Numerical solution of partial differential equations

Dr. Louise Olsen-Kettle
The University of Queensland
School of Earth Sciences
Centre for Geoscience Computing

E–mail: l.kettle1@uq.edu.au
Web: http://researchers.uq.edu.au/researcher/768
@DrOlsenKettle

ISBN: 978-1-74272-149-1

Acknowledgements

Special thanks to Cinnamon Eliot who helped typeset these lecture notes in LaTeX.

Note to reader
This document and code for the examples can be downloaded from

`http://espace.library.uq.edu.au/view/UQ:239427`.

Please note if any of the links to code are not working please contact my email address and I can send you the code.

Contents

List of Figures

Numerical solution of parabolic and hyperbolic PDEs

This document and code for the examples can be downloaded from

`http://espace.library.uq.edu.au/view/UQ:239427`.

Please contact me (l.kettle1@uq.edu.au) if you are unable to download the code and I can send it to you.

References:

- *Applied Numerical Methods for Engineers using Matlab and C*, R. J. Schilling and S. L. Harris.
- *Computational Physics Problem solving with computers*, R.H. Landau and M. L. Páez.
- *An Introduction to Computational Physics*, T. Pang.
- *Numerical Recipes in Fortran (2nd Ed.)*, W. H. Press *et al.*
- *Introduction to Partial Differential Equations with Matlab*, J. M. Cooper.
- *Numerical solution of partial differential equations*, K. W. Morton and D. F. Mayers.
- *Spectral methods in Matlab*, L. N. Trefethen

Chapter 1

Overview of PDEs

1.1 Classification of PDEs

The classification of PDEs is important for the numerical solution you choose.

$$A(x,y)U_{xx} + 2B(x,y)U_{xy} + C(x,y)U_{yy} \quad = \quad F(x,y,U_x,U_y,U)$$

1.1.1 Elliptic

$$AC \quad > \quad B^2$$

For example, Laplace's equation:

$$U_{xx} + U_{yy} \quad = \quad 0$$

$A = C = 1,\ B = 0$

1.1.2 Hyperbolic

$$AC \quad < \quad B^2$$

For example the 1-D wave equation:

$$U_{xx} \quad = \quad \frac{1}{c^2}U_{tt}$$

$A = 1, C = -1/c^2, B = 0$

1.1.3 Parabolic

$$AC = B^2$$

For example, the heat or diffusion Equation

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

$A = 1, B = C = 0$

1.2 Implicit Vs Explicit Methods to Solve PDEs

Explicit Methods:

- possible to solve (at a point) directly for all unknown values in the finite difference scheme.

- stable only for certain time step sizes (or possibly never stable!). Stability can be checked using Fourier or von Neumann analysis. Time step size governed by *Courant condition* for wave equation.

Implicit Methods:

- there is no explicit formula at each point, only a set of simultaneous equations which must be solved over the whole grid.

- Implicit methods are stable for all step sizes.

1.3 Well-posed and ill-posed PDEs

The heat equation is *well-posed* $U_t = U_{xx}$. However the *backwards* heat equation is *ill-posed*: $U_t = -U_{xx} \Rightarrow$ at high frequencies this blows up!

In order to demonstrate this we let $U(x,t) = a_n(t)\sin(nx)$
then:

$$U_{xx} = -a_n(t)n^2\sin(nx), \qquad \text{and} \qquad U_t = \dot{a}_n(t)\sin(nx)$$

$$\underbrace{U_t = U_{xx}}_{\text{Heat Equation}} \Rightarrow \dot{a}_n(t)\sin(nx) = -a_n(t)n^2\sin(nx)$$

$$\dot{a}_n = -a_n n^2 \Rightarrow a_n(t) = a_n(0)e^{-n^2 t}$$

For the heat equation the transient part of the solution decays and this has stable numerical solutions.

$$\underbrace{U_t = -U_{xx}}_{\text{Backwards Heat Equation}} \qquad \Rightarrow \dot{a}_n(t)\sin(nx) = a_n(t)n^2\sin(nx)$$

$$\dot{a}_n = a_n n^2 \Rightarrow a_n(t) = a_n(0)e^{n^2 t}$$

For the backwards heat equation the transient part of the solution blows up and the numerical solution would fail! In general it is difficult or impossible to obtain numerical solutions for ill-posed PDEs.

Part I

Numerical solution of parabolic equations

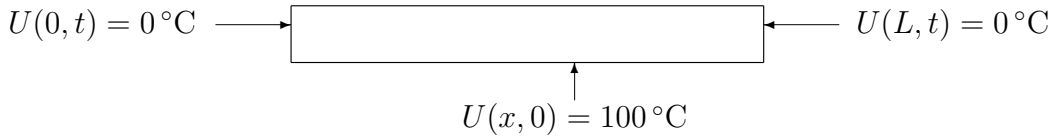
Chapter 2

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

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

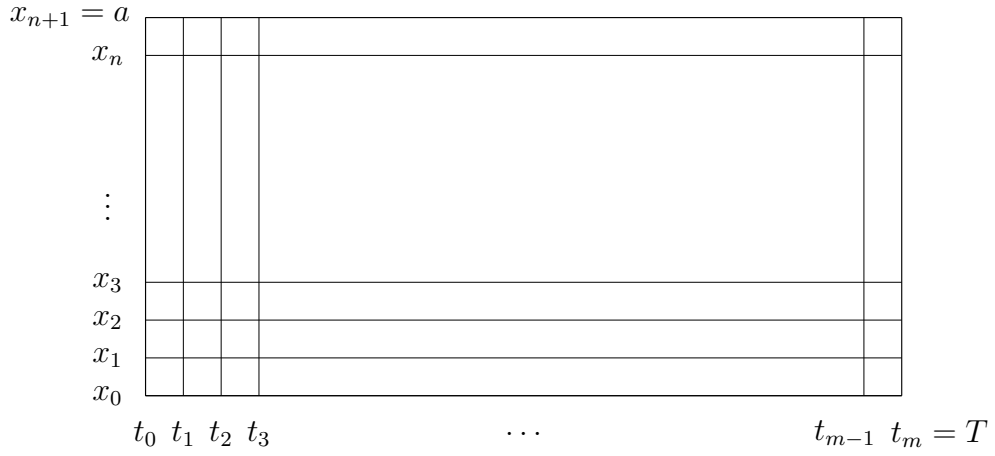
2.2.3 Explicit Forward Euler method

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:

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

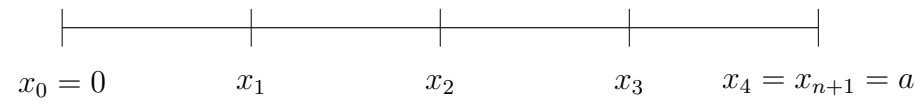
Numerical implementation of Explicit Forward Euler method

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

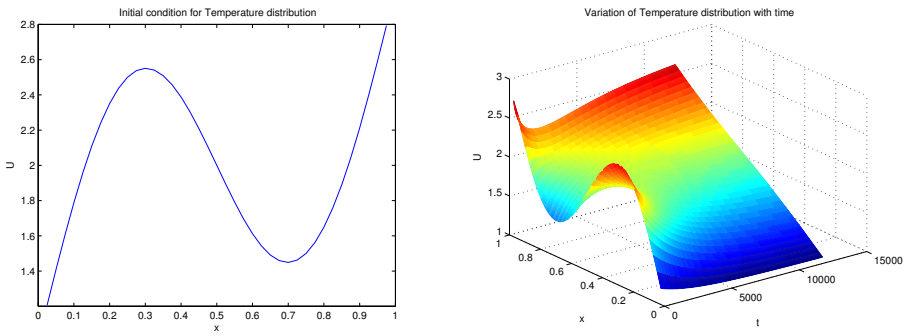


Figure 2.1: Initial conditions in (a) and matlab solution using Forward Euler method for temperature distribution along rod with time in (b)

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:

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

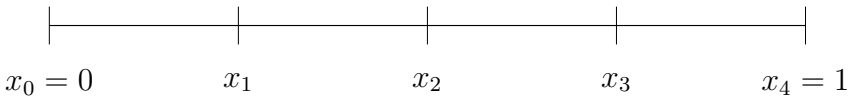
How? Replace

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

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**.

Chapter 3

Implicit methods for 1-D heat equation

3.1 Implicit Backward Euler Method for 1-D heat equation

- Unconditionally stable (but usually slower than explicit methods).
- *implicit* because it evaluates difference approximations to derivatives at next time step t_{k+1} and not current time step we are solving for t_k.

$$\begin{aligned} U_{xx}(t_{k+1}, x_j) &= \frac{U_{j+1}^{k+1} - 2U_j^{k+1} + U_{j-1}^{k+1}}{\Delta x^2} \\ U_t(t_{k+1}, x_j) &= \frac{U_j^{k+1} - U_j^k}{\Delta t} \end{aligned}$$

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

$$\begin{aligned} U_j^k &= U_j^{k+1} - \frac{\beta \Delta t}{\Delta x^2} \quad [U_{j+1}^{k+1} - 2U_j^{k+1} + U_{j-1}^{k+1}] \\ &= (1+2s)U_j^{k+1} - s(U_{j+1}^{k+1} + U_{j-1}^{k+1}) \end{aligned} \tag{3.1}$$

where $s = \frac{\beta \Delta t}{\Delta x^2}$ as before.
We still need to solve for U_j^{k+1} given U_j^k is known $\Rightarrow$ This requires solving a tridiagonal linear system of n equations.
Again we let $U_j^k = U(x_j, t_k)$; $x_j = j\Delta x, j = 0, ..., n+1, \Delta x = \frac{a}{n+1}$; $t_k = k\Delta t, k = 0, ..., m$, and $\Delta t = \frac{T}{m}$.

3.1.1 Numerical implementation of the Implicit Backward Euler Method

Again we are solving the same problem: $U_t = \beta U_{xx}, \quad U(x,0) = U_j^0 = f(x_j)$

3.1.2 Dirichlet boundary conditions

$U(0,t) = 1 = U_0^k, \quad U(a,t) = U(1,t) = 3 = U_{n+1}^k = U_4^k$

For simplicity we consider only 4 elements in x in this example to find the matrix system we need to solve for:

$x_0 = 0 \qquad x_1 = 0.25 \qquad x_2 = 0.5 \qquad x_3 = 0.75 \qquad x_4 = 1$

Rewriting $-s[U_{j+1}^{k+1} + U_{j-1}^{k+1}] + (1+2s)U_j^{k+1} = U_j^k$ as a matrix equation:

$$\underbrace{\begin{pmatrix} 1+2s & -s & 0 \\ -s & 1+2s & -s \\ 0 & -s & 1+2s \end{pmatrix}}_{\text{Tridiagonal matrix}} \underbrace{\begin{pmatrix} U_1^{k+1} \\ U_2^{k+1} \\ U_3^{k+1} \end{pmatrix}}_{\text{Solution } U \text{ at next time step}} = \begin{pmatrix} U_1^k \\ U_2^k \\ U_3^k \end{pmatrix} + s \underbrace{\begin{pmatrix} U_0^{k+1} \\ 0 \\ U_4^{k+1} \end{pmatrix}}_{\text{given from b.c.}}$$

$$\begin{aligned} A\vec{U}^{k+1} &= \vec{U}^k + \vec{b} \\ \Rightarrow \vec{U}^{k+1} &= A^{-1}[\vec{U}^k + \vec{b}] \end{aligned}$$

3.1.3 Mixed boundary conditions

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

$U(0,t) = 1 = U_0^k, \quad U_x(1,t) = 2 = \frac{\partial U_{n+1}^k}{\partial x}$

Using a leap-frog approximation for the spatial derivative:

$$\frac{\partial U_j^k}{\partial x} = \frac{U_{j+1}^k - U_{j-1}^k}{2\Delta x} + \mathrm{O}(\Delta x^2)$$

This is more accurate than the forward approximation we used previously (see section 2.2.1):

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

So for the *Neumann* boundary condition we have:

$$U_x(1,t) = \frac{\partial U_{n+1}^k}{\partial x} \quad \approx \quad \frac{U_{n+2}^k - U_n^k}{2\Delta x} = 2 \tag{3.2}$$

With $n = 4$, U_5^k is called a 'ghost point' because it lies outside the bar. So using equation 3.2 we define U_5^k:

$$\Rightarrow U_5^k = 4\Delta x + U_3^k$$

Because we have Neumann boundary conditions at $x = a(= 1)$, $U_{n+1}^k = U_4^k$ is unknown, and given by equation 3.1:

$$\begin{aligned} U_4^k &= -s[U_5^{k+1} + U_3^{k+1}] + (1+2s)U_4^{k+1} \\ &\text{use } U_5^{k+1} = 4\Delta x + U_3^{k+1} \\ \Rightarrow U_4^k &= -s[4\Delta x + 2U_3^{k+1}] + (1+2s)U_4^{k+1} \end{aligned}$$

Our system of equations becomes:

$$\underbrace{\begin{pmatrix} 1+2s & -s & 0 & 0 \\ -s & 1+2s & -s & 0 \\ 0 & -s & 1+2s & -s \\ 0 & 0 & \underbrace{-2s} & 1+2s \\ & & \text{Neumann b.c.} & \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} U_1^{k+1} \\ U_2^{k+1} \\ U_3^{k+1} \\ U_4^{k+1} \end{pmatrix}}_{\vec{U}^{k+1}} = \underbrace{\begin{pmatrix} U_1^k \\ U_2^k \\ U_3^k \\ U_4^k \end{pmatrix}}_{\vec{U}^k}$$

$$+ \underbrace{\begin{pmatrix} \text{Dirichlet b.c.} \\ \overbrace{sU_0^{k+1}} \\ 0 \\ 0 \\ \underbrace{4s\Delta x} \\ \text{Neumann b.c.} \end{pmatrix}}_{\vec{b}}$$

$$\begin{aligned} A\vec{U}^{k+1} &= \vec{U}^k + \vec{b} = \vec{c} \\ \Rightarrow \vec{U}^{k+1} &= A^{-1}\vec{c} \end{aligned}$$

Using an implicit solver means we have to invert the matrix A to solve for $\vec{U}^{k+1}$ which is a lot more computationally expensive than the matrix multiply operation in equation 2.7 to find $\vec{U}^{k+1}$ for explicit solvers in section 2.2.3. This method is stable for $s \geq 0$ so larger time steps can be used for implicit methods than explicit methods.

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

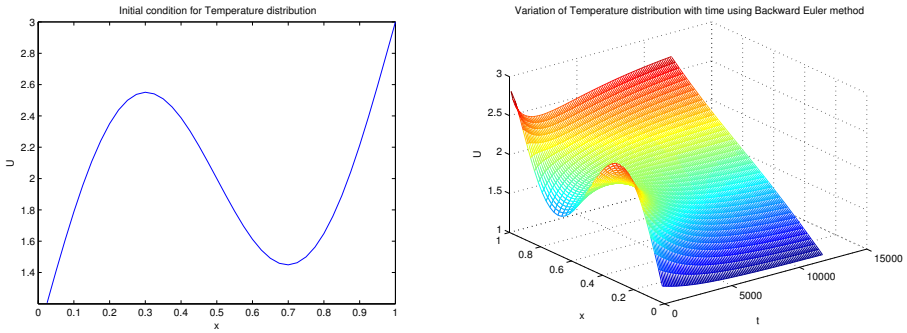


Figure 3.1: Initial conditions in (a) and matlab solution using Backward Euler method for temperature distribution along rod with time in (b)

Figure 3.1 shows the initial conditions in (a) and matlab solution for temperature distribution along rod with time in (b). The solution using the Backward Euler method in figure 3.1 is stable even for large time steps and this matlab code uses a time step $6\times$ greater than the solution using the Forward Euler method in figure 2.1. So even though there is more work in each time step (inverting a matrix) using the implicit Backward Euler method it allows larger time steps than the explicit Forward Euler method.

3.2 Crank-Nicolson Scheme

- Average of the explicit (forward Euler) and implicit (backward Euler) schemes.

- Uses:

$$U_t = \frac{U_j^{k+1} - U_j^k}{\Delta t}$$

$$U_{xx} = \frac{1}{2}\left[\underbrace{\frac{U_{j+1}^{k+1} - 2U_j^{k+1} + U_j^{k+1}}{\Delta x^2}}_{\text{implicit}} + \underbrace{\frac{U_{j+1}^{k} - 2U_j^{k} + U_{j-1}^{k}}{\Delta x^2}}_{\text{explicit}}\right]$$

- Often used for simple diffusion problems.

Chapter 4

Iterative methods

Implicit methods are stable - however they can take much longer to compute than explicit methods. We saw that the Backward Euler method requires a system of linear equations to be solved at each time step:

$$A\vec{U}^{k+1} = \vec{c} \quad (\text{where:} \quad \vec{c} = \vec{U}^k + \vec{b})$$

where A is a tridiagonal matrix. How can we speed up this calculation? By using *iterative methods*.

Iterative methods:

- improve the solution of $A\vec{x} = \vec{b}$.
- use a direct method or a 'guess' for an initial estimate of the solution.
- useful for solving large, sparse systems (eg. tridiagonal matrix A in Backward Euler scheme).
- many different methods such as Jacobi, Gauss-Seidel, relaxation methods.
- iterative methods are not always applicable and convergence criteria need to be met before they can be applied. However they are ideal for finite difference methods (involving solution of large sparse matrices).

Iterative methods begin with an initial guess for the solution $\vec{x}^0$ to the matrix equation we are trying to solve: $A\vec{x} = \vec{b}$. Each iteration updates the new k^{th} estimate ($\vec{x}^k$) which converge on the exact solution $\vec{x}$. Different methods have different convergence times and for big inverse matrix problems are much faster than direct matrix inverse methods.

4.1 Jacobi method

A is decomposed into a sum of lower-triangular (L), diagonal (D) and upper-triangular terms (U):

$$A = L + D + U$$

$$A = \begin{pmatrix} & & U \\ & D & \\ L & & \end{pmatrix}$$

4.1.1 Example

A for Backward Euler Method with Dirichlet boundary conditions:

$$\mathbf{A} = \begin{pmatrix} 1+2s & -s & & & 0 \\ -s & 1+2s & -s & & \\ & \ddots & \ddots & \ddots & \\ & & -s & 1+2s & -s \\ 0 & & & -s & 1+2s \end{pmatrix}$$

$$\Rightarrow \mathbf{D} = \begin{pmatrix} 1+2s & & & 0 \\ & 1+2s & & \\ & & \ddots & \\ 0 & & & 1+2s \end{pmatrix}, \mathbf{L} = \begin{pmatrix} 0 & & 0 \\ -s & \ddots & \\ & \ddots & \ddots \\ 0 & -s & 0 \end{pmatrix},$$

$$\mathbf{U} = \begin{pmatrix} 0 & -s & 0 \\ \ddots & \ddots & \\ & \ddots & -s \\ 0 & & 0 \end{pmatrix}$$

We want to solve $A\vec{x} = b$

$$\Rightarrow (D + L + U)\vec{x} = \vec{b}$$

$$\text{or} \qquad D\vec{x} = b - (L+U)\vec{x}$$

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

If $\vec{x}^k$ is k^{th} estimate of solution $A\vec{x} = \vec{b}$ then the $(k+1)^{th}$ estimate is:

$$D\vec{x}^{k+1} = b - (L+U)\vec{x}^k \qquad \text{(Jacobi Method)}$$

Since D is diagonal ($D_{ij} = \delta_{ij}A_{ij}$), we can write the vector equation above for $\vec{x}^{k+1}$ for each component $(x_1^{k+1}, ...x_n^{k+1})$.

$$x_i^{k+1} = \underbrace{\frac{1}{A_{ii}}}_{D^{-1}} \left(b_i - \underbrace{\sum_{j \neq i} A_{ij} x_j^k}_{\text{L+U part}} \right), \qquad 1 \leq i \leq n \tag{4.1}$$

4.1.2 Applying the Jacobi method

To start the scheme use an initial guess $\vec{x}^0$, (eg. $\vec{x}^0 = \vec{0}$). The iterations are repeated until $A\vec{x}^k \approx \vec{b}$ or the residual:

$$|\vec{b} - A\vec{x}^k| \quad < \quad \text{error tolerance} \quad (\text{eg.}10^{-5})$$

Jacobi method *converges* to correct solution $x^k \to x$ as $k \to \infty$ *if*:

$$\|D^{-1}(L+U)\| < 1 \Rightarrow \qquad \underbrace{|A_{ii}| > \sum_{j \neq i} |A_{ij}|}_{\text{A is } \textit{strictly diagonally dominant}} \quad , \quad 1 \leq i \leq n$$

where $\|B\|$ is the row-sum norm defined below:

$$\|B\| \quad = \quad \max_{1 \leq i \leq n} \sum_{j=1}^{n} |B_{ij}| \quad = \quad \max_{1 \leq i \leq n} \sum_{j=1, j \neq i}^{n} \frac{|a_{ij}|}{|a_{ii}|} \quad < 1.$$

The degree to which the convergence criteria:

$$|A_{ii}| > \sum_{j \neq i} |A_{ij}|, 1 \leq i \leq n$$

holds is a measure of how fast the estimate $\vec{x}^k$ converges to actual solution $\vec{x}$.

Look for matlab code on this free source website: `http://www.netlib.org/` which implements the Jacobi method and try for yourself.

4.2 Gauss-Seidel Method

- improves convergence of Jacobi method by simple modification
- in Jacobi method the new estimate, x_i^{k+1} is computed using only the current estimate, x_j^k
- Gauss-Seidel method uses all the possible new estimates ($j \leq i-1$) $x_{i-1}^{k+1}, x_{i-2}^{k+1}, ..., x_1^{k+1}, x_0^{k+1}$ when updating the new estimate x_i^{k+1}:

$$x_i^{k+1} = \frac{1}{A_{ii}} \left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} A_{ij} x_j^k \right), \qquad 1 \leq i \leq n.$$

  This is an improvement over the Jacobi method because it uses the new estimate x_j^{k+1} when it can. In vector form: $\vec{x}^{k+1} = D^{-1}(b - L\vec{x}^{k+1} - U\vec{x}^k)$ or $(D+L)\vec{x}^{k+1} = b - Ux^k$.

