix}, \quad s = \begin{bmatrix} 4 \\ 2 \\ 2 \\ 10 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 \\ 4 \\ 4.667 \\ 13.5 \end{bmatrix}$$

For $i = 4$:

$s_4 = 10.0 - (-0.75 \times 4.667)$

$= 13.5$

Thomas Algorithm

- Step 3: Backward substitution

$$\begin{bmatrix} \beta_1 & \gamma_1 & & \\ \alpha_2 & \beta_2 & \gamma_2 & \\ & \alpha_{\dots} & \beta_{\dots} & \gamma_{\dots} \\ & & \alpha_n & \beta_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_{\dots} \\ x_n \end{Bmatrix} = \begin{Bmatrix} s_1 \\ s_2 \\ s_{\dots} \\ s_n \end{Bmatrix}$$

- Apply to example problem:

$$\begin{bmatrix} 2 & -1 & & \\ -0.5 & 1.5 & -1 & \\ & -0.667 & 1.333 & -1 \\ & & -0.75 & 1.25 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 \\ 4 \\ 4.667 \\ 13.5 \end{bmatrix}$$

Backward sub. pseudocode

$x_n = \dfrac{s_n}{\beta_n}$

`For` $i = (n-1), 1, (i-1)$

`do:`

$\quad x_i = (s_i - \gamma_i x_{i+1})/\beta\_i$

`end`

Thomas Algorithm

- Step 3: Backward substitution

$$\begin{bmatrix} \beta_1 & \gamma_1 & & \\ \alpha_2 & \beta_2 & \gamma_2 & \\ & \alpha_{\dots} & \beta_{\dots} & \gamma_{\dots} \\ & & \alpha_n & \beta_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_{\dots} \\ x_n \end{Bmatrix} = \begin{Bmatrix} s_1 \\ s_2 \\ s_{\dots} \\ s_n \end{Bmatrix}$$

- Apply to example problem:

$$\begin{bmatrix} 2 & -1 & & \\ -0.5 & 1.5 & -1 & \\ & -0.667 & 1.333 & -1 \\ & & -0.75 & 1.25 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 10.8 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 \\ 4 \\ 4.667 \\ 13.5 \end{bmatrix}$$

Backward sub. pseudocode

$x_n = \dfrac{s_n}{\beta_n}$

For $i = (n-1), 1, (i-1)$

do:

$\quad x_i = (s_i - \gamma_i x_{i+1})/\beta\_i$

end

$$x_4 = \frac{s_4}{\beta_4} = \frac{13.5}{1.25} = 10.8$$

Thomas Algorithm

- Step 3: Backward substitution

$$\begin{bmatrix} \beta_1 & \gamma_1 & & \\ \alpha_2 & \beta_2 & \gamma_2 & \\ & \alpha_{\dots} & \beta_{\dots} & \gamma_{\dots} \\ & & \alpha_n & \beta_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_{\dots} \\ x_n \end{Bmatrix} = \begin{Bmatrix} s_1 \\ s_2 \\ s_{\dots} \\ s_n \end{Bmatrix}$$

- Apply to example problem:

$$\begin{bmatrix} 2 & -1 & & \\ -0.5 & 1.5 & -1 & \\ & -0.667 & 1.333 & -1 \\ & & -0.75 & 1.25 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 11.6 \\ 10.8 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 \\ 4 \\ 4.667 \\ 13.5 \end{bmatrix}$$

Backward sub. pseudocode

$x_n = \dfrac{s_n}{\beta_n}$

For $i = (n-1), 1, (i-1)$

do:

$\quad x_i = (s_i - \gamma_i x_{i+1})/\beta\_i$

end

For $i = 3$:

$x_3 = (4.667 - (-1 \times 10.8))/1.333$

$\quad = 11.6$

Thomas Algorithm

The University of Manchester

- Step 3: Backward substitution

$$\begin{bmatrix} \beta_1 & \gamma_1 & & \\ \alpha_2 & \beta_2 & \gamma_2 & \\ & \alpha_{\dots} & \beta_{\dots} & \gamma_{\dots} \\ & & \alpha_n & \beta_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_{\dots} \\ x_n \end{Bmatrix} = \begin{Bmatrix} s_1 \\ s_2 \\ s_{\dots} \\ s_n \end{Bmatrix}$$

- Apply to example problem:

$$\begin{bmatrix} 2 & -1 & & \\ -0.5 & 1.5 & -1 & \\ & -0.667 & 1.333 & -1 \\ & & -0.75 & 1.25 \end{bmatrix} \begin{bmatrix} x_1 \\ 10.4 \\ 11.6 \\ 10.8 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 \\ 4 \\ 4.667 \\ 13.5 \end{bmatrix}$$

Backward sub. pseudocode

$x_n = \dfrac{s_n}{\beta_n}$

`For` $i = (n-1), 1, (i-1)$

`do:`

$\quad x_i = (s_i - \gamma_i x_{i+1})/\beta\_i$

`end`

For $i = 2$:

$x_2 = (4 - (-1 \times 11.6))/1.5$

$\quad = 10.4$

Thomas Algorithm

- Step 3: Backward substitution

$$\begin{bmatrix} \beta_1 & \gamma_1 & & \\ \alpha_2 & \beta_2 & \gamma_2 & \\ & \alpha_{\dots} & \beta_{\dots} & \gamma_{\dots} \\ & & \alpha_n & \beta_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_{\dots} \\ x_n \end{Bmatrix} = \begin{Bmatrix} s_1 \\ s_2 \\ s_{\dots} \\ s_n \end{Bmatrix}$$

- Apply to example problem:

$$\begin{bmatrix} 2 & -1 & & \\ -0.5 & 1.5 & -1 & \\ & -0.667 & 1.333 & -1 \\ & & -0.75 & 1.25 \end{bmatrix} \begin{bmatrix} 7.2 \\ 10.4 \\ 11.6 \\ 10.8 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 \\ 4 \\ 4.667 \\ 13.5 \end{bmatrix}$$

Backward sub. pseudocode

$x_n = \dfrac{s_n}{\beta_n}$

`For` $i = (n-1), 1, (i-1)$
`do:`
    $x_i = (s_i - \gamma_i x_{i+1})/\beta\_i$
`end`

For $i = 1$:
$x_1 = (4 - (-1 \times 10.4))/2$
$= 7.2$

Check

- Final solution from Thomas Algorithm:

$$\begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ & & -1 & 2 \end{bmatrix} \begin{bmatrix} 7.2 \\ 10.4 \\ 11.6 \\ 10.8 \end{bmatrix} \Rightarrow \begin{Bmatrix} 4 \\ 2 \\ 2 \\ 10 \end{Bmatrix}$$

- Multiply out rows to check solution:

$$2\times7.2 \; - 1\times10.4 = 4$$
$$-0.5 \times7.2 + 1.5\times10.4 - 1\times11.6 = 2$$
$$-1\times10.4 + 2\times11.6 \; - 1\times10.8 = 2$$
$$-1\times11.6 + 2\times10.8 = 10$$

Summary

- Thomas algorithm provides an efficient solution for tridiagonal systems

- Consists of three steps: decomposition, forward substitution, and backward substitution

- Decomposition phase can be applied once, and used to evaluate multiple right hand side vectors (b)

- Enables the matrix to store only three values per row, reducing memory footprint for large systems (relative to storing sparse matrix)

COMP36212
Solving the Hyperbolic Wave Equation

Dr Oliver Rhodes
oliver.rhodes@manchester.ac.uk

Overview

- Solution of hyperbolic PDEs – wave equation
- Application of derivative boundary conditions
- Example: vibration of a string

Wave Equation

- The hyperbolic PDE known as the wave equation is defined as:

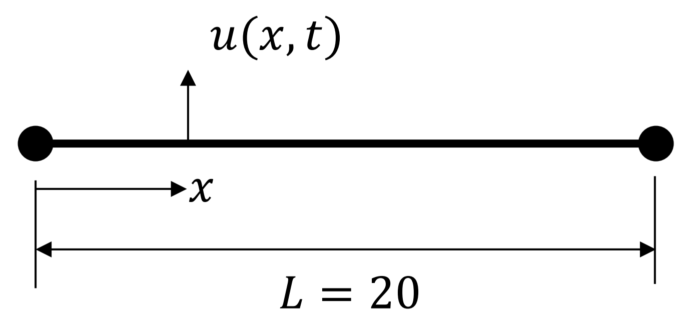
$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2}$$

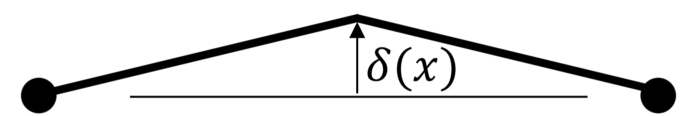
- Solve for auxiliary conditions (assume $c^2 = 1$, and remains constant):
  - $u(x = 0, t) = 0$
  - $u(x = L, t) = 0$
  - $u\left(x = \frac{L}{2}, 0\right) = \delta(x)$
  - $\left[\frac{\partial u}{\partial t}\right]_{t=0} = g(x)$

- Analytical Solution:

$$u(x,t) = \frac{8}{\pi^2}\sum_{n=1}^{\infty}\frac{1}{n^2}\sin\left(\frac{n\pi x}{20}\right)\sin\left(\frac{n\pi}{2}\right)\cos\left(\frac{n\pi t}{20}\right)$$

Vibration on a string





Initial conditions

Solve via FDs

- Substitute finite difference approximations for derivatives – use central difference based approximations