- solution converges $x^k \to x$ as $k \to \infty$ if: $\|(D+L)^{-1}U\| \leq 1$.

4.2.1 Example: using Gauss-Seidel method to solve a matrix equation

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

Solve $A\vec{x} = \vec{b}$ for $Res = |b - Ax^{k+1}| < 1e^{-3}$ with:

$$A = \begin{pmatrix} 5 & 0 & -2 \\ 3 & 5 & 1 \\ 0 & -3 & 4 \end{pmatrix}, \quad b = \begin{pmatrix} 7 \\ 2 \\ -4 \end{pmatrix}, \quad x^0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

with initial guess $\vec{x}^0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \Rightarrow$ takes 9 iterations. The Jacobi method needs 17 iterations to converge so takes nearly twice as long as the Gauss-Seidel method.

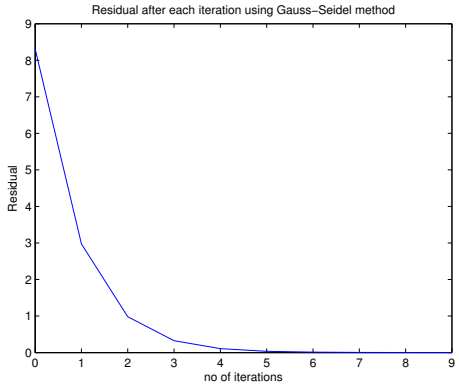


Figure 4.1: Plot of residual using the Gauss-Seidel method after each iteration

Figure 4.1 shows the residual using the Gauss-Seidel method after each iteration, it takes 9 iterations for the residual error to be less than 0.001.

4.3 Relaxation Methods

Relaxation methods generalise Gauss-Seidel method by introducing a relaxation factor, $\alpha > 0$. If α is optimised for the system this can increase the rate of convergence of the solution x^k by modifying the size of the correction:

$$x_i^{k+1} = x_i^k + \frac{\alpha}{A_{ii}} \left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} - \sum_{j=i}^{n} A_{ij} x_j^k \right), \qquad 1 \leq i \leq n. \tag{4.2}$$

This is called the successive relaxation (SR) method and for:

- $0 < \alpha < 1 \Rightarrow$ under-relaxation
- $\alpha = 1 \Rightarrow$ Gauss-Seidel method
- $\alpha > 1 \Rightarrow$ over-relaxation

We can re-write equation 4.2:

$$x_i^{k+1} = (1-\alpha)x_i^k + \frac{\alpha}{A_{ii}}\left[b_i - \sum_{j=1}^{i-1} A_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} A_{ij}x_j^k\right], \qquad 1 \leq i \leq n$$

Solution converges, ie $x^k \to x$ as $k \to \infty$ if: $\|(D+\alpha L)^{-1}[(1-\alpha)D - \alpha U]\| < 1$.

Chapter 5

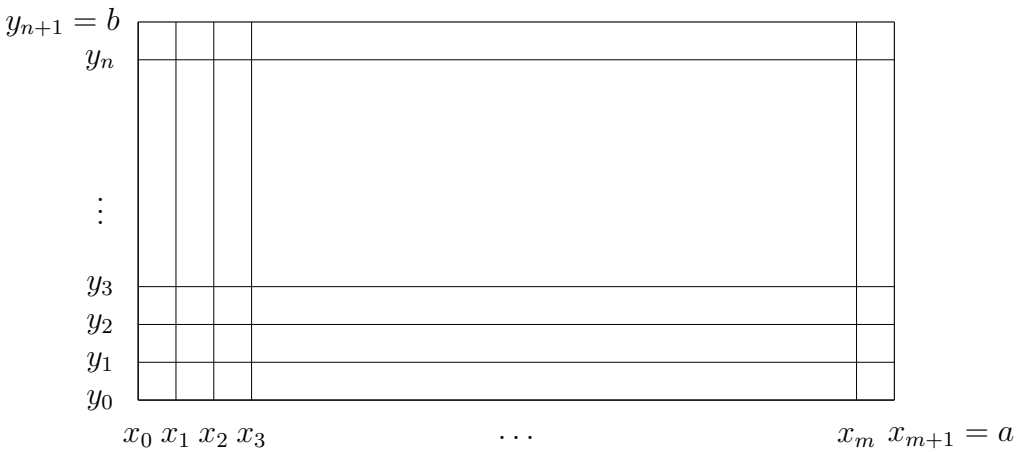
2-D Finite Difference

5.1 2-D Poisson's equation

Solving Laplace's ($f = 0$) or Poisson's equation in 2-D:

$$U_{xx} + U_{yy} = f \tag{5.1}$$

We discretise in x and y-directions:



We discretise in x-direction with grid spacing: $\Delta x = a/(m+1)$ and in y-direction with grid spacing: $\Delta y = b/(n+1)$, and let $x_k = k\Delta x$ where $0 \leq k \leq m+1$ and $y_j = j\Delta y$ where $0 \leq j \leq n+1$. We let $U_{kj} = U(x_k, y_j)$ and $f_{kj} = f(x_k, y_j)$ We are solving equation (5.1) using Dirichlet boundary

conditions:

$$\begin{aligned} U(0,y) &= U_{0,j} = g_{0,j}(y_j) \\ U(a,y) &= U_{m+1,j} = g_{m+1,j}(y_j) \\ U(x,0) &= U_{k,0} = g_{k,0}(x_k) \\ U(x,b) &= U_{k,n+1} = g_{k,n+1}(x_k) \end{aligned}$$

Using central difference approximations for U_{xx} and U_{yy} then the finite difference approximations for equation (5.1) are given by:

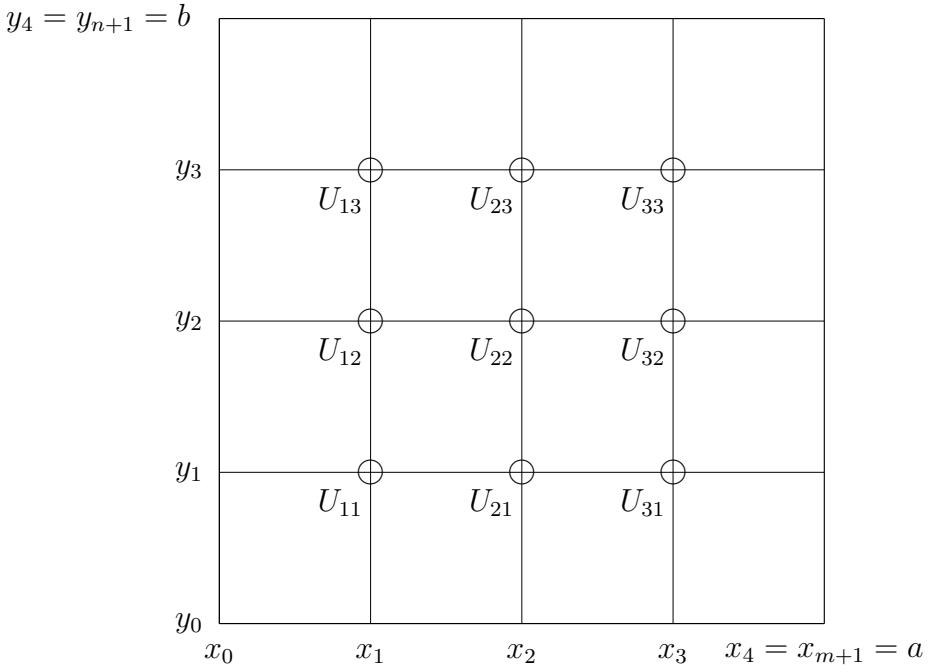
$$\begin{aligned} \frac{\partial^2 U}{\partial x^2} = U_{xx}(x_k, y_j) &= \frac{U_{k+1,j} - 2U_{k,j} + U_{k-1,j}}{\Delta x^2}, \\ \frac{\partial^2 U}{\partial y^2} = U_{yy}(x_k, y_j) &= \frac{U_{k,j+1} - 2U_{k,j} + U_{k,j-1}}{\Delta y^2}. \end{aligned}$$

Our discretised PDE (equation 5.1) becomes (if $\Delta x = \Delta y = h$):

$$\begin{aligned} \frac{U_{k+1,j} - 2U_{k,j} + U_{k-1,j}}{h^2} + \frac{U_{k,j+1} - 2U_{k,j} + U_{k,j-1}}{h^2} &= f_{k,j}, \\ \text{or } U_{k+1,j} + U_{k-1,j} - 4U_{k,j} + U_{k,j+1} + U_{k,j-1} &= h^2 f_{k,j}. \end{aligned} \quad (5.2)$$

Since we have Dirichlet boundary conditions: the outer boundaries of the region we are solving for are known: $U_{0,j}, U_{m+1,j}, U_{k,0}, U_{k,n+1}$, and we need to find the interior values: $U_{k,j}$ for $1 \leq k \leq m$ and $1 \leq j \leq n$.

For example: $m = 3$ and $n = 3$:



Thus we need to solve for the interior values marked with a circle above as the boundary values are already given. We let the vector of interior values we are solving for be defined as:

$$\vec{U} = \begin{pmatrix} U_{11} \\ U_{12} \\ U_{13} \\ U_{21} \\ U_{22} \\ U_{23} \\ U_{31} \\ U_{32} \\ U_{33} \end{pmatrix}, \text{ vector of interior values we are solving for.}$$

Thus the matrix system for $\vec{U}$ using equation (5.2):

$U_{k+1,j} + U_{k-1,j} - 4U_{k,j} + U_{k,j+1} + U_{k,j-1} = h^2 f_{k,j}$ becomes:

$$
\begin{array}{l}
k=1 \\ j=1 \\ k=1 \\ j=2 \\ k=1 \\ j=3 \\ k=1 \\ j=1 \\ k=1 \\ j=2 \\ k=2 \\ j=3 \\ k=2 \\ j=1 \\ k=3 \\ j=2 \\ k=3 \\ j=3
\end{array}
\begin{pmatrix}
-4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
U_{k,j} & U_{k,j+1} & & U_{k+1,j} & & & & & \\
1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
U_{k,j-1} & U_{k,j} & U_{k,j+1} & & U_{k+1,j} & & & & \\
0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\
 & U_{k,j-1} & U_{k,j} & & & U_{k+1,j} & & & \\
1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\
U_{k-1,j} & & & U_{k,j} & U_{k,j+1} & & U_{k+1,j} & & \\
0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\
 & U_{k-1,j} & & U_{k,j-1} & U_{k,j} & U_{k,j+1} & & U_{k+1,j} & \\
0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\
 & & U_{k-1,j} & & U_{k,j-1} & U_{k,j} & & & U_{k+1,j} \\
0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\
 & & & U_{k-1,j} & & & U_{k,j} & U_{k,j+1} & \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\
 & & & & U_{k-1,j} & & U_{k,j-1} & U_{k,j} & U_{k,j+1} \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \\
 & & & & & U_{k-1,j} & & U_{k,j-1} & U_{k,j}
\end{pmatrix}
\begin{pmatrix} U_{11} \\ U_{12} \\ U_{13} \\ U_{21} \\ U_{22} \\ U_{23} \\ U_{31} \\ U_{32} \\ U_{33} \end{pmatrix}
$$

$$
+ \begin{pmatrix}
U_{10} + U_{01} \\
U_{k,j-1} + U_{k-1,j} \\
U_{02} \\
U_{k-1,j} \\
U_{14} + U_{03} \\
U_{k,j+1} + U_{k-1,j} \\
U_{20} \\
U_{k,j-1} \\
0 \\
\\
U_{24} \\
U_{k,j+1} \\
U_{30} + U_{41} \\
U_{k,j-1} + U_{k+1,j} \\
U_{42} \\
U_{k+1,j} \\
U_{34} + U_{43} \\
U_{k,j+1} + U_{k+1,j}
\end{pmatrix}
= h^2 \begin{pmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{pmatrix}
$$

ie:

$$\begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix} \begin{pmatrix} U_{11} \\ U_{12} \\ U_{13} \\ U_{21} \\ U_{22} \\ U_{23} \\ U_{31} \\ U_{32} \\ U_{33} \end{pmatrix}$$

$$= \begin{pmatrix} h^2 f_{11} - g_{10} - g_{01} \\ h^2 f_{12} - g_{02} \\ h^2 f_{13} - g_{14} - g_{03} \\ h^2 f_{21} - g_{20} \\ h^2 f_{22} \\ h^2 f_{23} - g_{24} \\ h^2 f_{31} - g_{30} - g_{41} \\ h^2 f_{32} - g_{42} \\ h^2 f_{33} - g_{34} - g_{43} \end{pmatrix},$$

$$\text{or} \qquad A\vec{U} = \vec{f}. \tag{5.3}$$

If $m \times n \leq 100$ then direct matrix elimination methods can be used. Otherwise iterative methods such as Jacobi, Gauss-Seidel, relaxation methods should be used to solve for $\vec{U}$, as discussed in previous chapter 4. Because A is sparse and diagonally dominant, iterative solutions are ideal here.

However we see in the next section 5.2.1 that when we solve 2-D parabolic equations (2-D heat or diffusion equations) that the numerical solution requires 'tweaking' since the matrix A is no longer tridiagonal for 2-D as it was for 1-D.

5.2 2-D Heat (or Diffusion) Equation

We consider the 2-D heat equation:

$$U_t = \beta(U_{xx} + U_{yy}) \quad \text{for} \quad 0 \leq x \leq a, \quad 0 \leq y \leq b \quad \text{and} \quad 0 \leq t \leq T.$$

with initial conditions: $U(0, x, y) = f(x, y)$ and boundary conditions: $U(t, x, y) = g(t, x, y)$ for (x, y) on boundary.

- If we use explicit forward Euler scheme as we did for the 1-D heat equation the stability criteria is even stricter than for 1-D:

$$\Delta t \leq \frac{\Delta x^2 + \Delta y^2}{8\beta}$$

  $\rightarrow$ this is less attractive because the time step is much smaller.

- if we use backward Euler or the Crank-Nicolson methods they are no longer as attractive because the matrix systems to be solved are much larger and *no longer tridiagonal*.

- We will look at an alternative finite difference method specifically tailored to the 2-D heat equation: the alternating direction implicit (ADI) method.

5.2.1 Alternating Direct/Implicit method for the 2-D heat equation

$$U_t = \beta(U_{xx} + U_{yy})$$

We let $\Delta t = T/m,\ \Delta x = a/(n+1),\ \Delta y = b/(p+1)\ t_k = k\Delta t,\ 0 \leq k \leq m,\ x_i = i\Delta x, 0 \leq i \leq n+1,\ y_j = j\Delta y,\ 0 \leq j \leq p+1$, and:

$$U(t_k, x_i, y_j) = U_{ij}^k$$

The time derivative, U_t, is approximated using a leap-frog step in time about the mid-point $t_{k+1/2}$, where $t_{k+1/2} = (t_k + t_{k+1})/2$ using a time step of $\Delta t/2$:

$$\frac{\partial U_{ij}^{k+1/2}}{\partial t} = \frac{U_{ij}^{k+1} - U_{ij}^{k-1}}{\Delta t} \qquad \text{at time } t_{k+1/2}.$$

The spatial derivatives, U_{xx} and U_{yy}, are approximated using central differences:

$$\frac{\partial^2 U_{ij}^k}{\partial x^2} = \underbrace{\frac{U_{i+1,j}^k - 2U_{ij}^k + U_{i-1,j}^k}{\Delta x^2}}_{\text{EARLY STEP}} \qquad \text{at time } t_k,$$

$$\frac{\partial^2 U_{ij}^{k+1}}{\partial y^2} = \frac{U_{i,j+1}^{k+1} - 2U_{ij}^{k+1} + U_{i,j-1}^{k+1}}{\Delta y^2} \qquad \text{at time } t_{k+1}.$$

This introduces an 'early' bias which evaluates $U_{xx}(t_k)$ at an earlier time than $U_{yy}(t_{k+1})$. This bias is compensated by also evaluating $U_{xx}(t_{k+2})$ at t_{k+2}; $U_{yy}(t_{k+1})$ again at t_{k+1} and $U_t(t_{k+3/2})$ at mid-point between t_{k+1} and t_{k+2}:

$$\begin{aligned} \frac{\partial U_{ij}^{k+3/2}}{\partial t} &= \frac{U_{ij}^{k+2} - U_{ij}^{k+1}}{\Delta t}, \\ \frac{\partial^2 U_{ij}^{k+2}}{\partial x^2} &= \underbrace{\frac{U_{i+1,j}^{k+2} - 2U_{ij}^{k+2} + U_{i-1,j}^{k+2}}{\Delta x^2}}_{\text{LATE STEP}} \\ \text{and} \quad & \frac{\partial^2 U_{ij}^{k+1}}{\partial y^2} \text{ as before.} \end{aligned}$$

The boundary conditions specify U_{oj}^k, $U_{m+1,j}^k$, U_{io}^k, $U_{i,p+1}^k$, and initial conditions specify U_{ij}^0. So we are solving for $1 \le k \le m$, $1 \le i \le m$, $1 \le j \le p$. ie. for each interior point at time t_k.
We solve the problem for both time steps t_{k+1} and t_{k+2} at the same time using early and late definitions. If we let $s_x = \beta\Delta t/\Delta x^2$, $s_y = \beta\Delta t/\Delta y^2$ then $U_t = \beta(U_{xx} + U_{yy})$ becomes:
Early step:

$$U_{ij}^{k+1}(1 + 2s_y) - s_y[U_{i,j+1}^{k+1} + U_{i,j-1}^{k+1}] = U_{ij}^k(1 - 2s_x) + s_x[U_{i+1,j}^k + U_{i-1,j}^k]$$

This is solved first for i^{th} row of U^{k+1} matrix, for $1 \le i \le n$.

Late step:

$$U_{ij}^{k+2}(1 + 2s_x) - s_x[U_{i+1,j}^{k+2} + U_{i-1,j}^{k+2}] = U_{ij}^{k+1}(1 - 2s_y) + s_y[U_{i,j+1}^{k+1} + U_{i,j-1}^{k+1}]$$

This is solved for j^{th} column of U^{k+2} matrix, for $1 \le j \le p$.
These are solved using LU decomposition. (For more details see Schilling and Harris, p. 445).

5.3 Cylindrical and spherical polar co-ordinates

- Spherical and cylindrical symmetry in problems are often exploited to reduce 2-D $\rightarrow$ 1-D or 3-D $\rightarrow$ 1-D.

3-D Cylindrical Co-ordinates

$x = r\cos\theta$
$y = r\sin\theta$

$z = z$
$\Rightarrow r = \sqrt{x^2 + y^2}$
$\theta = \arctan(\frac{y}{x})$

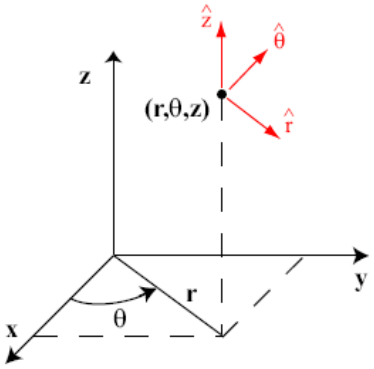


Figure 5.1: 3-D Cylindrical Co-ordinates

3-D Spherical Polar Co-ordinates

$x = r \sin\theta \cos\phi$
$y = r \sin\theta \sin\phi$
$z = r \cos\theta$

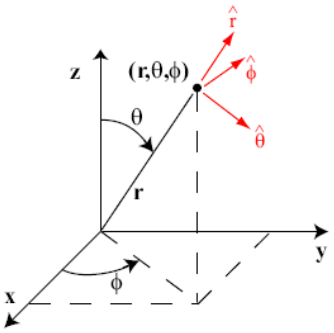


Figure 5.2: 3-D Spherical Polar Co-ordinates

2-D Polar Co-ordinates

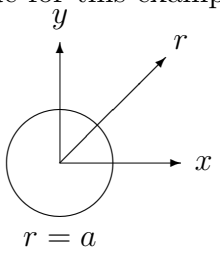
$x = r \cos\phi$
$y = r \sin\phi$

$\Rightarrow r = \sqrt{x^2 + y^2}$
$\phi = \arctan(\frac{y}{x})$.

$$\frac{\partial}{\partial x} = \frac{\partial r}{\partial x}\frac{\partial}{\partial r} + \frac{\partial \phi}{\partial x}\frac{\partial}{\partial \phi} = \cos\phi \frac{\partial}{\partial r} - \frac{\sin\phi}{r}\frac{\partial}{\partial \phi}$$
$$\frac{\partial}{\partial y} = \frac{\partial r}{\partial y}\frac{\partial}{\partial r} + \frac{\partial \phi}{\partial y}\frac{\partial}{\partial \phi} = \sin\phi \frac{\partial}{\partial r} + \frac{\cos\phi}{r}\frac{\partial}{\partial \phi}$$

5.3.1 Example: Temperature around a nuclear waste rod

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



Nuclear rod buried in ground

We consider the temperature increase due to storage of nuclear rods which release heat due to radioactive decay:

$$\underbrace{\frac{1}{\kappa}\frac{\partial T}{\partial t}(r,t) - \bigtriangledown^2 T(r,t)}_{\text{2 -D heat equation}} = \underbrace{S(r,t)}_{\text{source term}}$$

where the source term due to the radioactive decay of rod is given by:

$$S(r,t) \quad = \quad \begin{cases} T_{rod} e^{-t/\tau_0}/a^2 & \text{for } r \leq a \\ 0 & \text{elsewhere.} \end{cases}$$

where $a = 25$cm, $\kappa = 2 \times 10^7 \text{cm}^2/\text{year}$, $T_{rod} = 1K$, $\tau_0 = 100$years, $r_c = 100$cm, $T_E = 300K$, $0 < r < 100$cm and $0 < t < 100$years. Initially $T(r, t = 0) = 300K$.

Because the problem has circular symmetry (ie. no ϕ dependence) $\Rightarrow$ 2-D problem in (x, y) reduced to 1-D problem in r. $\bigtriangledown^2 T = T_{xx} + T_{yy}$ is 2-D in Cartesian co-ordinates. However if we choose to use polar co-ordinates then the temperature, $T(r, t)$ is a function of r and t only because the rod circularly symmetric and there is no ϕ dependence. This reduces the original

2-D problem to 1-D!
How do we evaluate $\bigtriangledown^2 T$ in polar co-ordinates?

$$\begin{aligned} T_{xx} &= \frac{\partial^2 T}{\partial x^2} = (\cos\phi\frac{\partial}{\partial r} - \frac{\sin\phi}{r}\frac{\partial}{\partial\phi})(\cos\phi\frac{\partial T}{\partial r} - \frac{\sin\phi}{r}\frac{\partial T}{\partial\phi}) \\ &= \cos^2\phi T_{rr} - \frac{2\sin\phi\cos\phi}{r}T_{r\phi} + \frac{\sin^2\phi}{r}T_r + \frac{2\cos\phi\sin\phi}{r}T_\phi + \frac{\sin^2\phi}{r}T_{\phi\phi} \end{aligned}$$

$$\begin{aligned} T_{yy} &= (\sin\phi\frac{\partial}{\partial r} + \frac{\cos\phi}{r}\frac{\partial}{\partial\phi})(\sin\phi\frac{\partial T}{\partial r} + \frac{\cos\phi}{r}\frac{\partial T}{\partial\phi}) \\ &= \sin^2\phi T_{rr} + \frac{2\cos\phi\sin\theta}{r}T_{r\phi} - \frac{2\cos\phi\sin\phi}{r^2}T_\phi + \frac{\cos^2\phi}{r}T_r + \frac{\cos^2\phi}{r^2}T_{\phi\phi} \end{aligned}$$

$$\text{and } T_{xx} + T_{yy} = T_{rr} + \frac{1}{r}T_r + \frac{1}{r}^2 T_{\phi\phi}$$

using $\cos^2\phi + \sin^2\phi = 1$.

Since the temperature $T(r,t)$ has no ϕ dependence then $T_{\phi\phi} = 0$ and we are solving the 1-D heat equation in polar co-ordinates:

$$\frac{1}{K}\frac{\partial T}{\partial t} - \frac{\partial^2 T}{\partial r^2} - \frac{1}{r}\frac{\partial T}{\partial r} = S(r,t)$$

We know that in the steady state solution eventually the nuclear rod is no longer radioactive and stops releasing heat: $S(r,t) \to 0$ as $t \to \infty$, and further enough away from the rod the temperature equals the environment temperature, $T(r = r_c, t) = 300K$. So the solution should approach the environmental temperature $T(r,t) = 300K$ once rod has finished radioactive decaying.
We use finite differences to solve:

$$\frac{1}{K}\frac{\partial T}{\partial t} - \frac{\partial^2 T}{\partial r^2} - \frac{1}{r}\frac{\partial T}{\partial r} = S(r,t) \tag{5.4}$$

We observe that there is a singularity at $r = 0$ in the above equation where special care needs to be taken so that the numerical solution is stable.

Initial conditions $T(r,0) = 300K$.
Neumann boundary conditions at $r = 0$ (temperature cannot flow into $r = 0$ region)

$$\frac{\partial T}{\partial r}(r = 0, t) = 0$$

Dirichlet boundary conditions at $r = r_c$

$$T(r = r_c, t) = 300K$$

Again we discretise space and time: $\Delta r = r_c/(n+1),\ \Delta t = T_f/m,\ r_j = j\Delta r,\ 0 \le j \le n+1,\ t_k = k\Delta t,\ 0 \le k \le m,\ T(r_j, t_k) = T_j^k$, and $S(r_j, t_k) = S_j^k$.

Discrete Neumann boundary conditions at $r = 0$ become:

$$\frac{\partial T_j^k}{\partial t}(r = 0, t) = \frac{\partial T_0^k}{\partial t} = 0 \approx \frac{T_1^k - T_0^k}{\Delta t} \Rightarrow T_0^k \approx T_1^k$$

Discrete Dirichlet boundary conditions at $r = r_c$ become:

$$T_j^k(r = r_c, t) = T_{n+1}^k = 300$$

We will use the backward Euler method (*implicit*) to solve the PDE. This means evaluating the spatial derivatives in r at the future time step t_{k+1}:

$$\begin{aligned}
T_t(t_{k+1}, r_j) &= \frac{T_j^{k+1} - T_j^k}{\Delta t} \\
T_{rr}(t_{k+1}, r_j) &= \frac{T_{j+1}^{k+1} - 2T_j^{k+1} + T_{j-1}^{k+1}}{\Delta r^2} \text{ (centred difference at } t_{k+1}) \\
T_r(t_{k+1}, r_j) &= \frac{T_{j+1}^{k+1} - T_{j-1}^{k+1}}{2\Delta r} \text{ (leap-frog in space)}
\end{aligned}$$

Using $r_j = j\Delta r$ our discretised PDE 5.4 becomes:

$$\begin{aligned}
\frac{1}{\kappa} T_t - T_{rr} - \frac{1}{r} T_r &= S(r, t) \\
\underbrace{\frac{1}{\kappa\Delta t}[T_j^{k+1} - T_j^k]}_{T_t/\kappa} - \underbrace{[\frac{T_{j+1}^{k+1} - 2T_j^{k+1} + T_{j-1}^{k+1}}{\Delta r^2}]}_{T_{rr}} - \underbrace{\frac{1}{j\Delta r}[\frac{T_{j+1}^{k+1} - T_{j-1}^{k+1}}{2\Delta r}]}_{T_r/r} &= S_j^k
\end{aligned}$$

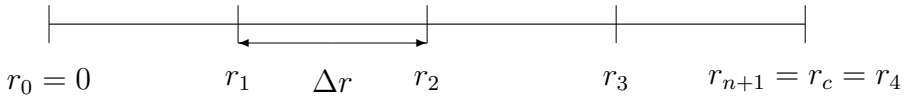
We let $s = \kappa\Delta t/\Delta r^2$ and we arrive at:

$$T_{j+1}^{k+1}[-s - \frac{s}{2j}] + T_{j-1}^{k+1}[-s + \frac{s}{2j}] + T_j^{k+1}[1 + 2s] = T_j^k + S_j^k \kappa\Delta t \qquad (5.5)$$

This is a tridiagonal matrix for $1 \le j \le n$.

Numerical solution of the 1-D heat equation in polar co-ordinates using the Backward Euler method

For $n = 3$:



The boundary conditions give $T_0^k \approx T_1^k$ using $\frac{\partial T}{\partial r}(r = 0, t) = 0)$ and $T_4^k = 300K$ using $(T(r = r_c, t) = 300)$ and the initial conditions are $T_j^0 = 300K$.

We solve equation 5.5 for T_1^k, T_2^k, T_3^k at each time step (t_k):

$$\begin{pmatrix} 1+2s & (-s-\frac{s}{2j}) & 0 \\ (-s+\frac{s}{2j}) & 1+2s & (-s-\frac{s}{2j}) \\ 0 & (-s+\frac{s}{2j}) & 1+2s \end{pmatrix} \begin{pmatrix} T_1^{k+1} \\ T_2^{k+1} \\ T_3^{k+1} \end{pmatrix} + \begin{pmatrix} (-s+\frac{s}{2j})T_0^{k+1} \\ 0 \\ (-s-\frac{s}{2j})T_4^{k+1} \end{pmatrix} = \begin{pmatrix} T_1^k \\ T_2^k \\ T_3^k \end{pmatrix} + \kappa\Delta t \begin{pmatrix} S_1^k \\ S_2^k \\ S_3^k \end{pmatrix}$$

Using the boundary conditions: $T_0^{k+1} \approx T_1^{k+1}, T_4^{k+1} = 300K$

$$\begin{pmatrix} (1+s+\frac{s}{2j}) & (-s-\frac{s}{2j}) & 0 \\ (-s+\frac{s}{2j}) & (1+2s) & (-s-\frac{s}{2j}) \\ 0 & (-s+\frac{s}{2j}) & (1+2s) \end{pmatrix} \begin{pmatrix} T_1^{k+1} \\ T_2^{k+1} \\ T_3^{k+1} \end{pmatrix} = \begin{pmatrix} T_1^k \\ T_2^k \\ T_3^k \end{pmatrix} + \kappa\Delta t \begin{pmatrix} S_1^k \\ S_2^k \\ S_3^k \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ (-s-\frac{s}{2j})T_4^{k+1} \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} (1+s+\frac{s}{2}) & (-s-\frac{s}{2}) & 0 \\ (-s+\frac{s}{4}) & (1+2s) & (-s-\frac{s}{4}) \\ 0 & (-s+\frac{s}{6}) & (1+2s) \end{pmatrix} \begin{pmatrix} T_1^{k+1} \\ T_2^{k+1} \\ T_3^{k+1} \end{pmatrix} = \begin{pmatrix} T_1^k \\ T_2^k \\ T_3^k \end{pmatrix} + \kappa\Delta t \begin{pmatrix} S_1^k \\ S_2^k \\ S_3^k \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ (-s-\frac{s}{6})300 \end{pmatrix}$$

Or to simplify we are solving the following matrix equation for the vector of unknown temperatures $\vec{T}^{k+1}$:

$$A\vec{T}^{k+1} = \vec{T}^k + \kappa\Delta t\vec{S}^k + \vec{b}$$

The matlab code **NuclearWaste.m** can be used to check that solution for $T \to 300K$ as $t \to \infty$ (steady state approaches environment temperature, 300K).

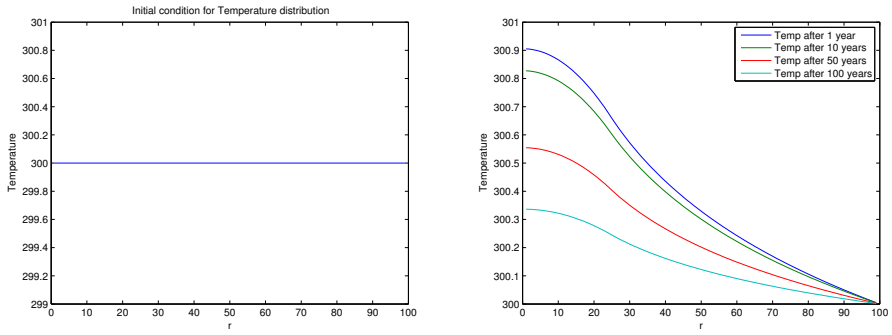


Figure 5.3: Initial conditions in (a) and matlab solution using Backward Euler method for temperature distribution near nuclear rod at different time intervals in (b)

Figure 5.3 shows the initial conditions and temperature distribution near the nuclear rod at different time intervals.

Part II

Numerical solution of hyperbolic equations

Chapter 6

Analytical solutions to the 1-D Wave equation

6.1 1-D Wave equation

$$\begin{aligned} U_{tt} - c^2 U_{xx} &= 0 \\ \text{or } (\frac{\partial}{\partial t} - c\frac{\partial}{\partial x})(\frac{\partial}{\partial t} + c\frac{\partial}{\partial x})U &= 0 \end{aligned} \tag{6.1}$$

This is a hyperbolic equation since $A = 1, C = -c^2, B = 0$ so that $AC < B^2$

6.2 d'Alembert's solution

We introduce a change of variables:
$\xi = x + ct$
$\eta = x - ct$
Then:

$$\begin{aligned} \frac{\partial}{\partial \xi} &= \frac{\partial x}{\partial \xi}\frac{\partial}{\partial x} + \frac{\partial t}{\partial \xi}\frac{\partial}{\partial t} = \frac{\partial}{\partial x} + \frac{1}{c}\frac{\partial}{\partial t} \\ \frac{\partial}{\partial \eta} &= \frac{\partial x}{\partial \eta}\frac{\partial}{\partial x} + \frac{\partial t}{\partial \eta}\frac{\partial}{\partial t} = \frac{\partial}{\partial x} - \frac{1}{c}\frac{\partial}{\partial t} \end{aligned}$$

So equation 6.1 becomes:

$$\begin{aligned} \Rightarrow -c\frac{\partial}{\partial \eta}(c\frac{\partial}{\partial \xi})U &= -c^2 U_{\xi\eta} = 0 \\ \Rightarrow U(\xi, \eta) &= g(\xi) + f(\eta) \\ &= g(x + ct) + f(x - ct) \end{aligned}$$

- $g(x+ct)$ defines waves that travel in left direction with speed c
- $f(x-ct)$ defines waves that travel in right direction with speed c.
- The pulses move without dispersion and the initial pulse breaks into a left and right pulse.

6.3 Separation of variables

This time we will derive the analytical solution using the separation of variables technique as we did for the 1-D heat equation in section 2.1. We want to solve the 1-D heat equation:

$$\begin{aligned} U_{tt} &= c^2 U_{xx} \qquad (6.2) \\ \text{with periodic boundary conditions } U(0,t) &= 0 = U(L,t) \end{aligned}$$

Again we assume $U(x,t) = X(x)T(t)$ then substitute into equation 6.2:

$$\begin{aligned} X(x)T''(t) &= c^2 X''(x)T(t) \\ &\text{then divide by } XT \quad \Rightarrow \\ \underbrace{\frac{T''}{T}} &= \underbrace{c^2\frac{X''}{X}} = -\omega^2 \text{ (constant)} \\ \text{function of } t \text{ only} &\qquad \text{function of } x \text{ only} \end{aligned}$$

Solving $X'' = -k^2 X$, where $k = \omega/c$ for $X(x)$ gives:

$$X = A\sin(kx) + B\cos(kx)$$

The boundary conditions give $X(0) = B = 0$ and $X(L) = A\sin kL = 0 \quad \Rightarrow$ $k = k_n = n\pi/L, \quad n = 0, 1, \ldots$. Thus the general solution for $X(x)$ is:

$$X(x) = \sum_n a_n \sin(\frac{n\pi x}{L})$$

Similarly if we solve $T'' = -\omega_n^2 T$ (where $\omega_n = ck_n$) we find the general solution:

$$T(t) = C\sin(\omega_n t) + D\cos(\omega_n t)$$

. So the solution for $U(x,t)$ is:

$$\begin{aligned} U(x,t) &= X(x)T(t) \\ &= \sum_n [a_n \sin(\omega_n t) + b_n \cos(\omega_n t)] \sin(k_n x) \end{aligned}$$

where a_n, b_n are given by initial conditions:

$$\begin{aligned} U(x,0) = U_0(x), \qquad & \frac{\partial U}{\partial t}(x,0) = V_0(x) \\ \Rightarrow U_0 = \sum_n b_n \sin(k_n x) \qquad & V_0 = \sum_n a_n \omega_n \sin(k_n x) \end{aligned}$$

using orthogonality of sine functions: $\int_0^L \sin(k_m x)\ \sin(k_n x) dx = \delta_{nm} \Rightarrow$

$$\begin{aligned} b_m &= \frac{2}{L} \int_0^L U_0(x) \sin(k_m x) dx \\ a_m &= \frac{2}{w_m L} \int_0^L V_0(x) \sin(k_m x) dx \end{aligned}$$

where $k_n = n\pi/L$ and $k_m = m\pi/L$.

Chapter 7

Flux conservative problems

7.1 Flux Conservative Equation

A large class of PDEs can be cast into the form of a flux conservative equation:

$$\frac{\partial \vec{U}}{\partial t} = \frac{-\partial f}{\partial x}(\vec{U}, \vec{U}_x, \vec{U}_{xx}, ...)$$

Example: flux conservative form for the wave equation

We consider the 1-D wave equation $U_{tt} = c^2 U_{xx}$. If we let:

$$\vec{w} = \begin{pmatrix} r \\ s \end{pmatrix}, \qquad \text{where } r = c\frac{\partial U}{\partial x}, \text{ and } s = \frac{\partial U}{\partial t}.$$

This means that:

$$\begin{aligned} \frac{\partial \vec{w}}{\partial t} &= \begin{pmatrix} \frac{\partial r}{\partial t} \\ \frac{\partial s}{\partial t} \end{pmatrix} = \begin{pmatrix} c\frac{\partial s}{\partial x} \\ c\frac{\partial r}{\partial x} \end{pmatrix} \\ \text{or } \frac{\partial \vec{w}}{\partial t} &= -\frac{\partial}{\partial x}\begin{pmatrix} 0 & -c \\ -c & 0 \end{pmatrix}\vec{w} = -\frac{\partial}{\partial x} f(\vec{w}) \end{aligned}$$

7.2 Stability analysis of numerical solutions of the first order flux conservative or 1-D advection equation

$$\frac{\partial U}{\partial t} = -c\frac{\partial U}{\partial x} \tag{7.1}$$

We introduce a change of variable $\xi = x - ct$ and:

$$\frac{\partial}{\partial t} = \frac{\partial \xi}{\partial t}\frac{\partial}{\partial \xi} = -c\frac{\partial}{\partial \xi}, \quad \frac{\partial}{\partial x} = \frac{\partial \xi}{\partial x}\frac{\partial}{\partial \xi} = \frac{\partial}{\partial \xi}$$

We see that equation 7.1 holds: $-c\frac{\partial U}{\partial \xi} = -c\frac{\partial U}{\partial \xi}$. So, $U(x,t) = U(\xi) = f(x-ct)$ is the analytic general solution of equation 7.1, which is a wave propagating in the right (positive x) direction.

We study the stability of different finite difference schemes in solving the flux conservative or 1-D advection equation:

$$U_t = -cU_x,\ x_0 \le x \le x_1,\ t_0 \le t \le T$$

Again we discretise problem $\Delta x = \frac{x_1 - x_0}{n+1}$, $\Delta t = \frac{T - t_0}{m}$ and let $x_j = x_0 + j\Delta x,\ j = 0, \ldots, n+1,\ t_k = t_0 + k\Delta t,\ k = 0, \ldots, m$, and $U_j^k = U(x_j, t_k)$.

7.3 Forward Time Centred Space (FTCS)

Forward Euler method in time:

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

Leap-frog or centred difference in space:

$$\frac{\partial U_j^k}{\partial x} = \frac{U_{j+1}^k - U_{j-1}^k}{2\Delta x} + \mathrm{O}(\Delta x^2)$$

Using FTCS method: $U_t = -cU_x$ gives:

$$U_j^{k+1} = U_j^k - \frac{c\Delta t}{2\Delta x}[U_{j+1}^k - U_{j-1}^k] \tag{7.2}$$

7.3.1 von Neumann stability analysis of FTCS method

FTCS is unstable! Why?
We assume that independent solutions (eigenmodes) of equation 7.2 (or any difference equation) are of the form:

$$U_j^k = \xi^k e^{ipj\Delta x} \tag{7.3}$$

where p is a real spatial wavenumber and $\xi = \xi(p)$ is a complex number that depends on p.
Equation 7.3 shows that the time dependence of a single eigenmode U_j^k is

only through successive powers of $\xi(\xi^k)$. $\Rightarrow$ Difference equations are *unstable* if $|\xi(p)| > 1$ for some p. ξ is called the amplification factor.
To find $\xi(p)$ for FTCS method substitute $U_j^k = \xi^k e^{ipj\Delta x}$ into equation 7.2:

$$\begin{aligned} \xi^{k+1} e^{ipj\Delta x} &= \xi^k e^{ipj\Delta x} \left(1 - \frac{c\Delta t}{2\Delta x} \underbrace{(e^{ip\Delta x} - e^{-ip\Delta x})}_{2i\sin(p\Delta x)} \right) \\ \Rightarrow \xi(p) &= 1 - \frac{ic\Delta t}{\Delta x} \sin(p\Delta x) \end{aligned}$$

and $|\xi(p)| \geq 1 \; \forall \; p \Rightarrow$ FTCS scheme is unconditionally *unstable* for solving $U_t = -cU_x$.