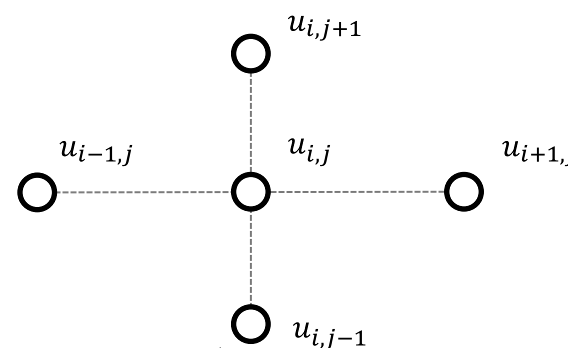
$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2}$$

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta x^2} = \frac{1}{c^2}\frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta t^2}$$

$$u_{i-1,j} - 2u_{i,j} + u_{i+1,j} = \frac{\Delta x^2}{c^2 \Delta t^2}\left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right)$$

- Collect terms and rearrange to provide explicit update (let $\lambda = \frac{c^2 \Delta t^2}{\Delta x^2}$):

$$\lambda u_{i-1,j} - u_{i,j}(2\lambda - 2) + \lambda u_{i+1,j} - u_{i,j-1} = u_{i,j+1}$$



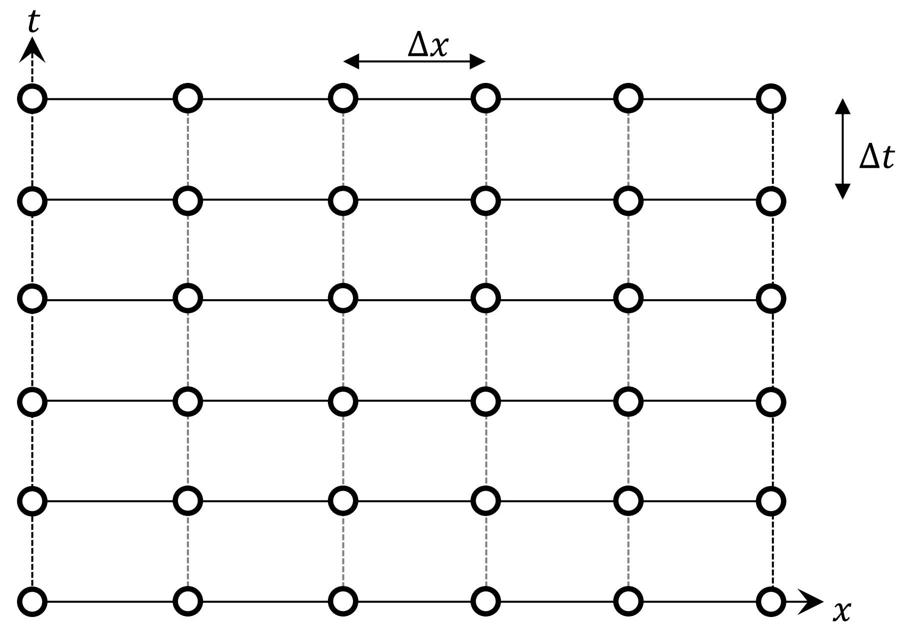
How to deal with $u_{i,j-1}$ term?

Applying Boundary Conditions

- Given the update equation:

$$\lambda u_{i-1,j} - u_{i,j}(2\lambda - 2) + \lambda u_{i+1,j} - u_{i,j-1} = u_{i,j+1}$$