7.4 Lax Method

Again we are solving the flux conservative equation: $U_t = -cU_x$. The instability in the FTCS method is removed in the Lax method by using the average for $U_j^k = \frac{U_{j+1}^k + U_{j-1}^k}{2}$ instead of U_j^k in approximating U_t:

$$\frac{\partial U_j^k}{\partial t} = \frac{U_j^{k+1} - \frac{1}{2}[U_{j+1}^k + U_{j-1}^k]}{\Delta t}$$

and centred difference again for U_x. Then $U_t = -cU_x$ becomes:

$$U_j^{k+1} = \frac{1}{2}[U_{j+1}^k + U_{j-1}^k] - \frac{c\Delta t}{2\Delta x}[U_{j+1}^k - U_{j-1}^k] \tag{7.4}$$

7.4.1 von Neumann Stability Analysis of Lax Method

The Lax method is conditionally stable. To see substitute $U_j^k = \xi^k e^{ipj\Delta x}$ into equation 7.4:

$$\xi^{k+1} e^{ipj\Delta x} = \xi^k e^{ipj\Delta x} \left(\underbrace{\frac{1}{2}[e^{ip\Delta x} + e^{-ip\Delta x}]}_{\cos(p\Delta x)} - \frac{c\Delta t}{2\Delta x} \underbrace{(e^{ip\Delta x} - e^{-ip\Delta x})}_{2i\sin(p\Delta x)} \right)$$

$$\Rightarrow \xi = \cos(p\Delta x) - i\frac{c\Delta t}{\Delta x} \sin(p\Delta x)$$

Lax method stable when $|\xi|^2 \leq 1$:

$$\Rightarrow |\cos^2(p\Delta x) + \frac{c^2\Delta t^2}{\Delta x^2} \sin^2(p\Delta x)| \leq 1$$

$$\text{or } |1 - (1 - \frac{c^2\Delta t^2}{\Delta x^2})\sin^2(p\Delta x)| \leq 1$$

$$\Rightarrow 1 - \frac{c^2\Delta t^2}{\Delta x^2} \geq 0$$

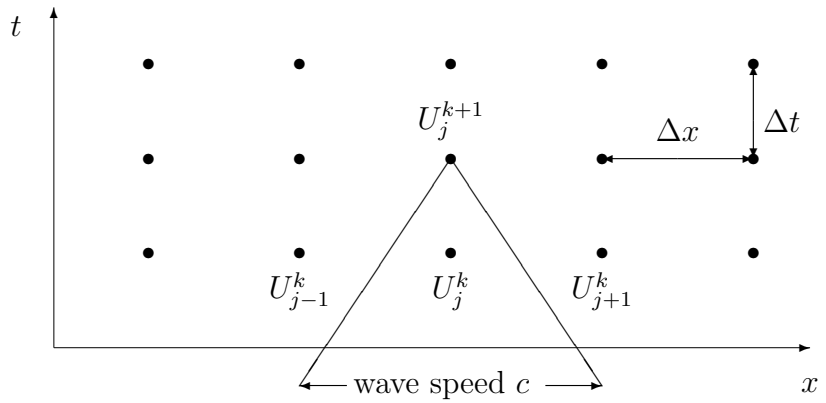
$$\text{or } \frac{c^2\Delta t^2}{\Delta x^2} \leq 1$$

$$\text{or } \underbrace{\Delta t \leq \frac{\Delta x}{c}} \quad (c > 0)$$

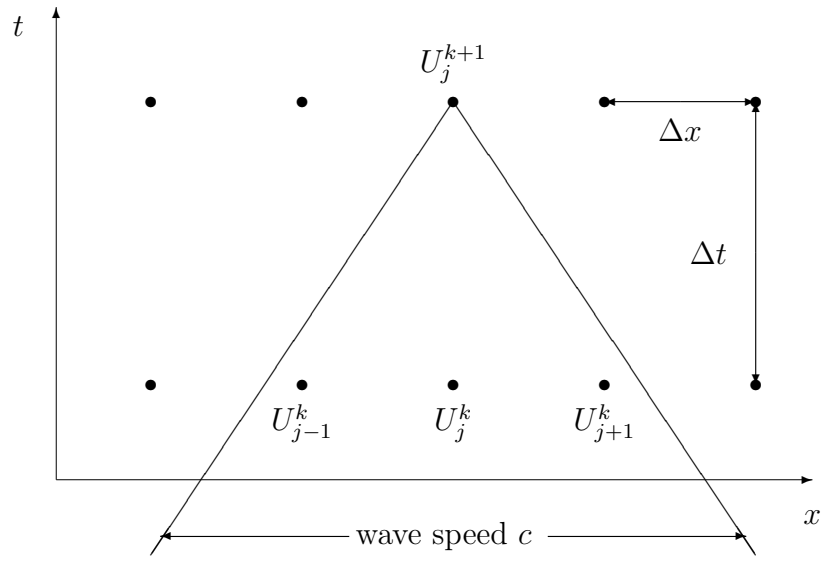
COURANT CONDITION

7.5 Courant Condition

- The Courant condition means Lax method is stable when $\Delta t \leq \Delta x/c$
- The physical meaning is that value U_j^{k+1} is computed from information at points $j-1$ and $j+1$ at time k in a stable scheme, where the wave speed is less that the mesh spacing divided by time. ie. in a continuum wave equation information propagates at maximum speed c, so Lax method is *stable* when $\frac{\Delta x}{\Delta t} \geq c$. This is shown in the plot below:



- Unstable schemes arise when $\frac{\Delta x}{\Delta t} \leq c$ ie. when the time step Δt becomes too large because U_j^{k+1} requires information from points outside $[U_{j-1}^k, U_{j+1}^k]$ as shown in the plot below. (see Press et al, Numerical Recipes, p. 825-830)



7.6 von Neumann Stability Analysis For Wave Equation

$$\begin{aligned} U_{tt} &= c^2 U_{xx} \\ \text{let} \quad \vec{w} &= \begin{pmatrix} r \\ s \end{pmatrix} = \begin{pmatrix} cU_x \\ U_t \end{pmatrix} \end{aligned}$$

We saw earlier that:

$$\frac{\partial \vec{w}}{\partial t} = \begin{pmatrix} \frac{\partial r}{\partial t} \\ \frac{\partial s}{\partial t} \end{pmatrix} = \begin{pmatrix} c\frac{\partial s}{\partial x} \\ c\frac{\partial r}{\partial x} \end{pmatrix} = c\frac{\partial}{\partial x}\begin{pmatrix} s \\ r \end{pmatrix}$$

7.6.1 Lax method

We will solve the wave equation using the Lax method:
$r_t = cs_x$ becomes:

$$r_j^{k+1} = \frac{1}{2}[r_{j-1}^k + r_{j+1}^k] + \frac{c\Delta t}{2\Delta x}(s_{j+1}^k - s_{j-1}^k) \tag{7.5}$$

$s_t = cr_x$ becomes:

$$s_j^{k+1} = \frac{1}{2}[s_{j-1}^k + s_{j+1}^k] + \frac{c\Delta t}{2\Delta x}(r_{j+1}^k - r_{j-1}^k) \tag{7.6}$$

where $r_j^k = r(x_j, t_k)$ and $s_j^k = s(x_j, t_k)$

For von Neumann stability analysis assume eigen-modes for r_j^k and s_j^k are of the form:

$$\begin{pmatrix} r_j^k \\ s_j^k \end{pmatrix} = \xi^k e^{ipj\Delta x} \begin{pmatrix} r_j^0 \\ s_j^0 \end{pmatrix} \Rightarrow \text{solutions stable if } |\xi| \leq 1$$

Equation 7.5 and 7.6 give:

$$\begin{pmatrix} \xi - \cos(p\Delta x) & -\frac{ic\Delta t}{\Delta x}\sin(p\Delta x) \\ -\frac{ic\Delta t}{\Delta x}\sin(p\Delta x) & \xi - \cos(p\Delta x) \end{pmatrix} \begin{pmatrix} r^0 \\ s^0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

- This has a solution only if determinant = 0.
- This gives $\xi = \cos(p\Delta x) \pm i\frac{c\Delta t}{\Delta x}\sin(p\Delta x)$.
- This is stable if $|\xi|^2 \leq 1$ which gives same *Courant condition* $\Delta t \leq \frac{\Delta x}{c}$.

7.7 Other sources of error

7.7.1 Phase Errors (through dispersion)

- Fourier analysis of the Lax method shows how phase errors arise.
- The Fourier mode $U(x,t) = e^{i(px+\omega t)}$ is an exact solution of $U_t = -cU_x$ if ω and p satisfy the dispersion relation $\omega = -cp$, then $U(x,t) = e^{ip(x-ct)} = f(x-ct)$ gives the exact solution of $U_t = -cU_x$.
- ie. this mode is completely undamped and the amplitude is constant (no dispersion) for the numerical solution using a time step which satisifes this dispersion relation.
- We will show the effects of phase errors by studying the numerical solution of the 1-D advection equation using different time steps which lead to dispersion being absent or present in section 7.7.2.

Dispersion relation for the Lax Method

The dispersion relation is only satisfied if: $\Delta t = \frac{\Delta x}{c}$.
Why? Consider $U_j^k = \xi^k e^{ipj\Delta x}$
In section 7.4.1 we found:

$$\begin{aligned} \xi &= \cos(p\Delta x) - i\frac{c\Delta t}{\Delta x}\sin(p\Delta x) \\ &= e^{-ip\Delta x} + i(1 - \frac{c\Delta t}{\Delta x})\sin(p\Delta x) \end{aligned}$$

If we let $\Delta t = \frac{\Delta x}{c} \Rightarrow \xi = e^{-ip\Delta x}$ and $U_j^k = \xi^k e^{ipj\Delta x} = e^{ip(-k\Delta x + j\Delta x)}$
When we substitute $x_j = j\Delta x$, $t_k = k\Delta t$ and the dipsersion relation $\Delta x = c\Delta t$ then: $U_j^k = e^{ip(-ck\Delta t + j\Delta x)} = e^{ip(x_j - ct_k)} = \underbrace{f(x_j - ct_k)}_{\text{exact solution}}$.

Thus the Lax method has no dispersion present when the time step satisfies the dispersion relation exactly: $\Delta t = \frac{\Delta x}{c}$. We will show this in the next section.

x

7.7.2 Dispersion in the numerical solution of the 1-D advection equation using the Lax method

The matlab code for this section is **Lax_Flux.m**.

Use Lax Method to solve:

$$U_t + U_x = 0, \quad 0 \le x \le 2 \approx \infty, \quad 0 \le t \le 1$$

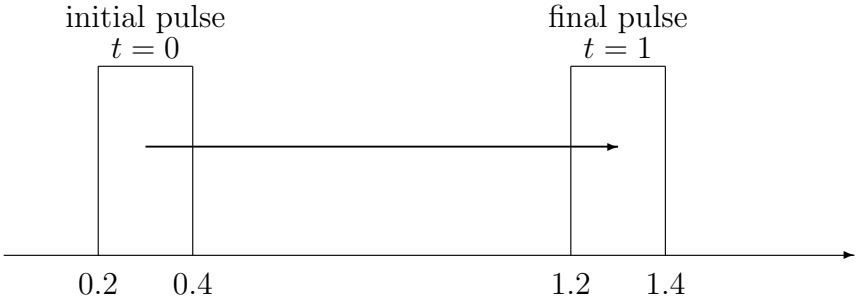
initial conditions:

$$U(x,0) = \begin{cases} 1, & 0.2 \le x \le 0.4 \\ 0, & \text{otherwise} \end{cases} = U^0(x)$$

boundary conditions:

$$U(0,t) = U(2,t) = 0$$

Exact solution



$$U(x,t) = U^0(x - ct)$$

$$= \quad U^0(x-t) \qquad (c=1)$$

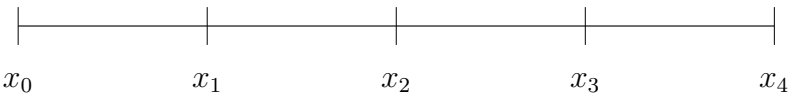
In the next section we compare the above exact solution with the numerical solution using the Lax method with different time steps:

- $\Delta t = \frac{\Delta x}{c} \Rightarrow$ *no dispersion* matches analytic solution.
- $\Delta t = \frac{\Delta x}{2c} \Rightarrow$ *dispersion* present but pulse matches speed of wave.
- $\Delta t = \frac{1.001\Delta x}{c} \Rightarrow$ courant condition not met $\rightarrow$ unstable!

Lax Method for $U_t + U_x = 0$

Equation 7.4 gives: $U_j^{k+1} = \frac{1}{2}[U_{j+1}^k + U_{j-1}^k] - \frac{c\Delta t}{2\Delta x}[U_{j+1}^k - U_{j-1}^k]$
let $s = \frac{c\Delta t}{\Delta x}$
$\Rightarrow U_j^{k+1} = \frac{1}{2}(1-s)U_{j+1}^k + U_{j-1}^k + \frac{1}{2}(1+s)U_{j-1}^k$

Again for simplicity we only consider 4 elements in x:



Solve for U_j^{k+1} for $0 \leq k \leq m$, $1 \leq j \leq 3$ with *boundary conditions*: $U_0^k = 0$, $U_4^k = 0$. We have:

$$\begin{pmatrix} U_1^{k+1} \\ U_2^{k+1} \\ U_3^{k+1} \end{pmatrix} = \begin{pmatrix} 0 & \frac{1}{2}(1-s) & 0 \\ \frac{1}{2}(1+s) & 0 & \frac{1}{2}(1-s) \\ 0 & \frac{1}{2}(1+s) & 0 \end{pmatrix} \begin{pmatrix} U_1^k \\ U_2^k \\ U_3^k \end{pmatrix} + \begin{pmatrix} \frac{1}{2}(1+s)U_0^k \\ 0 \\ \frac{1}{2}(1-s)U_4^k \end{pmatrix}$$

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

Dispersion means the initial pulse changes shape (unlike analytical solution) because wave components with different frequencies travel at different speeds. The matlab code is **Lax_Flux.m**.

The numerical solution changes for different time steps depending on whether or not the scheme is stable or dispersion is present. Figure 2.1 shows how the solution changes for different time steps depending on whether or not the scheme is stable or dispersion is present.

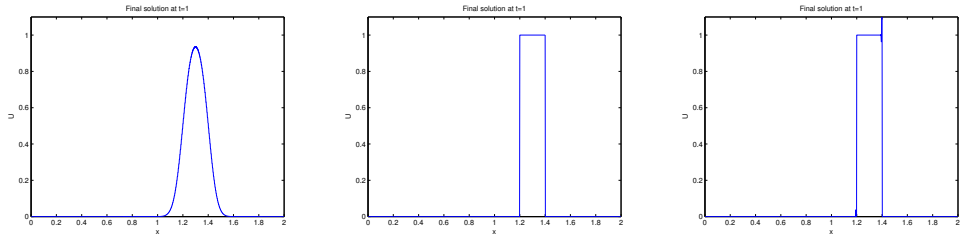


Figure 7.1: Solution at $t = 1$ using the Lax method with different time steps, (a) $\Delta t = \Delta x/2c$ where dispersion is present but the pulse matches the analytical solution for the speed of the wave, (b) $\Delta t = \Delta x/c$ where no dispersion is present and numerical solution matches analytical solution exactly, and (c) $\Delta t = 1.001\Delta x/c$ where the Courant condition is not met and solution is becoming unstable.

7.7.3 Error due to nonlinear terms

Example

Shock wave equation:

$$U_t + \underbrace{UU_x}_{\text{nonlinear term}} = 0$$

- nonlinear term causes wave profile to steepen resulting in a shock.
- schemes stable for linear problems can become unstable.
- this will be discussed later in chapter 11

7.7.4 Aliasing error

Example

Alising error occurs when a short wavelength (λ_1) is not represented well by the mesh-spacing (Δx), and may be misinterpreted as a longer wavelength oscillation (λ_2).

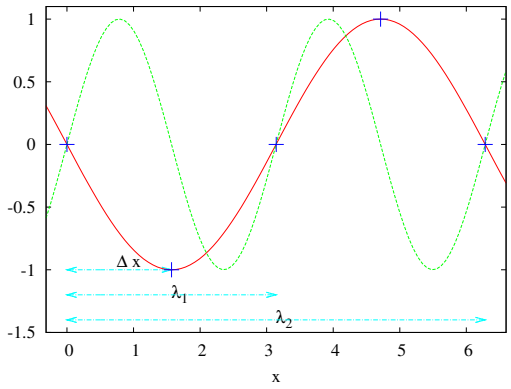


Figure 7.2: Aliasing error occurs when the mesh spacing Δx is too large to represent the smallest wavelength λ_1 and misinterprets it as a longer wavelength oscillation λ_2

Chapter 8

Numerical Solution of 1-D and 2-D Wave Equation

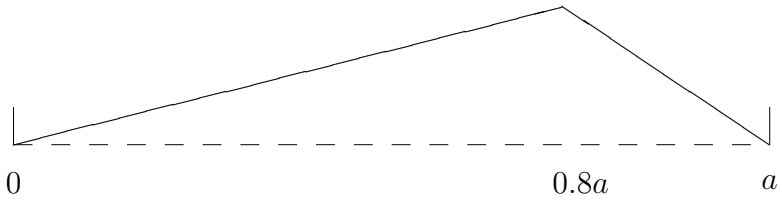
8.1 Explicit Central Difference for 1-D Wave Equation

$$U_{tt} = c^2 U_{xx}, \quad 0 \le t \le T,\ 0 \le x \le a$$

Discretise: $\Delta t = \frac{T}{m},\ \Delta x = \frac{a}{n+1}$,
$t_k = k\Delta t,\ 0 \le k \le m,\ x_j = j\Delta x$ and $0 \le j \le n+1$.

8.1.1 Example: plucking a string

The matlab code for this example is **Wave1D.m**.



A string is initially plucked or lifted from rest:
boundary conditions: $U(0,t) = 0,\ U(a,t) = 0$ or $U_0^k = 0,\ U_{n+1}^k = 0$
initial conditions: string is "plucked" or lifted 1mm at $x = 0.8a$:

$$U(x, t=0) = f(x) = \begin{cases} \frac{1.25x}{a}, & \text{for } x \le 0.8a \\ 5(1 - \frac{x}{a}), & \text{for } x > 0.8a \end{cases}$$

Plucked string is released from rest:

$$\frac{\partial U}{\partial t}(x,0) = g(x) = 0$$

$$U(x,t=0) = f(x) \Rightarrow U_j^0 = f_j = f(x_j)$$

$$\frac{\partial U}{\partial t}(x,t=0) = g(x) \Rightarrow \underbrace{\frac{\partial U_j^0}{\partial t} \approx \frac{U_j^1 - U_j^{-1}}{2\Delta t}}_{\text{leap-frog in time}} = g_j = g(x_j)$$

We can solve for 'ghost' point U_j^{-1}:

$$U_j^- 1 = U_j^1 - 2\Delta t g(x_j)$$

We approximate U_{tt} and U_{xx} using central differences:

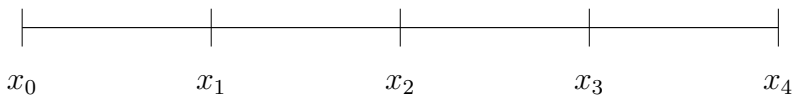
$$\begin{aligned} U_{tt} &= \frac{U_j^{k+1} - 2U_j^k + U_j^{k-1}}{\Delta t^2} \\ U_{xx} &= \frac{U_{j+1}^k - 2U_j^k + U_{j-1}^k}{\Delta x^2} \end{aligned}$$

Using $U_{tt} = c^2 U_{xx}$ and $s = \frac{c^2 \Delta t^2}{\Delta x^2}$, we solve for U_j^{k+1} at time step $k+1$:

$$U_j^{k+1} = \underbrace{-U_j^{k-1}}_{\text{solution at } t_{k-1}} + \underbrace{2U_j^k(1-s) + s(U_{j+1}^k + U_{j-1}^k)}_{\text{solution at } t_k}$$

In order to find U_j^2 we need to know U_j^0 and U_j^1.

We consider $n = 3$:



boundary conditions: $U_0^k = 0,\ U_4^k = 0$

initial conditions: $U_j^0 = f_j,\ U_j^{-1} = U_j^1 - 2\Delta t g(x_j) = U_j^1$, since $g(x_j) = 0$.

First find $\vec{U}^1 = \begin{pmatrix} U_1^1 \\ U_2^1 \\ U_3^1 \end{pmatrix}$

$$\vec{U}^1 = \begin{pmatrix} U_1^1 \\ U_2^1 \\ U_3^1 \end{pmatrix} = \underbrace{\begin{pmatrix} 2(1-s) & s & 0 \\ s & 2(1-s) & s \\ 0 & s & 2(1-s) \end{pmatrix}}_{A} \begin{pmatrix} U_1^0 \\ U_2^0 \\ U_3^0 \end{pmatrix}$$

$$+ \quad \underbrace{s \begin{pmatrix} U_0^0 \\ 0 \\ U_4^0 \end{pmatrix}}_{b} - \begin{pmatrix} U_1^{-1} \\ U_2^{-1} \\ U_3^{-1} \end{pmatrix}$$

Use $U_j^0 = f_j$ and $U_j^{-1} = U_j^1 - 2\Delta t g_j$

$$\vec{U}^1 = \begin{pmatrix} U_1^1 \\ U_2^1 \\ U_3^1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 2(1-s) & s & 0 \\ s & 2(1-s) & s \\ 0 & s & 2(1-s) \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}$$
$$+ \quad \frac{s}{2} \begin{pmatrix} U_0^0 \\ 0 \\ U_4^0 \end{pmatrix} + \Delta t \begin{pmatrix} g_1 \\ g_2 \\ g_3 \end{pmatrix}$$

$$\vec{U}^1 = \frac{1}{2} A \vec{U}^0 + \frac{1}{2} \vec{b} + \vec{d}$$

For this example, $U_0^0 = 0,\ U_4^0 = 0$ and:

$$\frac{\partial U_j^0}{\partial t}(x, t = 0) = g(x_j) = 0 \Rightarrow \vec{d} = \vec{0}$$

for $\vec{U}^2, \ldots, \vec{U}^m$ we have:

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

for $1 \leq k \leq m$:

$$\vec{U}^{k+1} = \begin{pmatrix} U_1^{k+1} \\ U_2^{k+1} \\ U_3^{k+1} \end{pmatrix} = \underbrace{\begin{pmatrix} 2(1-s) & s & 0 \\ s & 2(1-s) & s \\ 0 & s & 2(1-s) \end{pmatrix}}_{A} \begin{pmatrix} U_1^k \\ U_2^k \\ U_3^k \end{pmatrix}$$
$$+ \quad \underbrace{s \begin{pmatrix} U_0^k \\ 0 \\ U_4^k \end{pmatrix}}_{b} - \begin{pmatrix} U_1^{k-1} \\ U_2^{k-1} \\ U_3^{k-1} \end{pmatrix}$$

$$\vec{U}^{k+1} = A\vec{U}^k + \vec{b} - \vec{U}^{k-1}$$

The matlab code is **Wave1D.m**.

In our example $U_0^k = 0,\ U_4^k = 0$ and $\vec{b} = \vec{0}$, since $U_0^k = 0 = U_4^k$
At fixed boundaries $U(0,t) = 0 = U(a,t) \Rightarrow$ wave is reflected. We plot the numerical solution in figure 8.1.

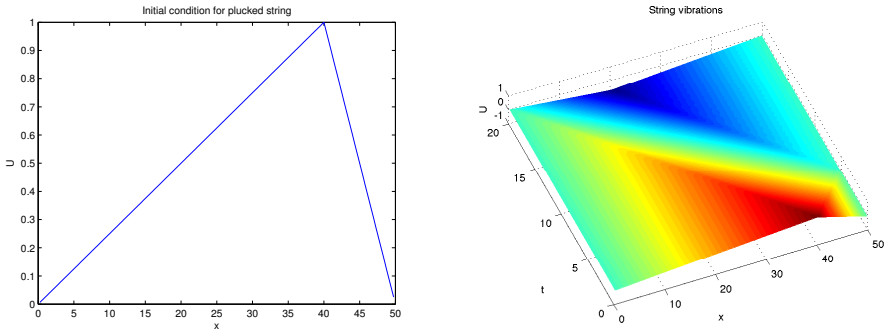


Figure 8.1: Initial conditions in (a) and matlab solution using explicit central difference method for 1D wave equation in (b)

We can compare with *D'Alembert's solution* which gives:

$$U(x,t) = \frac{1}{2}[f(x-ct) + f(x+ct)] \text{ since } U_t(x,0) = 0$$
$$\text{where } U(x,0) = f(x) \text{ (initial conditions) for } -\infty < x < \infty$$

What if we want to solve the wave equation for $0 \leq x \leq a$, with fixed boundary condition $U(t,0) = 0 = U(t,a)$? We can extend D'Alembert's general solution for $U_{tt} = c^2 U_{xx}$ with initial conditions:$U(x,0) = f(x)$ $U_t(x,0) = g(x)$:

$$U(x,t) = \frac{f(x+ct) + f(x-ct)}{2} + \frac{1}{2c}\int_{x-ct}^{x+ct} g(z)dz$$
$$\text{for } -\infty \leq x \leq \infty$$

In our example we have initial conditions:

$$U_t(x,0) = 0,\ U(x,0) = f(x) = \begin{cases} \frac{1.25x}{a}, & 0 \leq x \leq 0.8a \\ 5(1 - \frac{x}{a}), & \text{for } x \geq 0.8a \end{cases}$$

$$0 \leq x \leq a$$

with *fixed* boundary conditions:
The boundary condition $U(0,t) = 0$ is equivalent to f and g being odd functions:

$$U(0,t) = 0 \Rightarrow f(-x) = -f(x)$$
$$g(-x) = -g(x)$$
$$(f \text{ and } g \text{ are odd functions})$$

The boundary condition $U(a,t) = 0$ is equivalent to f and g being periodic with period $2a$:

$$\begin{aligned} U(a,t) = 0 \Rightarrow f(x+2a) &= f(x) \\ g(x+2a) &= g(x) \\ (f \text{ and } g \text{ are periodic with period } 2a) & \end{aligned}$$

Since $U_t(x,0) = g(x) = 0$ the analytical solution for our example:

$$U(t,x) = \frac{f(x+ct) + f(x-ct)}{2}$$

and we can compare the analytical solution with the numerical solution in figure 8.2.

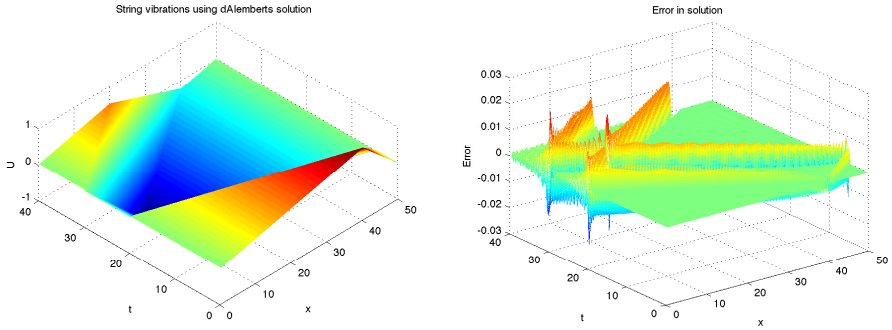


Figure 8.2: D'Alembert's solution in (a) and error using numerical matlab solution using explicit central difference method for 1D wave equation in (b)

8.1.2 1-D Wave Equation with Friction

The matlab code for this example is **Wave1DFriction.m**.

We consider friction due to viscosity of medium and density of string. Suppose we are solving:

$$\ddot{U} + 2\kappa \dot{U} = c^2 U_{xx}, \quad 0 \leq x \leq a = 50, \quad 0 \leq t \leq T = 20$$

The friction term κ opposes motion of string and means that eventually vibrations decay with time.

Suppose string is initially plucked in 2 places:

We have initial conditions:

$$\begin{aligned} U(x,0) &= \begin{cases} 0, & 0 \le x \le 0.1a \\ 5(10x-a), & 0.1a \le x \le 0.2a \\ 5(-10x+3a), & 0.2a \le x \le 0.3a \\ 0, & 0.3a \le x \le 0.7a \\ 5(10x-7a), & 0.7a \le x \le 0.8a \\ 5(-10x+9a), & 0.8a \le x \le 0.9a \\ 0, & x \ge 0.9a \end{cases} \\ U_t(x,0) &= 0 \end{aligned}$$

and boundary conditions: $U(x,0) = 0$, $U(x,a) = 0$.
Again we use central difference for U_{xx} and U_{tt} as in section 8.1.1.
We use a leap-frog step for U_t

$$\frac{\partial U_j^k}{\partial t} = \frac{U_j^{k+1} - U_j^{k-1}}{2\Delta t}$$

Now we substitute difference approximations into $U_{tt} + 2\kappa\, U_t = c^2 U_{xx}$

$$\frac{U_j^{k+1} - 2U_j^k + U_j^{k-1}}{\Delta t^2} + \kappa \frac{U_j^{k+1} - U_j^{k-1}}{\Delta t} = \frac{c^2(U_{j+1}^k - 2U_j^k + U_{j-1}^k)}{\Delta x^2}$$

let $s = \frac{c^2 \Delta t^2}{\Delta x^2}$
Rearranging for U_j^{k+1} gives:

$$U_j^{k+1} = \frac{1}{1+\kappa\Delta t}\left\{2(1-s)U_j^k - (1-\kappa\Delta t)U_j^{k-1} + s(U_{j+1}^k + U_{j-1}^k)\right\}$$

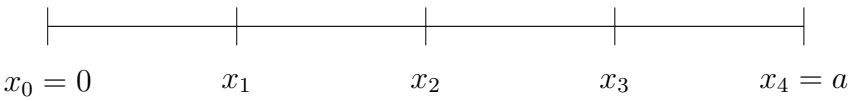
Special care is again needed to solve for U_j^1 which needs U_j^0 and the ghost point, U_j^{-1}. To find U_j^{-1} we use initial condition:

$$\frac{\partial U}{\partial t}(x, t=0) = \frac{\partial U_j^0}{\partial t} = 0 = \frac{U_j^1 - U_j^{-1}}{2\Delta t}$$
$$\text{or } U_j^{-1} = U_j^1 \qquad (\text{since } U_t(x,0) = 0)$$

We evaluate U_j^1:

$$\begin{aligned} U_j^1 &= \frac{1}{1+\kappa\Delta t}\left\{2(1-s)U_j^0 - (1-\kappa\Delta t)\underbrace{U_j^{-1}}_{=U_j^1} + s(U_{j+1}^0 - U_{j-1}^0)\right\} \\ &\Rightarrow \frac{2}{1+\kappa\Delta t}\,U_j^1 = \frac{1}{1+\kappa\Delta t}\left\{2(1-s)U_j^0 + s(U_{j+1}^0 - U_{j-1}^0)\right\} \\ &\Rightarrow U_j^1 = \frac{1}{2}\left\{2(1-s)U_j^0 + s(U_{j+1}^0 - U_{j-1}^0)\right\} \end{aligned}$$

Example $n = 3$



$$U_0^k = 0 = U(0,t), \quad U_{n+1}^k = U_4^k = U(a,t)$$

Again we solve for time step $k = 1$, $\vec{U}^1$ first:

$$\vec{U}^1 = \begin{pmatrix} U_1^1 \\ U_2^1 \\ U_3^1 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 2(1-s) & s & 0 \\ s & 2(1-s) & s \\ 0 & s & 2(1-s) \end{pmatrix}\begin{pmatrix} U_1^0 \\ U_2^0 \\ U_3^0 \end{pmatrix} + \frac{s}{2}\begin{pmatrix} U_0^0 \\ 0 \\ U_4^0 \end{pmatrix}$$

and the solution for time steps, $k \geq 1$, $\vec{U}^{k+1}$ are given by:

$$\begin{aligned} \vec{U}^{k+1} &= \begin{pmatrix} U_1^{k+1} \\ U_2^{k+1} \\ U_3^{k+1} \end{pmatrix} = \underbrace{\frac{1}{1+\kappa\Delta t}\begin{pmatrix} 2(1-s) & s & 0 \\ s & 2(1-s) & s \\ 0 & s & 2(1-s) \end{pmatrix}}_{A}\begin{pmatrix} U_1^k \\ U_2^k \\ U_3^k \end{pmatrix} \\ &+ \underbrace{\frac{s}{1+\kappa\Delta t}\begin{pmatrix} U_0^k \\ 0 \\ U_4^k \end{pmatrix}}_{b} - \underbrace{\frac{1-\kappa\Delta t}{1+\kappa\Delta t}}_{e}\begin{pmatrix} U_1^{k-1} \\ U_2^{k-1} \\ U_3^{k-1} \end{pmatrix} \\ &= A\vec{U}^k + \vec{b} - e\vec{U}^{k-1} \end{aligned}$$

The numerical solution is plotted in figure 8.3 below.

The matlab code is **Wave1DFriction.m**

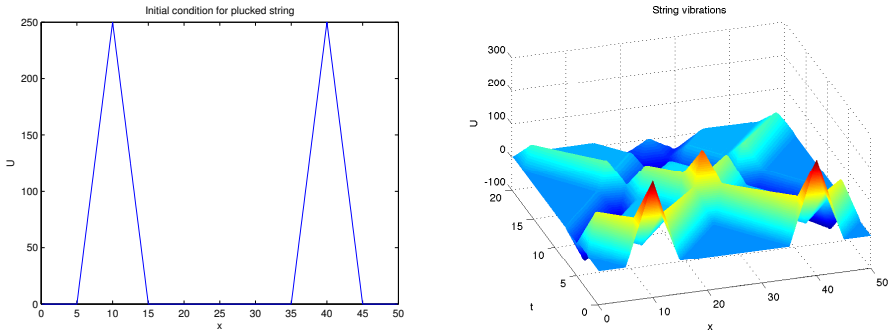


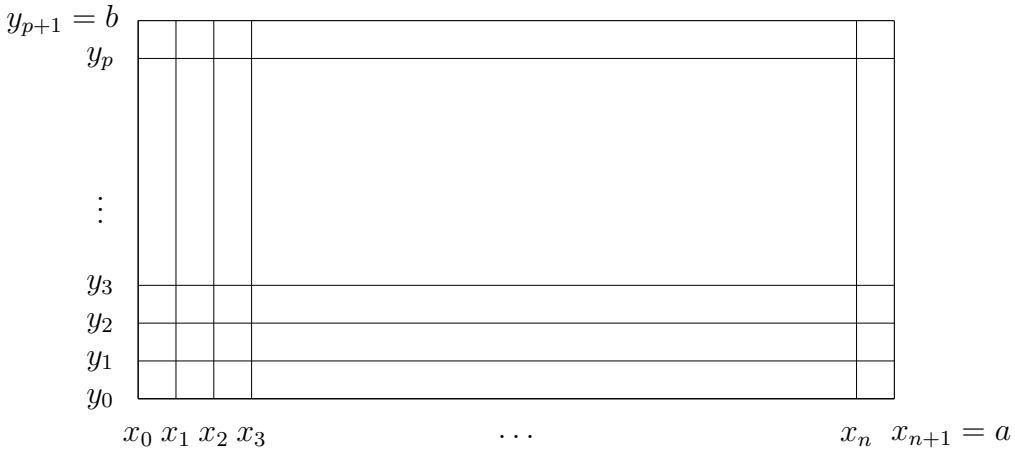
Figure 8.3: Initial conditions in (a) and matlab solution using explicit central difference method for 1D wave equation with friction in (b)

8.2 2-D Wave Equation

$$U_{tt} = \beta(U_{xx} + U_{yy}), \quad 0 \le x \le a,\ 0 \le y \le b,\ 0 \le t \le T$$

8.2.1 Example: vibrations of a thin elastic membrane fixed at its walls

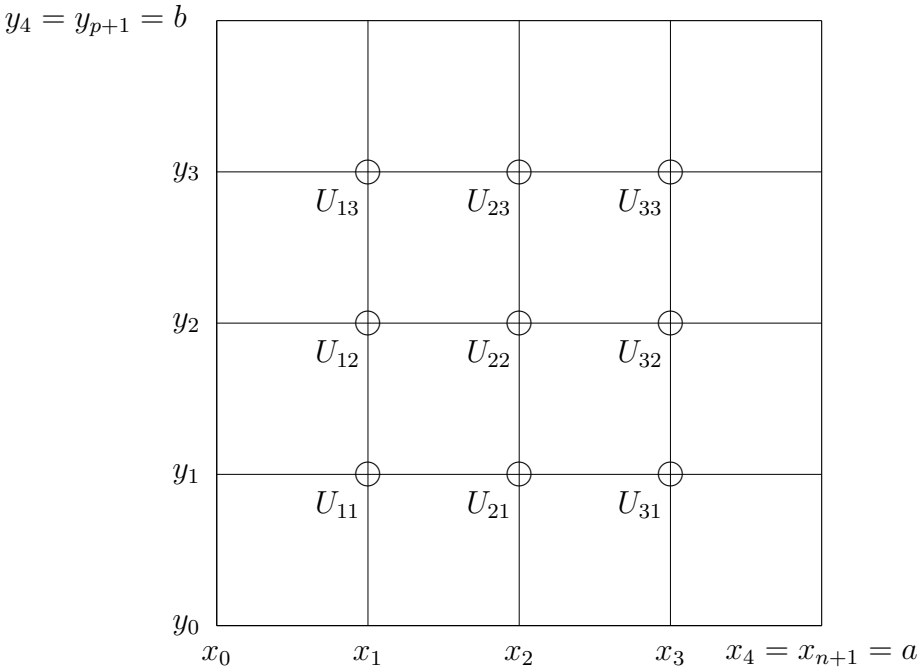
We discretise in x and y-directions:



We discretise: $\Delta t = \frac{T}{m},\ \Delta x = \frac{a}{n+1},\ \Delta y = \frac{b}{p+1},\ t_k = k\Delta t,\ x_i = i\Delta x,\ y_j = j\Delta y 0 \le k \le m,\ 0 \le i \le n+1,\ 0 \le j \le p+1$, and let $U_{ij}^k = U(t_k, x_i, y_j)$

Suppose we solve for $n = 3$ and $p = 3$ and have Dirichlet boundary conditions:
$U(0, y, t) = 0 = U^k_{oj}, \quad U(a, y, t) = 0 = U^k_{n+1,j} = U^k_{4j}, \quad U(x, 0, t) = 0 = U^k_{i0}, \quad U(x, b, t) = 0 = U^k_{i,p+1} = U^k_{i4}$
and initial conditions:
$U(x, y, 0) = f(x, y) = f_{ij} \quad U_t(x, y, 0) = g(x, y) = g_{ij}$.

Since we have Dirichlet boundary conditions: the outer boundaries of the region we are solving for are known: $U^k_{0,j}, U^k_{n+1,j}, U^k_{i,0}, U^k_{i,p+1}$, and we need to find the interior values: $U^k_{i,j}$ for $1 \le i \le n$ and $1 \le j \le p$.



We will use the *2-D Central Difference Method*

$$\begin{aligned} U_{tt} &= \frac{U^{k+1}_{ij} - 2U^k_{ij} + U^{k-1}_{ij}}{\Delta t^2}, \\ U_{xx} &= \frac{U^k_{i+1,j} - 2U^k_{ij} + U^k_{i-1,j}}{\Delta x^2}, \\ U_{yy} &= \frac{U^k_{i,j+1} - 2U^k_{ij} + U^k_{i,j+1}}{\Delta y^2} \end{aligned}$$

We let $s_x = \frac{\beta \Delta t^2}{\Delta x^2}$, $s_y = \frac{\beta \Delta t^2}{\Delta y^2}$ and substitute the central difference approx-

imations into our PDE, $U_{tt} = \beta(U_{xx} + U_{yy})$ we solve for U_{ij}^{k+1}:

$$U_{ij}^{k+1} = 2U_{ij}^{k}(1 - s_x - s_y) - U_{ij}^{k-1} + s_x(U_{i+1,j}^{k} + U_{i-1,j}^{k}) + s_y(U_{i,j+1}^{k} + U_{i,j-1}^{k})$$

computing $\vec{U}^{k+1}$ uses the solution at $\vec{U}^{k}$ and $\vec{U}^{k-1}$.
For first time step U_{ij}^{1} needs U_{ij}^{0} and U_{ij}^{-1}. Again we need to use the initial conditions to find the ghost point, U_{ij}^{-1}:

$$\frac{\partial U_{ij}^{0}}{\partial t} = U_t(x, y, 0) = \frac{U_{ij}^{1} - U_{ij}^{-1}}{2\Delta t} = g(x_i, y_j) = g_{ij} \Rightarrow U_{ij}^{-1} = U_{ij}^{1} - 2\Delta t g_{ij}$$

Solution at first time step $k = 1$:

$$U_{ij}^{1} = U_{ij}^{0}(1 - s_x - s_y) + \Delta t g_{ij} + \frac{s_x}{2}(U_{i+1,j}^{0} + U_{i-1,j}^{0}) + \frac{s_y}{2}(U_{i,j+1}^{0} + U_{i,j-1}^{0})$$

If we let $\vec{U}^{k} = \begin{pmatrix} U_{11}^{k} \\ U_{12}^{k} \\ U_{13}^{k} \\ U_{21}^{k} \\ U_{22}^{k} \\ U_{23}^{k} \\ U_{31}^{k} \\ U_{32}^{k} \\ U_{33}^{k} \end{pmatrix}$

then for time steps, $k > 1$, the solution is:

$$U_{ij}^{k+1} = 2U_{ij}^{k}(1 - s_x - s_y) - U_{ij}^{k-1} + s_x(U_{i+1,j}^{k} + U_{i-1,j}^{k}) + s_y(U_{i,j+1}^{k} + U_{i,j-1}^{k})$$

and we can write this in vector form:

$$\vec{U}^{k+1} = A\vec{U}^{k} + \vec{b} - \vec{U}^{k-1}$$

where:

$$A = \begin{pmatrix} \lambda & s_y & 0 & s_x & 0 & 0 & 0 & 0 & 0 \\ s_y & \lambda & s_y & 0 & s_x & 0 & 0 & 0 & 0 \\ 0 & s_y & \lambda & 0 & 0 & s_x & 0 & 0 & 0 \\ s_x & 0 & 0 & \lambda & s_y & 0 & s_x & 0 & 0 \\ 0 & s_x & 0 & s_y & \lambda & s_y & 0 & s_x & 0 \\ 0 & 0 & s_x & 0 & s_y & \lambda & 0 & 0 & s_x \\ 0 & 0 & 0 & s_x & 0 & 0 & \lambda & s_y & 0 \\ 0 & 0 & 0 & 0 & s_x & 0 & s_y & \lambda & s_y \\ 0 & 0 & 0 & 0 & 0 & s_x & 0 & s_y & \lambda \end{pmatrix}$$

and $\lambda = 2(1 - s_x - s_y)$

$$b = \begin{pmatrix} s_x U^k_{01} + s_y U^k_{10} \\ s_x U^k_{02} \\ s_x U^k_{03} + s_y U^k_{14} \\ s_y U^k_{20} \\ 0 \\ s_y U^k_{24} \\ s_x U^k_{41} + s_y U^k_{30} \\ s_x U^k_{42} \\ s_x U^k_{43} + s_y s^k_{34} \end{pmatrix}$$

8.2.2 Examples of wave equation

1. **Elastic wave propagation through rocks** in *1-D*

$$\sigma_{xx,x} = \rho U_{tt} \tag{8.1}$$

where

$$\begin{aligned} \sigma_{xx} &= E\varepsilon_{xx},\ \sigma_{xx} = \text{stress},\ \varepsilon_{xx} = \text{strain} \\ &= E\frac{\partial U}{\partial x} \end{aligned}$$

$$8.1 \Rightarrow EU_{xx} = \rho U_{tt} \quad \text{or } U_{tt} = \frac{E}{\rho} U_{xx}$$

elastic waves propagate with speed $\sqrt{\frac{E}{\rho}}$

2. **Electromagnetic Wave Equation**

$$c^2\nabla^2 E = \ddot{E} \text{ and } c^2\nabla^2 B = \ddot{B} \tag{8.2}$$

From Maxwell's equations where E is electric field, B is magnetic field. Derived using:

$$\nabla \cdot E = \frac{\rho}{\epsilon_0}, \quad \nabla \times E = -\frac{\partial B}{\partial t} \tag{8.3}$$

$$\nabla \cdot B = 0, \quad \nabla \times B = \mu_0 \varepsilon_0 \frac{\partial E}{\partial t} \tag{8.4}$$

taking curl of 8.3 and 8.4 and using $\nabla \times (\nabla \times \vec{V}) = \nabla(\nabla \cdot \vec{V}) - \nabla^2 \vec{V}$ and $\nabla(\nabla \cdot E) = \nabla\left(\frac{\rho}{\epsilon_0}\right) = 0, \quad \nabla(\nabla \cdot B) = 0$ gives Equation 8.2 where $c = \sqrt{\frac{1}{\mu_0 \varepsilon_0}} = 3 \times 10^8 \text{m/s}$.