- Construct the computational grid, and substitute in known values

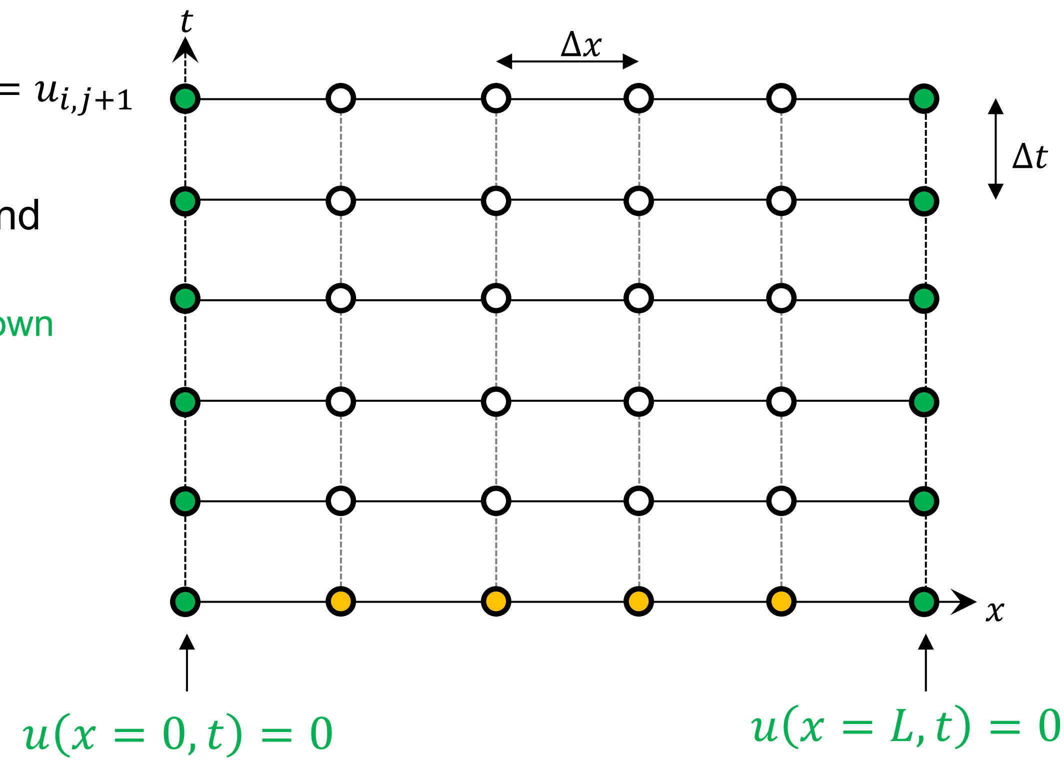


Applying Boundary Conditions

- Given the update equation:

$$\lambda u_{i-1,j} - u_{i,j}(2\lambda - 2) + \lambda u_{i+1,j} - u_{i,j-1} = u_{i,j+1}$$

- Construct the computational grid, and substitute in known values
  - Values at green filled nodes are known from boundary conditions,

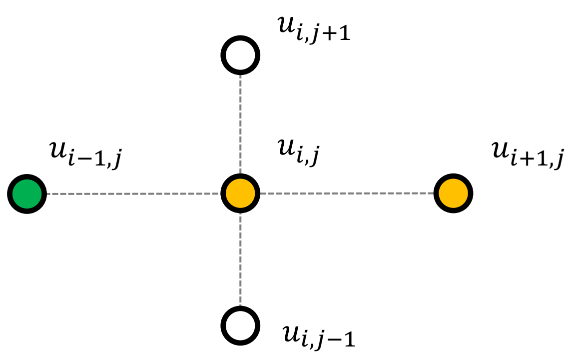


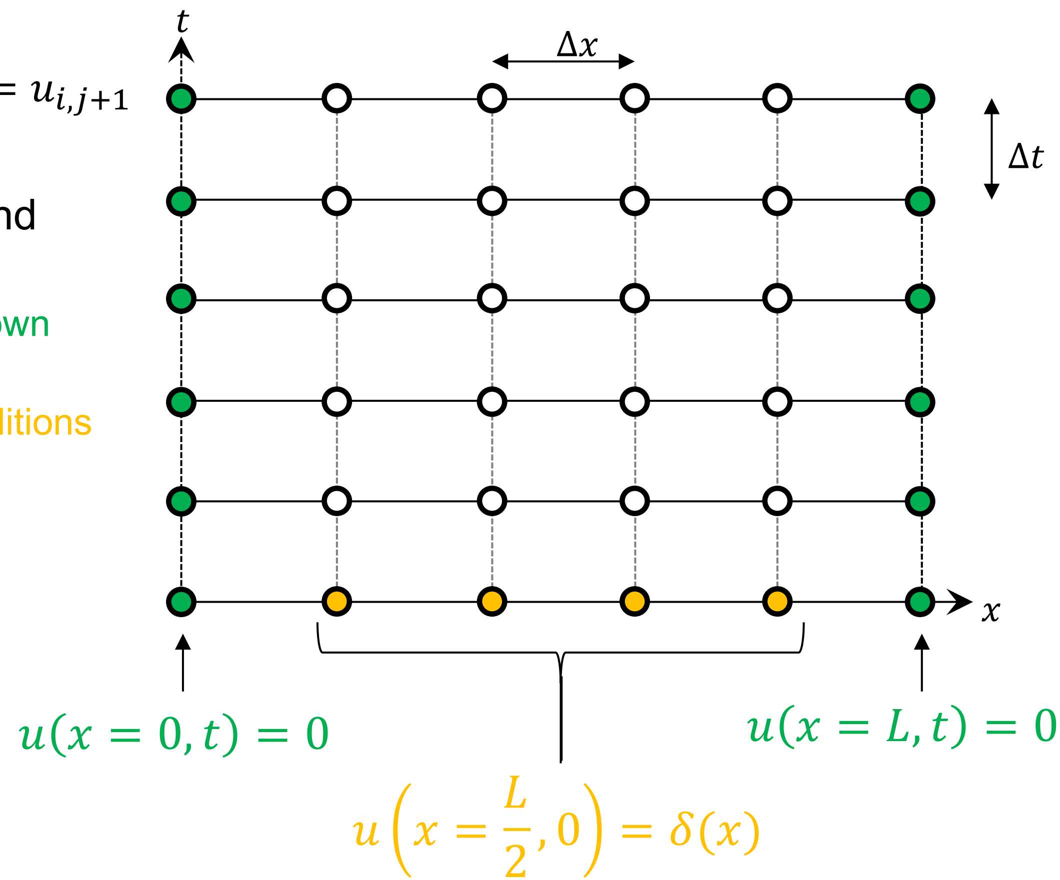
Applying Initial Conditions

- Given the update equation:

$$\lambda u_{i-1,j} - u_{i,j}(2\lambda - 2) + \lambda u_{i+1,j} - u_{i,j-1} = u_{i,j+1}$$

- Construct the computational grid, and substitute in known values
  - Values at green filled nodes are known from boundary conditions
  - Orange filled nodes from initial conditions





Apply Derivative ICs

- Use initial condition: velocity at time $t = 0$ is defined as $g(x) = 0$

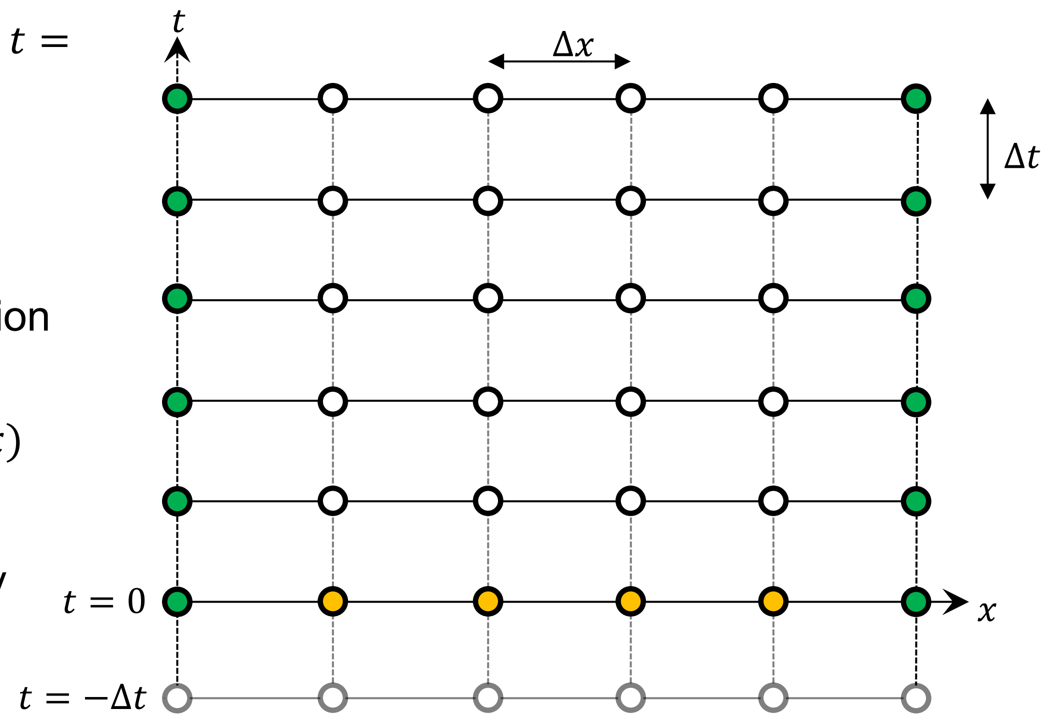
$$\left[\frac{\partial u}{\partial t}\right]_{t=0} = g(x) = 0$$

- Using central difference approximation for initial condition derivative:

$$\frac{u_{i,j+1} - u_{i,j-1}}{2\Delta t} = \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta t} = g(x)$$

- Approximation supported by dummy nodes. Rearranging:

$$u_{i,j-1} = u_{i,j+1} - 2\Delta t g(x)$$
$$u_{i,j-1} = u_{i,j+1}$$

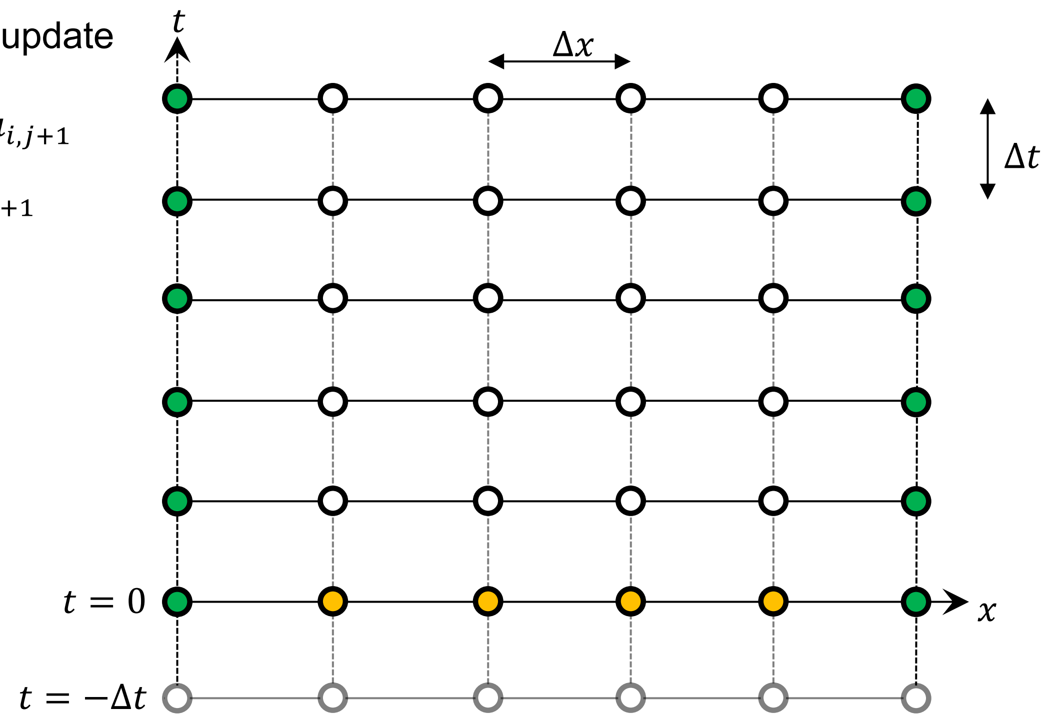


Apply Derivative ICs

- Substituting in this additional information update equation becomes:

$$\lambda u_{i-1,j} - u_{i,j}(2\lambda - 2) + \lambda u_{i+1,j} - u_{i,j+1} = u_{i,j+1}$$

$$\frac{1}{2}\left(\lambda u_{i-1,j} - u_{i,j}(2\lambda - 2) + \lambda u_{i+1,j}\right) = u_{i,j+1}$$



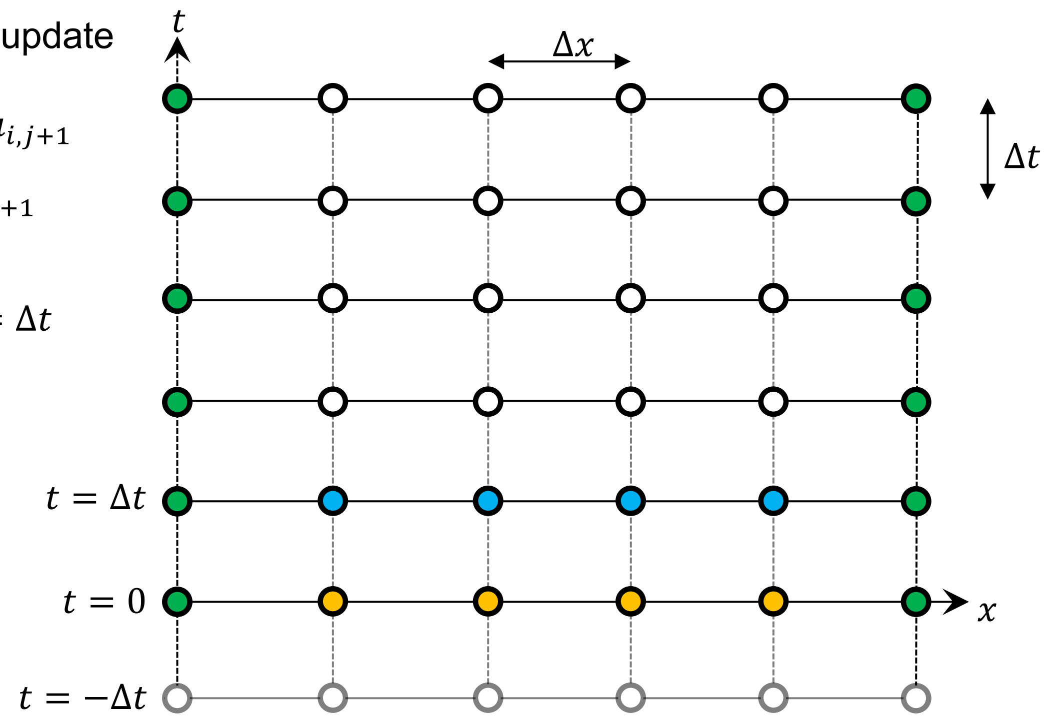
Apply Derivative ICs

- Substituting in this additional information update equation becomes:

$$\lambda u_{i-1,j} - u_{i,j}(2\lambda - 2) + \lambda u_{i+1,j} - u_{i,j+1} = u_{i,j+1}$$

$$\frac{1}{2}\left(\lambda u_{i-1,j} - u_{i,j}(2\lambda - 2) + \lambda u_{i+1,j}\right) = u_{i,j+1}$$

- Enables evaluation of solution at time $t = \Delta t$



MANCHESTER 1824
The University of Manchester

Apply Derivative ICs

- Substituting in this additional information update equation becomes:

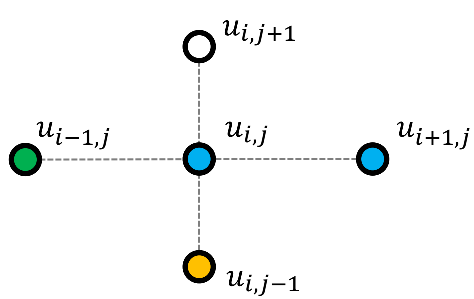
$$\lambda u_{i-1,j} - u_{i,j}(2\lambda - 2) + \lambda u_{i+1,j} - u_{i,j+1} = u_{i,j+1}$$

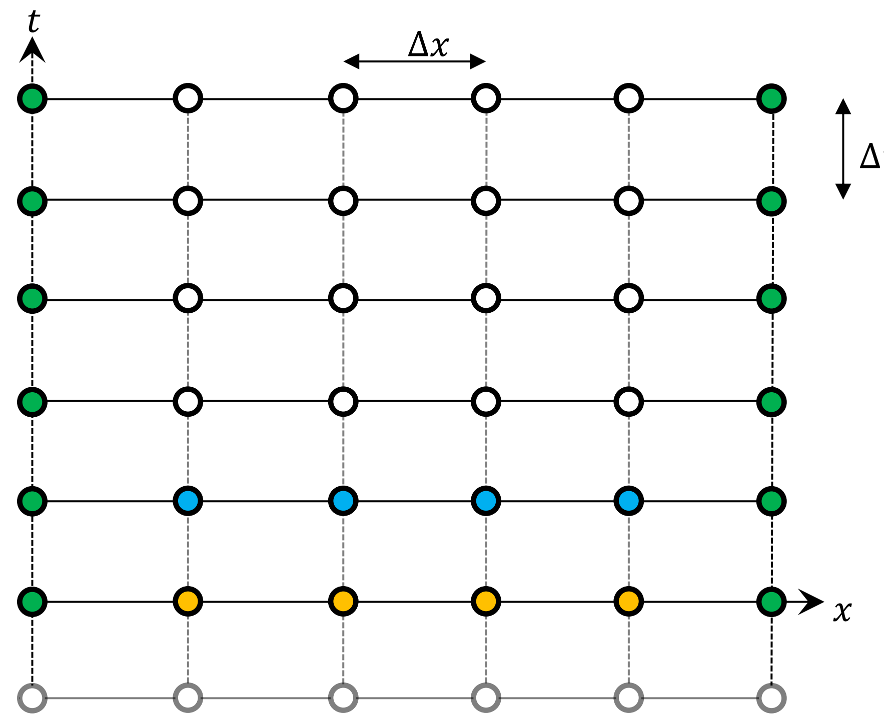
$$\frac{1}{2}\left(\lambda u_{i-1,j} - u_{i,j}(2\lambda - 2) + \lambda u_{i+1,j}\right) = u_{i,j+1}$$

- Enables evaluation of solution at time $t = \Delta t$

- Update equation can be used from there forward in time

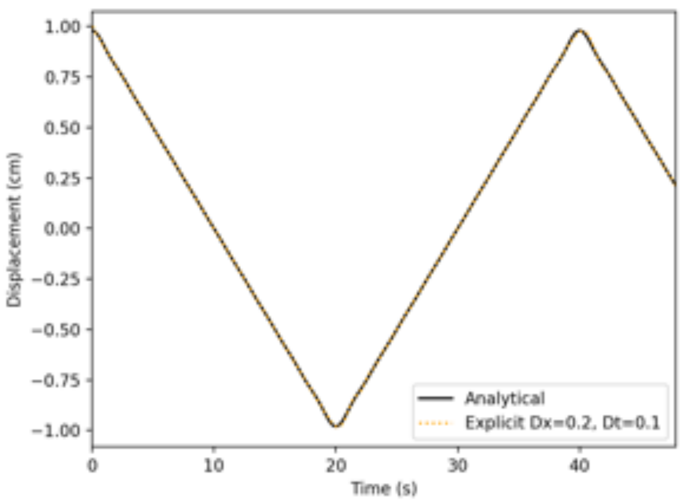
$$\lambda u_{i-1,j} - u_{i,j}(2\lambda - 2) + \lambda u_{i+1,j} - u_{i,j-1} = u_{i,j+1}$$