3. **Schrödinger's Wave Equation**

$$i\hbar\frac{\partial\Psi}{\partial t} = H\Psi$$

- for a wavefunction Ψ of a quantum system defined by Hamiltonian, H.
  eg. $H = KE + PE = -\frac{\hbar^2\nabla^2}{2m} + V(r)$
- numerical solutions also need to satisfy $\int_{-\infty}^{\infty} |\Psi(x)|^2_{dx} = 1$

Chapter 9

Finite element method

9.1 An introduction to the Finite Element Method

- Finite difference (FD) method is an approximation to the differential equation.
- Finite element method (FEM) is an approximation to its solution.
- FD methods are usually based on the assumption of regular domains eg line in 1-D, rectangle in 2-D with regular elements
- FEM is better for irregular regions as the domain can be partitioned into any simple subregion such as triangles or rectangles in 2-D or bricks and tetrahedra in 3-D. Figure 9.1 shows a finite element mesh with triangles for an irregular domain.

Example: Solving Poisson's equation in 1-D using FEM

$$-U_{xx} = q, \quad 0 \leq x \leq L \tag{9.1}$$

We consider Dirichlet boundary conditions: $U(0) = U(L) = 0$. A weak solution of (9.1) considers the variational form of (9.1):

$$\int_0^L U_{xx}(x)\phi(x)dx + \int_0^L q(x)\phi(x) = 0, \tag{9.2}$$

where $\phi(x)$ satisfy the boundary conditions: $\phi(0) = \phi(L) = 0$.

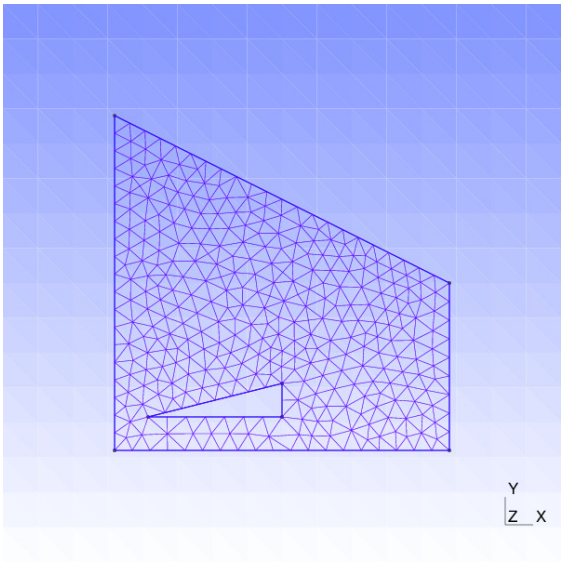


Figure 9.1: FEM mesh with triangles

We can integrate the first term by parts:

$$\begin{aligned}\int_0^L U_{xx}(x)\phi(x)dx &= U_x(x)\phi(x)]_{x=0}^{x=L} - \int_0^L U_x(x)\phi_x(x)dx \\ &= -\int_0^L U_x(x)\phi_x(x)dx\end{aligned}$$

using $\phi(0) = \phi(L) = 0$.

Then (9.2) becomes:

$$\int_0^L U_x(x)\phi_x(x)dx = \int_0^L q(x)\phi(x)dx \tag{9.3}$$

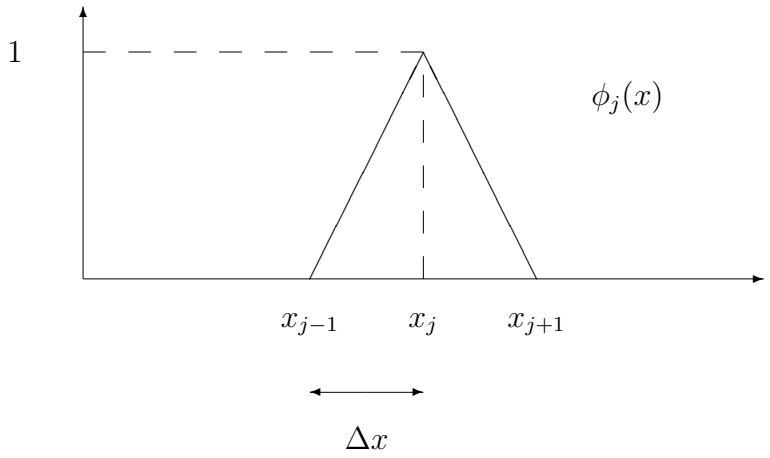
Equation (9.3) holds for all functions $\phi(x)$ which are piece-wise continous and satisfy the bc: $\phi(0) = \phi(L) = 0$.

To solve equation (9.3) using the FEM we again introduce a mesh (as in FD) on the interval $[0, L]$ with mesh points $x_j = j\Delta x$, $j = 0, \ldots, n+1$ where $\Delta x = \frac{L}{n+1}$. To complete the discretisation we must choose a basis for $\phi(x)$. The most common basis chosen for $\phi(x)$ are the "hat" functions, $\phi_j(x)$. We solve (9.3) using these:

$$\phi(x) = \sum_{j=1}^{n} a_j\phi_j(x)$$

where

$$\phi_j(x) = \begin{cases} 0, & \text{for } 0 \le x \le x_{j-1} \\ \frac{1}{\Delta x}(x - x_{j-1}), & \text{for } x_{j-1} \le x \le x_j \\ 1 - \frac{1}{\Delta x}(x - x_j), & \text{for } x_j \le x \le x_{j+1} \\ 0, & \text{for } x \ge x_{j+1} \end{cases}$$



with this construction: $\phi_j(x_i) = \delta_{ij}$ and:

$$\phi_j'(x) = \frac{\partial \phi_j}{\partial x} = \begin{cases} 0, & \text{for } 0 < x < x_{j-1} \\ \frac{1}{\Delta x}, & \text{for } x_{j-1} < x < x_j \\ -\frac{1}{\Delta x}, & \text{for } x_j < x < x_{j+1} \\ 0, & \text{for } x > x_{j+1} \end{cases}$$

We let $\phi(x) = \sum_{j=1}^{n} a_j \phi_j(x)$ and $\phi(x_i) = a_i$ for $i = 1, \ldots, n$, and $\phi(0) = \phi_1(0) = 0$ and $\phi(L) = \phi_n(L) = 0$ so that $\phi(x)$ satisfies boundary conditions.

The hat functions are advantageous as a basis as they are nearly "orthonormal", ie. $\int_0^L \phi_j(x)\phi_k(x)dx = 0$ when $|j - k| > 1$.

Using FEM we seek an **approximate** solution to (9.3) which is satisfied for all basis functions, $\phi_i(x)$, for $i = 1, \ldots, n$:

$$\int_0^L U_x(x)\phi_x(x)dx = \int_0^L q(x)\phi(x)dx$$

and require that 9.3 be satisfied for $\phi = \phi_i,\ i = 1, \ldots, n$. We also expand the solution $U(x)$ using the hat functions ϕ_i as a basis:

$$U(x) \approx U_h(x) = \sum_{j=1}^{n} b_j \phi_j(x)$$

This simplifies equation 9.3 and we solve for $\phi = \phi_i,\ i = 1, \ldots, n$: ie.

$$\int_0^L U_h^{'}(x)\phi_i^{'}(x)dx \quad = \quad \int_0^L q(x)\phi_i(x)dx, \qquad \text{for } i = 1, \ldots, n$$

where $f^{'}(x) = \frac{\partial f}{\partial x}$.

$$\begin{aligned} LHS &= \int_0^L U_h^{'}(x)\phi_i^{'}(x)dx \\ &= \int_0^L \sum_{j=1}^n b_j \phi_j^{'}(x)\phi_i^{'}(x)dx \\ &= \sum_{j=1}^n C_{i,j} b_j \end{aligned}$$

where $C_{i,j} = \int_0^L \phi_j^{'}(x)\phi_i^{'}(x)dx$. $C_{i,j}$ is known as the stiffness matrix in mechanics.

To find the coefficients b_j which define our solution $U(x)$ we must solve n linear equations:

$$LHS = \sum_{j=1}^n C_{i,j} b_j = RHS = \int_0^L q(x)\phi_i(x)dx = q_i \tag{9.4}$$

for $i = 1, \ldots, n$ with $q_i = \int_0^L q(x)\phi_i(x)dx)$.

We approximate the solution by expanding in the basis of "hat" functions: $U(x) \approx \sum_{j=1}^n b_j \phi_j(x)$. Thus we only need to know the coefficients b_j to define our solution $U(x)$ and FEM solves the following equation for vector $\vec{b} = (b_1, \ldots, b_n)$:

$$\begin{aligned} \sum_{j=1}^n b_j \int_0^L \phi_{j,x}(x)\phi_{i,x}(x)dx &= \int_0^L q(x)\phi_i(x)dx, \\ \text{or } \sum_{j=1}^n b_j C_{i,j} &= q_i \end{aligned}$$

for $i = 1, \ldots, n$.

We will show that the stiffness matrix C is tridiagonal for this example. We are solving the above system for coefficients b_j, thus we are solving $C\vec{b} = \vec{q}$ and can use iterative methods in FEM solutions too.

We can show that the stiffness matrix is tridiagonal:

$$C_{ij} \quad = \quad \int_0^L \phi_{j,x}(x)\phi_{i,x}(x)dx = \begin{cases} \frac{-1}{\Delta x}, & i = j-1 \\ \frac{2}{\Delta x}, & i = j \\ \frac{-1}{\Delta x}, & i = j+1 \\ 0, & \text{elsewhere} \end{cases}$$

We approximate q_i using:

$$\begin{aligned} q_i &= \int_0^L q(x)\phi_i(x)dx \approx q(x_i)\int_0^L \phi_i(x)dx \\ &= q(x_i)\left(\int_{x_{j-1}}^{x_j} \frac{1}{\Delta x}(x - x_{j-1})dx + \int_{x_j}^{x_{j+1}} 1 - \frac{1}{\Delta x}(x - x_j)dx\right) \\ &= \Delta x q(x_i) \end{aligned}$$

We can substitute the above simplifications into equation 9.4 and arrive at $C\vec{b} = \Delta x\vec{q}$ or $\frac{1}{\Delta x}C\vec{b} = \vec{q}$:

$$\begin{pmatrix} \frac{2}{\Delta x^2} & \frac{-1}{\Delta x^2} & 0 & \cdots \\ \frac{-1}{\Delta x^2} & \frac{2}{\Delta x^2} & \frac{-1}{\Delta x^2} & \ddots \\ 0 & \ddots & \ddots & \ddots \\ \vdots & \ddots & \frac{-1}{\Delta x^2} & \frac{2}{\Delta x^2} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{pmatrix}$$

Thus the matrix system to solve is the same as FD solution in this example and the solution involves inverting the stiffness matrix C:

$$\vec{b} = C^{-1}\Delta x\vec{q}$$

Iterative methods are useful in FEM too as it involves inverting large, sparse matrices.

Once $\vec{b}$ is known, the solution U to the PDE is given by:

$$U(x) \approx \sum_{j=1}^{n} b_j\phi_j(x)$$

This is a weak solution of the PDE $-U_{xx} = q$.

9.2 Comparing FEM solution to FD solution for our example

$$-U_{xx} = q, \quad 0 \leq x \leq L, \quad U(0) = U(L) = 0$$

9.2.1 FD solution

Discretise using $x_j = j\Delta x,\ j = 0, 1, \ldots, n+1$ where $\Delta x = \frac{L}{n+1},\ U_0 = 0 = U_{n+1}$ (using boundary conditions).

We let $U(x_j) = U_j,\ q(x_j) = q_j$. The central difference approximation to the PDE is:

$$\begin{aligned} U_{xx} &= \frac{U_{j+1} - 2U_j + U_{j-1}}{\Delta x^2} \\ \text{and} \quad -U_{xx} &= q \quad \text{becomes} \\ -\frac{U_{j+1} + 2U_j - U_{j-1}}{\Delta x^2} &= q_j \end{aligned}$$

We solve for $U_1, \ldots, n$ since U_0 and U_{n+1} given from boundary conditions and we can rewrite in matrix form:

$$\underbrace{\begin{pmatrix} \frac{2}{\Delta x^2} & \frac{-1}{\Delta x^2} & 0 & \cdots \\ \frac{-1}{\Delta x^2} & \frac{2}{\Delta x^2} & \frac{-1}{\Delta x^2} & \ddots \\ 0 & \ddots & \ddots & \ddots \\ \vdots & \ddots & \frac{-1}{\Delta x^2} & \frac{2}{\Delta x^2} \end{pmatrix}}_{\text{same coefficient matrix for FD as FEM}} \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_n \end{pmatrix} + \begin{pmatrix} -\frac{U_0(=0)}{\Delta x^2} \\ 0 \\ \vdots \\ -\frac{U_{n+1}=0}{\Delta x^2} \end{pmatrix} = \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{pmatrix}$$

In this example FEM and FD methods solve the same matrix system.

9.3 2-D Finite Element Method

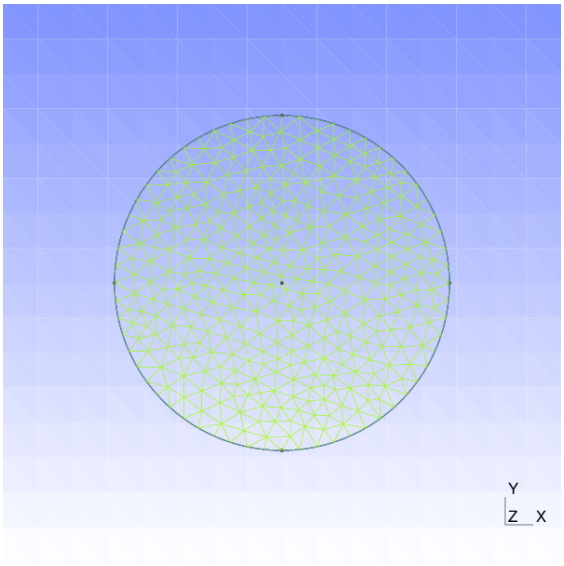


Figure 9.2: FEM mesh with triangles

We consider a triangular mesh (could also be rectangular) shown in figure 9.2 where G is the domain inside the circle and ∂G is the domain's boundary.

We are solving:

$$-\nabla^2 U = q \text{ in } G \tag{9.5}$$

with boundary conditions, $U = 0$ on ∂G.
The weak solution for U satisfies the variational form for Equation 9.5:

$$-\int\int_G \nabla^2 U(x,y)\phi(x,y)dxdy = \int\int_G q(x,y)\phi(x,y)dxdy \tag{9.6}$$

where $\phi(x,y) = 0$ on ∂G (satisfies boundary conditions).
Using Green's first identity:

$$\begin{aligned}\int\int_G (\phi\nabla^2 U)dxdy &= \underbrace{\oint_{\partial G} \phi(\nabla U \cdot \hat{n})dS}_{=0 \text{ since } \phi=0 \text{ on } \partial G} - \int\int_G (\nabla\phi \cdot \nabla U)dxdy \\ &= -\int\int_G (\nabla\phi \cdot \nabla U)dxdy\end{aligned}$$

Thus equation 9.6 becomes:

$$\int\int_G (\nabla\phi \cdot \nabla U)dxdy = \int\int_G q\phi dxdy \tag{9.7}$$

which holds $\forall\phi \in G$ where $\phi = 0$ on ∂G.
Similarly to the 1-D case we seek an approximate solution to equation 9.7 by expanding $U(x,y)$ in a basis of 2-D *"hat"* functions:

$$U(x,y) \approx U_h(x,y) = \sum_{j=1}^{n} b_j \phi_j(x,y)$$

where $U_h(x_i, y_i) = b_i$ and $U_h = 0$ on ∂G.

9.3.1 2-D "hat functions"

The 2-D hat functions satisfy $\phi_j(x_j, y_j) = 1, \quad \phi_j(x_i, y_l) = 0$ if $i \neq j$ and $j \neq l$ at all other vertices. The 2D hat function is plotted in figure 9.3.

We require that equation 9.7 holds for all $\phi(x,y)$ and solve for $\phi = \phi_1, \phi_2, \ldots, \phi_n$:

$$\int\int_G \nabla U_h \cdot \nabla\phi_i dxdy = \int\int_G q\phi_i dxdy, \text{ for } i = 1, \ldots, n \tag{9.8}$$

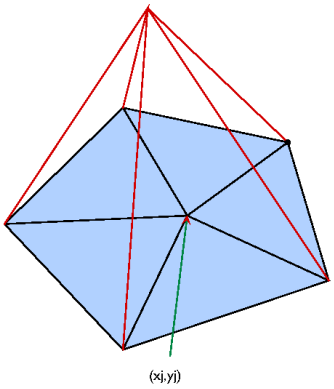


Figure 9.3: 2D hat function ($\phi_j(x_j, y_j) = 1, \quad \phi_j(x_i, y_l) = 0$ if $i \neq j$ and $j \neq l$)

$$LHS = \int \int_G \nabla U_h \cdot \nabla \phi_i dxdy = \sum_{j=1}^{n} C_{i,j} b_j$$

where $C_{i,j} = \int \int_G \nabla \phi_j \cdot \nabla \phi_i dxdy$ is called the "stiffness matrix".
Equation 9.8 becomes:

$$\sum_{j=1}^{n} C_{i,j} b_j = q_i, \text{ for } i = 1, \ldots, n \text{ where } q_i = \int \int_G q \phi_i dxdy.$$

If C is symmetric and positive definite then the system has a unique solution.

9.3.2 Example: 2-D Finite Element Method using eScript for elastic wave propagation from a point source.

- eScript is a general PDE solver which implements the finite element method written in python
  (see https://launchpad.net/escript-finley)

- eScript can be applied to any problem of the form:

  $$-(A_{jl} a_{,l} + B_j a)_{,j} + C_l a_{,l} + Da = -X_{j,j} + Y$$

  where a is the *scalar* we are solving for in this example.
  (eScript can also solve for a *vector* $\vec{a}$)

We are using *Einstein notation* and according to this convention if an index appears *twice* in a single term it implies we are summing over all possible values:

$$a_i f_{,ii} = a_i \frac{\partial^2 f_i}{\partial x_i^2} = a_1 \frac{\partial^2 f_1}{\partial x_1^2} + a_2 \frac{\partial^2 f_2}{\partial x_2}$$

We will see that the FEM takes care of *spatial* derivative in the problem below. However we still need to approximate *time* derivatives.

We want to solve the 2-D wave equation for a point source:

$$\Psi_{tt} = V_p^2(\Psi_{xx} + \Psi_{yy}) + F_{\text{PS}}$$

where p is the wave speed, Ψ is the wave-field and F_{PS} is the force due to the point source.
In eScript this becomes:

$$\begin{aligned} D_a &= -X_{j,j} + Y \\ \text{where } a &= \Psi_{tt} \\ D &= 1 \\ X_j &= -V_p^2 \Psi_{,j} \\ Y &= F_{\text{PS}} \end{aligned}$$

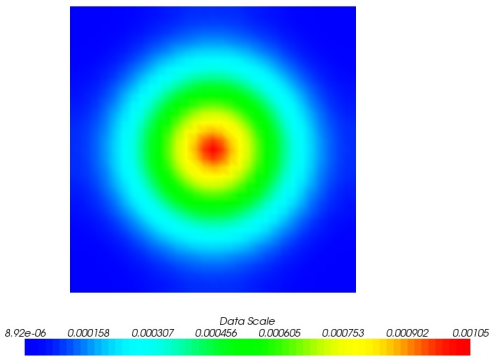


Figure 9.4: Plot of Euclidean normal of the displacement at $t > 0$ for a point source using eScript.

We solve for a^k at each time step t^k. Once a^k is known we use it to calculate the solution at the next time step, Ψ^{k+1} using the central difference

formula:

$$a^k = \frac{\partial^2 \Psi^k}{\partial t^2} \approx \frac{\Psi^{k+1} - 2\Psi^k + \Psi^{k-1}}{\Delta t^2}$$

or

$$\Psi^{k+1} = 2\Psi^k - \Psi^{k-1} - \Delta t^2 a^k$$

The eScript python code is **2Dpointsource.py** and the output from this code is shown in figure 9.4.

Chapter 10

Spectral methods

10.1 An introduction to spectral methods

- remove spurious dispersion and are highly accurate
- exponential convergence for smooth functions (smooth functions have rapidly decaying Fourier transforms)
- usually involve calling a Fast Fourier Transform (fft) subroutine.
- good for *smooth* solutions.
- Like the FEM, the spectral method also approximates the solution $U(x)$.
- FEM approximates the solution as a linear combination of piece-wise functions that are non-zero only on small subdomains ("hat" functions)- *local* approach.
- spectral methods approximate the solution as a linear combination of continuous functions that are generally nonzero throughout the domain (usually sinusoids or Chebychev polynomials)- *global* approach.

We will show an example using the spectral method where $U(x)$ is expanded as a Fourier series and this series and its spatial derivatives are then substituted into the PDE resulting in a system of ODEs in time.

10.1.1 Example 1: Comparing the accuracy of solutions of a variable speed wave equation with either the spectral or finite difference method

Spectral method for variable speed wave equation

In this example we will compare the accuracy of either the spectral or finite difference method when solving the 1D advection equation with a variable wave speed, $c(x) = \frac{1}{5} + \sin^2(x-1)$. First we will derive the solution using the spectral method:

$$\begin{aligned} U_t + c(x)U(x) &= 0, \quad 0 \le x \le 2\pi \text{ and } 0 \le t \le 9 \\ U(x,0) &= \exp(-100(x-1)^2) \\ c(x) &= \frac{1}{5} + \sin^2(x-1) \\ U(0,t) &= U(2\pi,t) \quad \text{periodic boundary condition} \end{aligned}$$

The matlab code is **spectral_variable_wave_speed.m**

Again we discretise in space and time: $\Delta x = \frac{2\pi}{2n} = \frac{\pi}{n},\ \Delta t = \frac{T}{m}$
$x_j = j\Delta x,\ j = 0, 1, 2, \ldots, 2n-1$
$t_k = k\Delta t,\ k = 0, 1, 2, \ldots, m$
$U(x_j, t_k) = U_j^k$

The spectral method uses the discrete Fourier transform of $U(x_j, t)$:

$$\begin{aligned} \hat{U}_\nu &= F(U), \\ &= \sum_{j=0}^{2n-1} U(x_j,t)\exp(-ix_j\nu) \\ &= \sum_{j=0}^{2n-1} U(x_j,t)\exp(-i2\pi j\nu/(2n)), \text{ using } x_j = j\Delta x = j2\pi/2n \\ &= \sum_{j=0}^{2n-1} U(x_j,t)\exp(-i\pi j\nu/n) \end{aligned}$$

for $\nu = -n+1, \ldots, n$.

$U(x_j, t)$ is then defined as the inverse discrete Fourier transform of $\hat{U}_\nu$:

$$\begin{aligned} U(x_j,t) &= U_j = F^{-1}(\hat{U}), \\ &= \frac{1}{2n}\sum_{\nu=-n+1}^{n} \hat{U}_\nu \exp(ix_j\nu) \\ &= \frac{1}{2n}\sum_{\nu=-n+1}^{n} \hat{U}_\nu \exp(i2\pi j\nu/(2n) \end{aligned}$$

where $j = 0, \ldots, 2n-1$.

With this definition the spatial derivatives are:

$$\begin{aligned}\frac{\partial U(x_j,t)}{\partial x} &= \frac{1}{2n}\sum_{\nu=-n+1}^{n} i\nu\hat{U}_\nu \exp(ix_j\nu) \\ &= F^{-1}(i\nu\hat{U}) \\ &= F^{-1}(i\nu F(U))\end{aligned}$$

We solve the advective equation with variable wave speeds and compare the solution with either FD or spectral method:

$$\begin{aligned}U_t + c(x)U_x &= 0 \\ \text{where } c(x) &= \frac{1}{5} + \sin^2(x-1) \\ \text{ic: } U(x,0) &= \exp(-100(x-1)^2) \\ \text{periodic bc: } U(0,t) &= U(\pi,t)\end{aligned}$$

The central difference approximation is used for U_t and spectral method for U_x:

$$\underbrace{\frac{U_j^{k+1}-U_j^{k-1}}{2\Delta t}}_{\text{leap-frog for } U_t} + c(x_j)\underbrace{F^{-1}(i\nu F(U_j^k))}_{\text{spectral method for } U_x} = 0$$

$$\text{or } U_j^{k+1} = U_j^{k-1} + 2\Delta t c(x_j) F^{-1}(i\nu F(U_j^k))$$

For U_j^1 need U_j^0 and U_j^{-1}

$$U(x,0) = U_j^0 = \exp(-100(x-1)^2)$$

since $c(x) \approx \frac{1}{5}$ we can assume a constant wave speed of $\approx 1/5$ to calculate U_j^{-1} at $t = -\Delta t$.

$$U_j^{-1} = U(x,-\Delta t) = U(x + c(x)\Delta t) \approx U^0(x + \frac{1}{5}\Delta t) = \exp(-100(x + \frac{\Delta t}{5} - 1)^2)$$

The matlab code is **spectral_variable_wave_speed.m**.

Comparing accuracy of solution with spectral method vs. finite difference method

Solve again using finite difference.

$$U_t + c(x)U_x = 0$$

This matlab code is **fd_variable_wave_speed.m**.

The Lax method is used for U_t and central difference method for U_x:

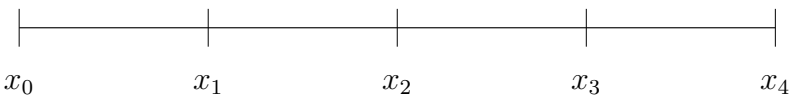
$$\begin{aligned} \frac{\partial U}{\partial t} &= \frac{U_j^{k+1} - \frac{1}{2}(U_{j-1}^k + U_{j+1}^k)}{\Delta t} \\ \frac{\partial U}{\partial x} &= \frac{U_{j+1}^k - U_{j-1}^k}{2\Delta x} \\ c(x_j) &= c_j. \end{aligned}$$

Plug the formulas into the PDE:

$$\underbrace{\frac{U_j^{k+1} - \frac{1}{2}(U_{j-1}^k + U_{j+1}^k)}{\Delta t}}_{U_t} + c(x_j) \underbrace{\frac{U_{j+1}^k - U_{j-1}^k}{2\Delta x}}_{U_x} = 0$$

$$\text{or } U_j^{k+1} = \frac{1}{2}(1 + sc_j)U_{j-1}^k + \frac{1}{2}(1 - sc_j)U_{j+1}^k$$

where $s = \Delta t/\Delta x$. Using 4 elements:



$U_0^k = 0 = U_4^k$ given by boundary conditions
$U_j^0 = U(x, 0)$ given by initial conditions

$$\vec{U}^{k+1} = \begin{pmatrix} U_1^{k+1} \\ U_2^{k+1} \\ U_3^{k+1} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 0 & 1 - sc_1 & 0 \\ 1 + sc_2 & 0 & 1 - sc_2 \\ 0 & 1 + sc_3 & 0 \end{pmatrix} \begin{pmatrix} U_1^k \\ U_2^k \\ U_3^k \end{pmatrix} + \frac{1}{2} \begin{pmatrix} (1 + sc_1)U_0^k \\ 0 \\ (1 - sc_3)U_4^k \end{pmatrix}$$

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

Figure 10.1(b) shows that the solution using finite differences is *much* worse than the spectral method because dispersion is introduced in FD method when a variable wave speed is applied. However figure 10.1(a) shows the spectral method performs well when smooth initial conditions of a Gaussian pulse are used and there is very little dispersion present.

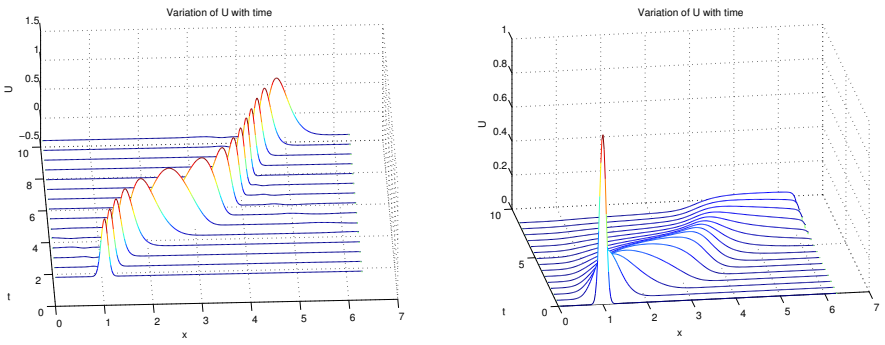


Figure 10.1: Numerical solution for 1D advection equation with initial conditions of a smooth Gaussian pulse with variable wave speed using the spectral method in (a) and finite difference method in (b)

10.1.2 Example 2 Comparing spectral and finite difference methods with constant wave speed conditions and initial conditions of a non-smooth pulse

We solve the advective equation with constant wave speed ($c = 1$) with initial conditions of a box pulse and compare the solution with either FD or spectral method:

$$\begin{aligned} U_t + U_x &= 0 \\ \text{ic: } U(x,0) &= \begin{cases} 1, & 0.5 \le x \le 1. \\ 0, & \text{otherwise} \end{cases} \\ \text{periodic bc: } U(0,t) &= U(\pi,t) \end{aligned}$$

- The matlab code for the *spectral* solution is **spectral_c_1_box.m**.
- The matlab code for the *FD* solution is **fd_c_1_box.m**. In this code we have chosen the time step carefully so that no dispersion is present for a constant wave speed of $c = 1$. Please see section 7.7.2 for a discussion on dispersion in finite difference methods.

Figure 10.2(a) shows that the solution with an initial condition which is not smooth like the box pulse we used here using the spectral method is *much* worse than the finite difference method. This is because the spectral method uses Fourier series to approximate the initial conditions and is unable to

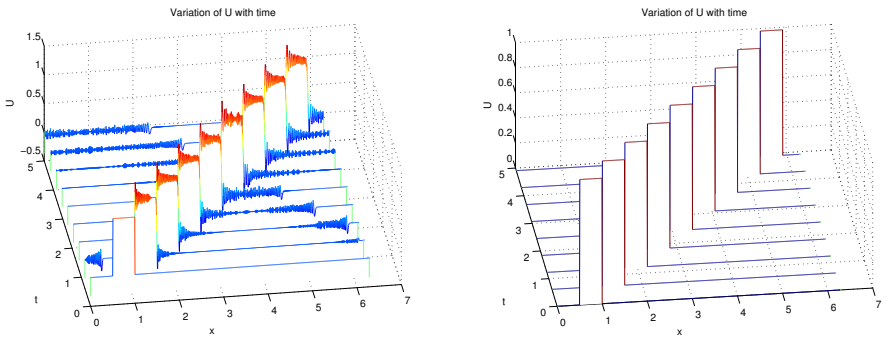


Figure 10.2: Numerical solution for 1D advection equation with initial conditions of a box pulse with a constant wave speed using the spectral method in (a) and finite difference method in (b)

approximate non-smooth initial conditions accurately. It is important to note that for the numerical solution using the FD method in figure 10.2(b) that we were able to remove dispersion in this example by carefully choosing $\Delta t = \Delta x/c$ (see section 7.7.2).

Part III

Nonlinear partial differential equations

Chapter 11

Shock wave

11.1 Analytical solution: Method of characteristics

Analytical solution to the shock wave equation is given by the method of characteristics. We will illustrate this method first for a linear first-order PDE:

- can be applied to first order PDEs:

$$a(x,t)U_x + b(x,t)U_t + c(x,t)U = 0 \tag{11.1}$$

  with initial conditions $U(x,0) = f(x)$

We change co-ordinates from (x,t) to (x_0,s) so that our *PDE* 11.1 becomes an *ODE* for certain *characteristic* curves in the x-t plane. The new variable, s, will vary along the characteristic curves, whereas x_0 will remain constant. How does it work?

$$\text{We let: } \frac{dx}{ds} = a(x,t), \quad \frac{dt}{ds} = b(x,t)$$

$$\text{Then } \frac{dU}{ds} = \frac{dx}{ds}\frac{\partial U}{\partial x} + \frac{dt}{ds}\frac{\partial U}{\partial t} = aU_x + bU_t \tag{11.2}$$

We substitute 11.2 into 11.1:

$$\Rightarrow \frac{dU}{ds} + c(x,t)U = 0$$

This is an *ODE* along the characteristic curves satisfying the characteristic equations:

$$\frac{dx}{dt} = a(x,t) \quad \text{and} \quad \frac{dt}{ds} = b(x,t)$$

11.1.1 Example 1: Using method of characteristics to solve the linear 1-D advection equation

$$\begin{aligned} U_t + cU_x &= 0 \\ \text{initial conditions: } x(s=0) &= x_0, \qquad t(s=0) = 0 \\ U(x, t=0) &= f(x) \quad \text{or} \quad U(s=0) = f(x_0) \end{aligned}$$

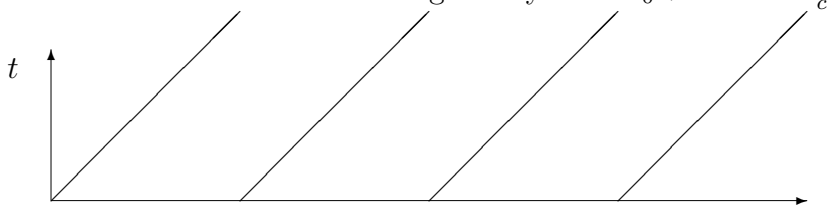
$$\begin{aligned} \frac{dx}{ds} = c \quad &\Rightarrow \quad x = cs + k_1, \text{ use } x(0) = x_0 = k_1 \\ &\Rightarrow \quad x = cs + x_0 \end{aligned}$$

$$\begin{aligned} \frac{dt}{ds} = 1 \quad &\Rightarrow \quad t = s + k_2, \text{ use } t(0) = 0 = k_2 \\ &\Rightarrow \quad t = s \end{aligned}$$

and since $t = s \Rightarrow x = ct + x_0$
Solve for U:

$$\frac{dU}{ds} = \frac{dt}{ds}U_t + \frac{dx}{ds}U_x = U_t + cU_x = 0$$

ie $\frac{dU}{ds} = 0 \Rightarrow U = k_3$ (U is constant along characteristic curves).
Use initial conditions $U(s = 0, x_0) = f(x_0) = k_3$
ie $U = f(x_0) = f(x - ct)$ since $x_0 = x - ct$.
This is the same as D'Alembert's solution for a wave moving to the right at speed c. The characteristic curves are given by: $x = x_0 + ct$ or $t = \frac{1}{c}(x - x_0)$.



slope ($= \frac{1}{c}$) and U are constant along curves

11.1.2 Example 2: Using method of characteristics to solve the *nonlinear* inviscid Burger's equation

Shock waves result when solving the nonlinear inviscid Burger's equation:

$$\begin{aligned} U_t + UU_x &= 0 \\ \text{initial conditions: } x(s=0) &= x_0, \qquad t(s=0) = 0 \\ U(x, t=0) &= f(x) \quad \text{or} \quad U(s=0) = f(x_0) \end{aligned}$$

Now the wave speed is *not* constant but depends on the amplitude $U(x,t)$. The characteristic equations are:

$$\frac{dt}{ds} = 1 \Rightarrow t = s \quad (\text{using } t(0) = 0)$$
$$\frac{dx}{ds} = U \Rightarrow x = Ut + x_0 \quad (\text{using } x(0) = x_0 \text{ and } t = s)$$

Again:

$$\begin{aligned} \frac{dU}{ds} &= \frac{dt}{ds}\frac{\partial U}{\partial t} + \frac{dx}{ds}\frac{\partial U}{\partial x} \\ &= U_t + UU_x = 0 \\ \Rightarrow U &= k_3 = f(x_0) = f(x - Ut) \end{aligned}$$

So $U = f(x - Ut)$ is given *implicitly* since U is a function of itself. The characteristic curves given by

$$t = \frac{1}{U}(x - x_0) = \frac{1}{f(x_0)}(x - x_0)$$

The characteristic curves *no longer* have *constant* slope - they may cross (meaning U is multiply defined $\rightarrow$ *shock* waves) or be discontinuous (regions with no solution for $U \rightarrow$ *expansion* waves) as we will see in the next example.

Example

Solving $U_t + UU_x = 0$ with the following initial conditions:

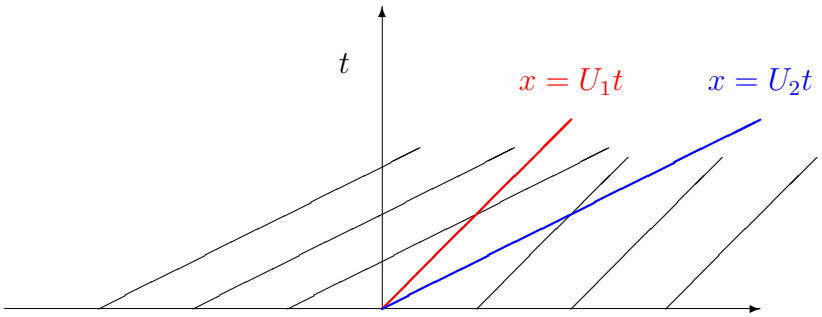
$$U(x, t = 0) = f(x) = \begin{cases} U_1, & x > 0 \\ U_2, & x < 0 \end{cases}$$

$$t = \begin{cases} \frac{1}{U_1}(x - x_0), & x > 0 \quad \text{or } x = U_1 t + x_0 \\ \frac{1}{U_2}(x - x_0), & x < 0 \quad \text{or } x = U_2 t + x_0 \end{cases}$$

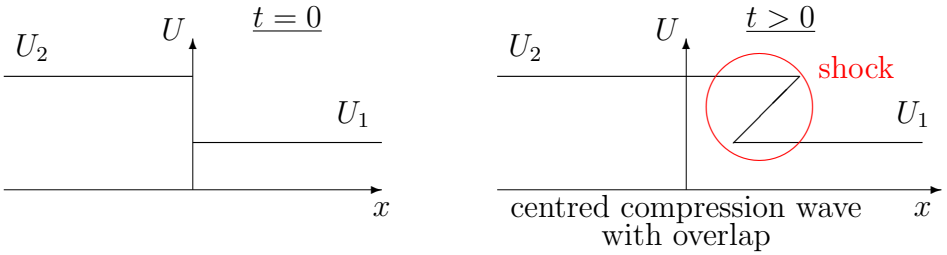
2 cases

$$\begin{aligned} U_1 &< U_2 \text{ - compression wave} \rightarrow \textit{shock} \\ U_1 &> U_2 \text{ - expansion wave} \rightarrow \textit{rarefaction} \end{aligned}$$

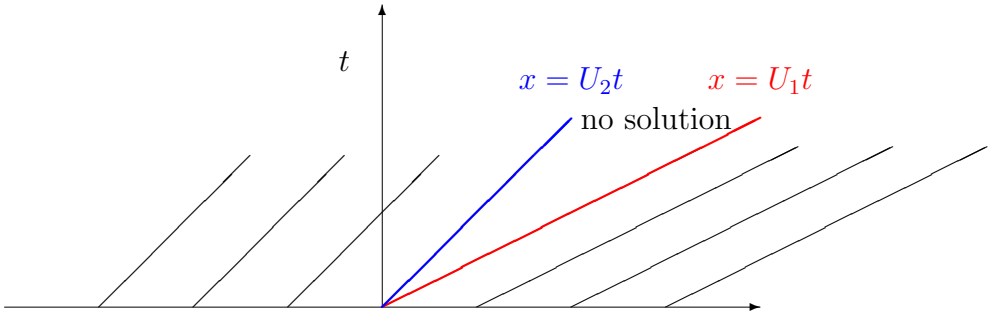
Case 1: Shock wave $U_1 < U_2$



In the fan bounded by $x = U_1 t$ and $x = U_2 t$ the characteristic curves are multi-valued leading to shocks (breaking waves). We illustrate this below:



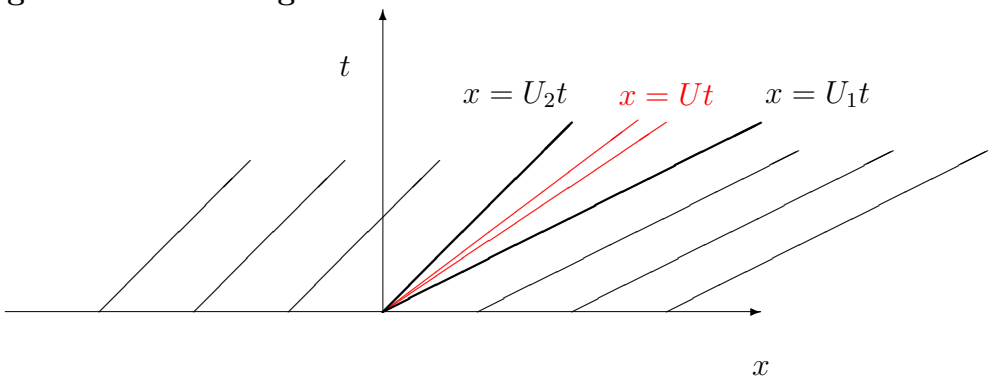
Case 2: Rarefaction or expansion wave $U_1 > U_2$



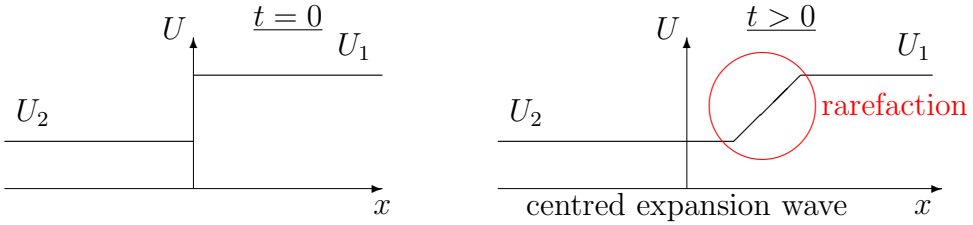
The solution is single-valued for $t > 0$ unlike the shock wave case. However in wedge between $x = U_2 t$ and $x = U_1 t$ there is *no* information. We assume $x = U_t$ in wedge since $U_2 t \leq x \leq U_1 t$ and speeds vary $U_2 \leq U \leq U_1$

and add solution to the wedge.

Adding solution to wedge:



$$\text{Thus } U = \begin{cases} U_2, & \frac{x}{t} < U_2 \\ \frac{x}{t}, & U_2 < \frac{x}{t} < U_1 \\ U_1, & \frac{x}{t} > U_1 \end{cases}$$



11.2 Numerical Solution for nonlinear Burger's Equation

$$U_t + UU_x = 0, \quad 0 \leq x \leq 1, \quad 0 \leq t \leq 1$$

$$U(x,0) = f(x) = \exp(-10(4x-1)^2)$$

Solution given implicitly by $U(x,t) = f(x - Ut)$ so *speed* depends on *amplitude*, U.

We study the numerical solution using 3 methods but we will see in each case that the numerical solution fails to produce a shock wave because we are unable to produce multi-valued solutions.