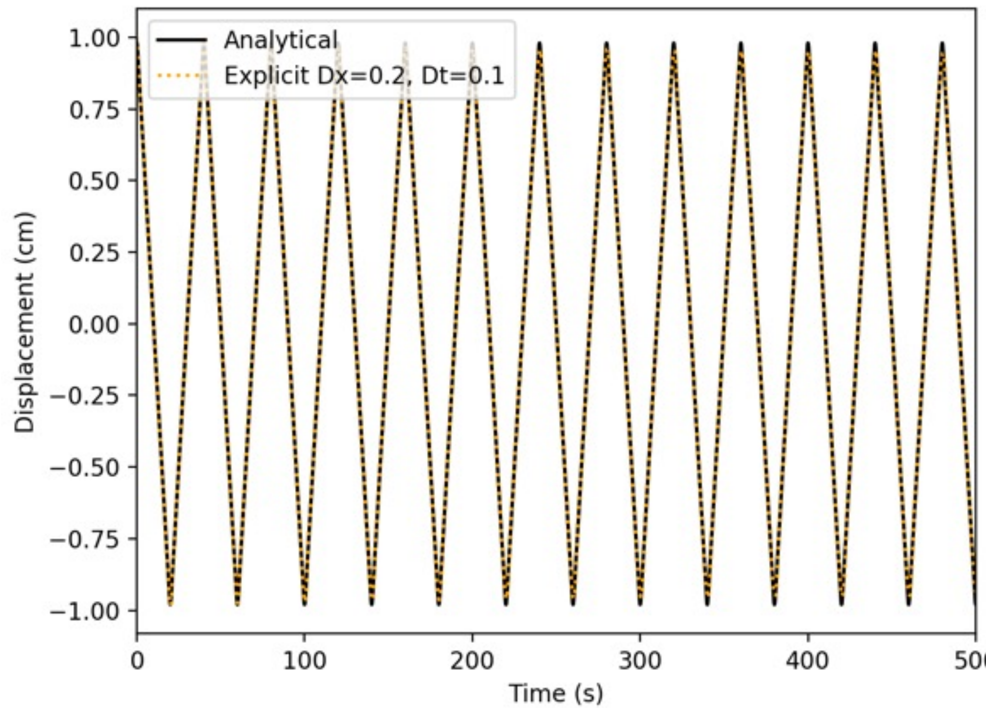




Explicit Solution

- Explicit method produces accurate results for $\Delta x = 0.2$, $\Delta t = 0.1$





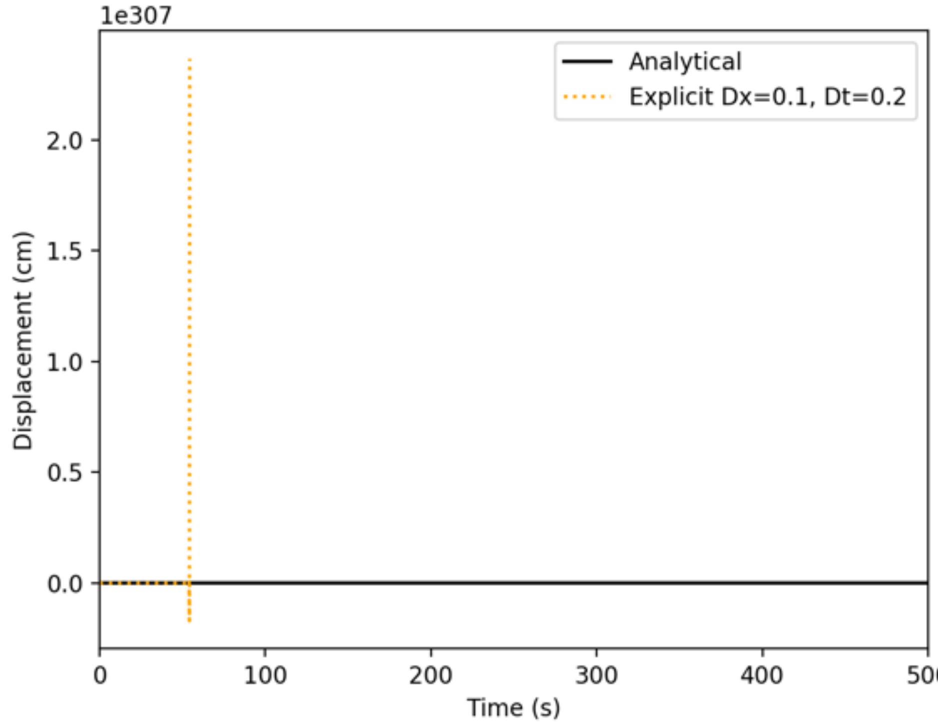
Explicit Method Stability

- Solution is unstable for certain step sizes: e.g. $\Delta x = 0.1, \Delta t = 0.2$

- It can be shown that for convergence (stability):

$$\Delta t \leq \frac{\Delta x}{c}$$

- Known as the Courant-Friedichs-Lewy (CFL) condition, it provides a bound for stability (but not accuracy) on the step size



Summary

- Explored solution of hyperbolic PDEs in the form of the wave equation
- Solved explicitly for the displacement of a vibrating string
- Used dummy nodes to implement *derivative boundary conditions*
- Explicit methods require small steps sizes to ensure accuracy and stability

Course Structure