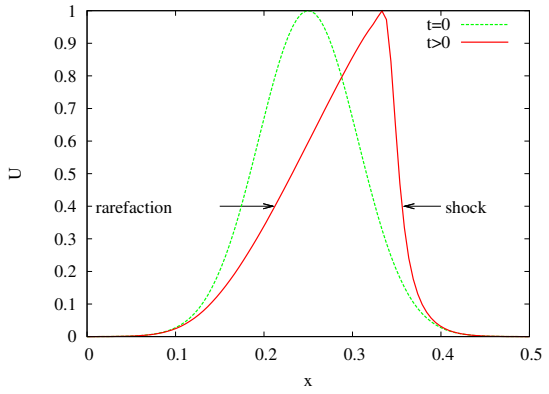


Figure 11.1: The analytical solution $U(x,t) = f(x - Ut)$ is plotted to show how shock and rarefaction develop for this example

11.2.1 Example I: Finite difference solution with *Lax Method*

The matlab code is **Shock_Lax.m**. We are solving:

$$U_t + UU_x = 0$$

$$\begin{aligned} \frac{\partial U_j^k}{\partial t} &= \frac{U_j^{k+1} - \frac{1}{2}(U_{j-1}^k + U_{j+1}^k)}{\Delta t} \quad \text{(Lax method for } U_t\text{)} \\ \frac{\partial U_j^k}{\partial x} &= \frac{U_{j+1}^k - U_{j-1}^k}{2\Delta x} \quad \text{(leap-frog for } U_x\text{)} \end{aligned}$$

The Courant condition only holds for *linear* wave equation. A good guess is $\Delta t << \frac{\Delta x}{\max(U)}$.(Waves travel at a maximum wave speed $U = 1$.)

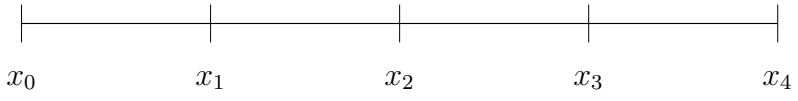
Put difference equations into PDE:

$$U_t + UU_x = 0 \quad \text{becomes:}$$

$$U_j^{k+1} = \frac{1}{2}\left\{U_{j+1}^k(1 - sU_j^k) + U_{j-1}^k(1 + sU_j^k)\right\} \tag{11.3}$$

Where $s = \frac{\Delta t}{\Delta x}$

Use boundary conditions $U(0,t) = U(1,t) = 0$ and for 4 elements:



So $U_0^k = 0 = U_4^k$ given by boundary conditions and we can rewrite Equation 11.3 as a matrix system of equations:

$$\begin{aligned}
\vec{U}^{k+1} &= \begin{pmatrix} U_1^{k+1} \\ U_2^{k+1} \\ U_3^{k+1} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 0 & 1 - sU_1^k & 0 \\ 1 + sU_2^k & 0 & 1 - sU_2^k \\ 0 & 1 + sU_3^k & 0 \end{pmatrix} \begin{pmatrix} U_1^k \\ U_2^k \\ U_3^k \end{pmatrix} \\
&+ \frac{1}{2} \begin{pmatrix} (1 + sU_1^k)U_0^k \\ 0 \\ (1 - sU_3^k)U_4^k \end{pmatrix} \\
&= \frac{1}{2} A\vec{U}^k + \frac{1}{2}\vec{b}
\end{aligned}$$

A varies with time because of U_j^k term in matrix!

We can compare the difference between the matlab code for the linear 1D advective equation ($U_t + U_x = 0$), **Lax_Flux.m** in section 7.7.2 and the shock wave equation ($U_t + UU_x = 0$) above, **Shock_Lax.m**.

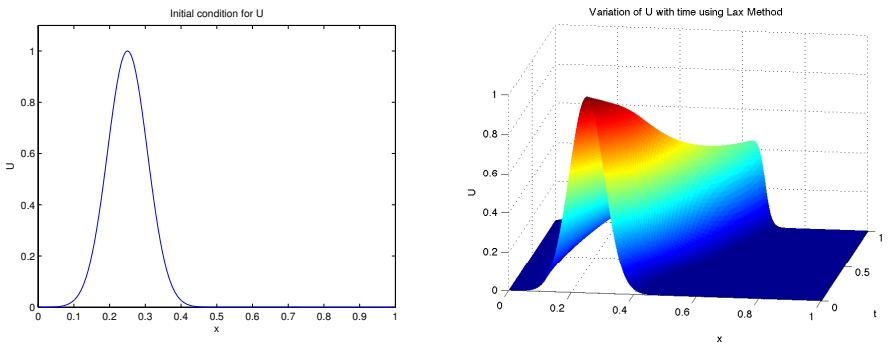


Figure 11.2: Initial conditions in (a) and solution for nonlinear Buger's equation using the Lax method in (b)

When we compare the analytical solution given by the method of characteristics to the numerical solution given by the Lax method we can see that the numerical solution is accurate for the linear 1D advection equation (see numerical solution in figure 7.1) but fails to give a shock wave for the nonlinear Burger's equation in figure 11.2. The Lax method introduces dispersion into the numerical solution and in the nonlinear case this "removes" the shock wave instability and flattens the wave front.

11.2.2 Example II: Solution using *Method of Lines*

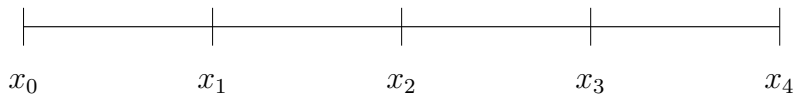
The matlab code is **Shock_mol.m** and **Uprime2.m**.

$$\frac{\partial U}{\partial t} = -UU_x$$

Using the method of lines solution (as demonstrated in section 2.3) we only replace spatial derivative U_x with *FD* approximation.

$$\frac{\partial U_j}{\partial x} = \frac{U_{j+1} - U_{j-1}}{2\Delta x}, \Rightarrow \frac{\partial U_j}{\partial t} = -U_j \left(\frac{U_{j+1} - U_{j-1}}{2\Delta x} \right)$$

and again we show the case with 4 elements:



$U_0 = 0 = U_4$ from boundary conditions.

$$\frac{\partial \vec{U}}{\partial t} = \begin{pmatrix} \dot{U}_1 \\ \dot{U}_2 \\ \dot{U}_3 \end{pmatrix} = \frac{1}{2\Delta x} \begin{pmatrix} 0 & -U_1 & 0 \\ U_2 & 0 & -U_2 \\ 0 & U_3 & 0 \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ U_3 \end{pmatrix} + \frac{1}{2\Delta x} \begin{pmatrix} U_1 U_0 \\ 0 \\ -U_3 U_4 \end{pmatrix}$$

or

$$\dot{\vec{U}} = \frac{1}{2\Delta x} A(U)\vec{U} + \vec{b}$$

When the shock develops in figure 11.3(b) the numerical solution becomes *unstable* using the method of lines.

11.2.3 Example III: Solution using *Spectral Method*

The matlab code is **Shock_spectral.m**.

$$U_t = -UU_x, \quad 0 \le x \le 2\pi$$

(we can change variable: $\xi = \frac{x}{2\pi}$ later so that the range for ξ is $0 \le \xi \le 1$)

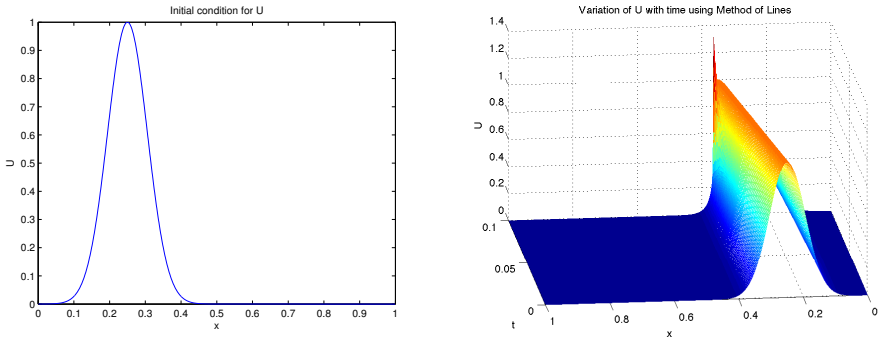


Figure 11.3: Initial conditions in (a) and solution for nonlinear Buger's equation using the method of lines in (b)

Spectral method

We let $U(x_j, t_k) = U_j^k,\ x_j = j\Delta x,\ j = 0, 1, \ldots, 2n-1,\ t_k = k\Delta t,\ k = 0, 1, \ldots, m,$ and $\Delta t = \frac{T}{m}$.

Take the discrete Fourier transform of U:

$$\begin{aligned} \hat{U}_\nu &= F(U) = \sum_{j=0}^{2n-1} U(x_j, t) \exp(-i x_j \nu) \text{ for } \nu = -n+1, \ldots, n \\ \text{where } x_j &= j\Delta x = \frac{j\pi}{n} \end{aligned}$$

then

$$\begin{aligned} U_j &= F^{-1}(\hat{U}) = \frac{1}{2n} \sum_{\nu=-n+1}^{n} \hat{U}_\nu \exp(i x_j \nu) \\ \text{for } j &= 0, 1, \ldots, 2n-1 \end{aligned}$$

and

$$\begin{aligned} \frac{\partial U_j^k}{\partial x} &= \frac{1}{2n} \sum_{\nu=-n+1}^{n} i\nu \hat{U}_\nu \exp(i x_j \nu) \\ &= F^{-1}(i\nu \hat{U}) \\ &= F^{-1}(i\nu\ F(U)) \end{aligned}$$

Use leap-frog for U_t:

$$U_t = \frac{U_j^{k+1} - U_j^{k-1}}{2\Delta t}$$

then $U_t = -UU_x$ becomes:

$$U_j^{k+1} = U_j^{k-1} - 2\Delta t U_j^k \; F^{-1}(i\nu \; F(U_j^k))$$

To find U_j^{-1} for spectral method we assume wave speed ≈ 1 and:

$$\begin{aligned} U_j^{-1} &= U(x, -\Delta t) = U^0(x - Ut) = f(x - Ut) \\ &\approx f(x + \Delta t) = \exp(-10(4(x + \Delta t) - 1)^2) \end{aligned}$$

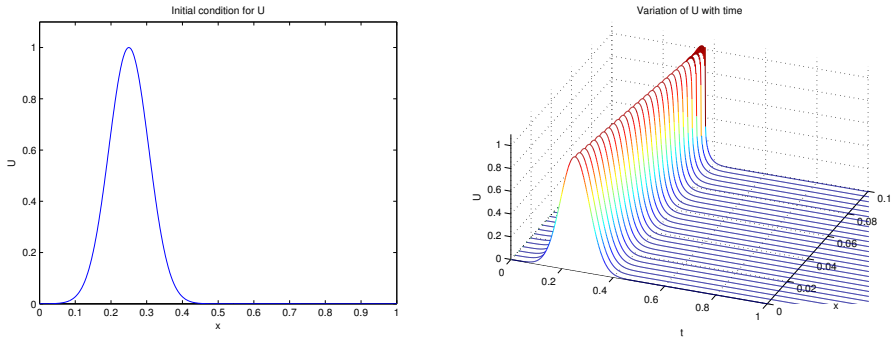


Figure 11.4: Initial conditions in (a) and solution for nonlinear Buger's equation using the spectral method in (b)

Again as in the method of lines the numerical solution becomes *unstable* as shock develops in figure 11.4 using the spectral method.

Chapter 12

Korteweg-de Vries Equation

12.1 Solitons

- solitons or solitary waves result from solution of the KdV equation
- KdV equation is a model for shallow water waves:

$$U_t + UU_x + U_{xxx} = 0 \quad \text{nonlinear PDE}$$

- analytical solutions exist
- solitons move in isolation and propagate without changing form. Velocity is amplitude dependent (linearly proportional to maximum amplitude).
- the nonlinear term causes waves to steepen (UU_x
- the dispersive term causes waves to disperse (U_{xxx})
- these effects are in exact balance for solitons $\rightarrow$ waveform maintains its size, shape and speed as it travels.
- solitons pass through each other without change of form except shifted.

12.2 Analytical solution

$$U_t + UU_x + U_{xxx} = 0$$

Let $U = f(\xi) = f(x - Vt)$ where $\xi = x - Vt$ then:

$$\frac{\partial}{\partial t} = \frac{\partial \xi}{\partial t}\frac{\partial}{\partial \xi} = -V\frac{\partial}{\partial \xi}, \quad \frac{\partial}{\partial x} = \frac{\partial \xi}{\partial x}\frac{\partial}{\partial \xi} = \frac{\partial}{\partial \xi}$$

Let $f'(\xi) = \frac{df}{\partial \xi}$ then $U_t + UU_x + U_{xxx} = 0$ becomes: (using $U = f(\xi) = f(x - Vt)$)

$$-Vf' + ff' + f''' = 0$$

We integrate once: (use $ff' = \frac{d}{d\xi}(\frac{f^2}{2})$)

$$-V\int f'd\xi + \int \frac{d}{d\xi}(\frac{f^2}{2})d\xi + \int f'''d\xi = 0$$

$$\Rightarrow -Vf + \frac{f^2}{2} + f'' = C \tag{12.1}$$

Multiply by f' and integrate again:

$$-\int Vff'd\xi + \int f'\frac{f^2}{2}d\xi + \int f'f''d\xi = \int Cf'd\xi + c_0$$

$$\text{Term 1 } = \int Vff'd\xi = Vf^2 - \int Vff'd\xi \Rightarrow \int Vff'd\xi = \frac{V}{2}f^2$$

$$\text{Term 2 } = \int f'\frac{f^2}{2}d\xi = \frac{f^3}{2} - \int f^2f'd\xi \Rightarrow \int f'\frac{f^2}{2}d\xi = \frac{f^3}{6}$$

$$\text{Term 3 } = \int f'f''d\xi = (f')^2 - \int f''f'd\xi \Rightarrow \int f'f''d\xi = \frac{f'^2}{2}$$

So multiplying 12.1 by f' and integrating again gives:

$$\frac{-V}{2}f^2 + \frac{f^3}{6} + \frac{f'^2}{2} = Cf + C_0 \tag{12.2}$$

We assume boundary conditions $f(\xi) \to 0$, $f'(\xi) \to 0$ as $\xi \to \pm\infty$.

So Equation 12.2$\Rightarrow C_0 = 0$ and Equation 12.1$\Rightarrow C = 0$.

We assume initial conditions for soliton:

$$U(x,0) = f(x) = A\text{sech}^2(\sqrt{\frac{A}{12}}x)$$

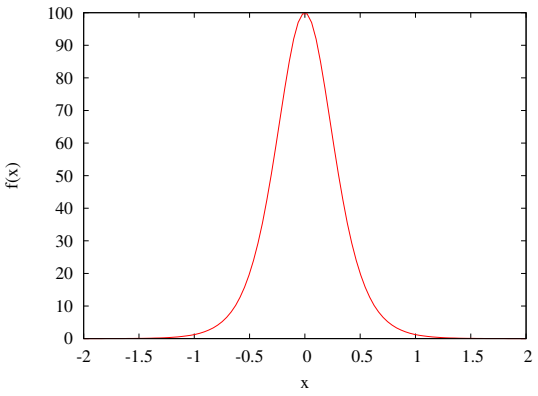


Figure 12.1: The initial conditions $U(x,0) = f(x) = A\text{sech}^2(\sqrt{\frac{A}{12}}x)$.

So $f(0) = A$ and $f'(0) = 0$ and $f'(\xi) \leq 0$ for $\xi \geq 0$.
Rearranging Equation 12.2 gives

$$f'^2 = Vf^2 - \frac{f^3}{3} \tag{12.3}$$

Use

$$\begin{aligned} f(0) = A, \quad f'(0) = 0 \quad &\Rightarrow \quad 0 = A^2(V - \frac{A}{3}) \\ &\Rightarrow \quad V = \frac{A}{3} \end{aligned}$$

Since $f'(\xi) \leq 0$ for $\xi \geq 0$ we take the negative square root of 12.3:

$$f' = \frac{-f}{\sqrt{3}}\sqrt{A - f}$$

This can be integrated analytically by making a change of variable if we let $f = A\text{sech}^2\theta$ then $df = -2A\text{sech}^2\theta \tanh\theta d\theta$ and integrating:

$$f' = \frac{-f}{\sqrt{3}}\sqrt{A - f}$$

Since we assumed $\xi \geq 0$ we integrate from $\xi = 0$ in integration limits:

$$\begin{aligned} \Rightarrow \int_0^\xi \frac{\frac{df}{d\xi}d\xi}{f\sqrt{A-f}} &= -\frac{1}{\sqrt{3}}\int_0^\xi d\xi \\ \Rightarrow -\frac{\xi}{\sqrt{3}} = \int_{f(0)}^{f(\xi)} \frac{df}{f\sqrt{A-f}} &= \int_A^f \frac{df}{f\sqrt{A-f}} \end{aligned}$$

now substitute $f = A\text{sech}^2\theta$

$$\begin{aligned} &= \int_0^\theta \frac{-2A\text{sech}^2\theta\tanh\theta d\theta}{A\text{sech}^2\theta\sqrt{A - A\text{sech}^2\theta}} \\ &= \int_0^\theta \frac{-2\tanh\theta d\theta}{\sqrt{A}\sqrt{1-\text{sech}^2\theta}} \end{aligned}$$

now use $1 - \text{sech}^2\theta = \tanh^2\theta$

$$\begin{aligned} \Rightarrow \int_0^\theta \frac{-2}{\sqrt{A}}d\theta &= \frac{-2\theta}{\sqrt{A}} = -\frac{\xi}{\sqrt{3}} \\ \Rightarrow \theta &= \sqrt{\frac{A}{12}}\xi \\ U(x,t) &= f(\xi) = A\text{sech}^2\theta \\ &= A\text{sech}^2\left(\sqrt{\frac{A}{12}}\xi\right) \\ \Rightarrow U(x,t) &= \underbrace{A\text{sech}^2\left(\sqrt{\frac{A}{12}}(x-\frac{A}{3}t)\right)}_{\text{soliton travelling to right}} \quad (\text{using } V = \frac{A}{3}) \end{aligned}$$

where $\text{sech}x = \frac{1}{\cosh x}$ and $\cosh x = \frac{1}{2}(e^{-x} + e^x)$.

12.3 Numerical solution of KdV Equation

$$U_t + UU_x + U_{xxx} = 0, \quad 0 \le x \le 2\pi$$

with periodic boundary conditions $U(0) = U(2\pi)$.
and initial conditions:

$$U(x,0) = A\text{sech}^2(\sqrt{\frac{A}{12}}(x-\pi)), \quad A = 100$$

The analytical solution is:

$$U(x,t) = A\text{sech}^2\left(\sqrt{\frac{A}{12}}(x-\pi-\frac{A}{3}t)\right)$$

If we apply the spectral method directly we find that the linear term, U_{xxx}, involves high frequencies making the numerical solution unstable as we will see in section 12.3.1. Section 12.3.2 shows how to modify this term to gain stability using a modified spectral method.

12.3.1 Solving directly with Spectral Method

$$U_t + UU_x + U_{xxx} = 0$$

The matlab code is **SpectralDirectSoliton.m**.
Take discrete Fourier transform of U:

$$\hat{U}_\nu = F(U) = \sum_{j=0}^{2n-1} U(x_j, t)\exp(-ix_j\nu), \quad \text{for } \nu = -n+1, \ldots, n$$

and the inverse discrete Fourier transform of $\hat{U}$:

$$U_j = F^{-1}(\hat{U}) = \frac{1}{2n}\sum_{\nu=-n+1}^{n} \hat{U}_\nu \exp(ix_j\nu), \quad \text{for } j = 0, \ldots, 2n-1$$

We calculate spatial derviatives using spectral method:

$$\begin{aligned} U_x &= \frac{\partial U_j^k}{\partial x} = \frac{1}{2n}\sum_{\nu=-n+1}^{n} \hat{U}_\nu (i\nu)\exp(ix_j\nu) \\ &= F^{-1}(i\nu\hat{U}) = F^{-1}(i\nu F(U)) \end{aligned}$$

and:

$$\begin{aligned} U_{xxx} &= \frac{\partial^3 U_j^k}{\partial x^3} = \frac{1}{2n}\sum_{\nu=-n+1}^{n} \hat{U}_\nu (-i\nu^3)\exp(ix_j\nu) \\ &= F^{-1}(-i\nu^3\hat{U}) = F^{-1}(-i\nu^3 F(U)) \end{aligned}$$

$$\uparrow$$

At high wavenumbers, ν, this term causes instabilities in solution.
We use a leap-frog approximation for U_t:

$$\frac{\partial U_j^k}{\partial t} = \frac{U_j^{k+1} - U_j^{k-1}}{2\Delta t}$$

Plug approximations into PDE: $U_t + UU_x + U_{xxx} = 0$,

$U_j^{k+1} = U_j^{k-1} - 2\Delta t\left(U_j^k F^{-1}(i\nu F(U)) + F^{-1}(-i\nu^3 F(U))\right)$ $\Rightarrow$ solution blows up!

Figure 12.2 shows that the numerical solution for the KdV equation blows up using a direct spectral method. In the next section we modify the U_{xxx} term causing instabilities.

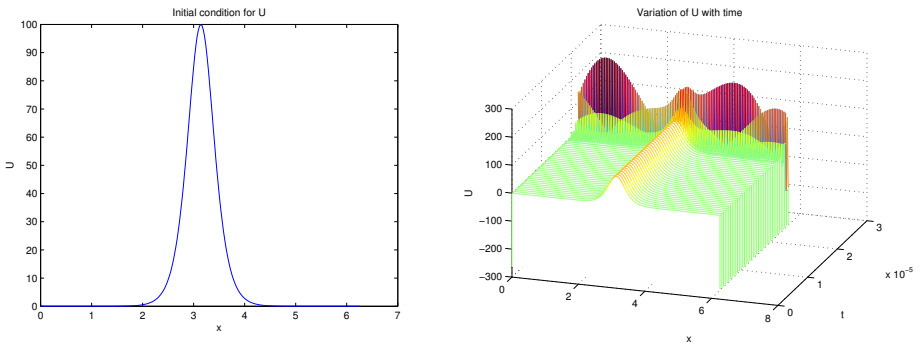


Figure 12.2: Initial conditions in (a) and solution for nonlinear KdV equation using the direct spectral method in (b)

12.3.2 Modifying U_{xxx} term causing instabilities in direct spectral method

The matlab code is **SpectralModifiedSoliton.m**.

The direct method solves:

$$U_j^{k+1} = U_j^{k-1} + 2\Delta t[U_j^k F^{-1}(ivF(U)) + F^{-1}(-iv^3 F(U))]$$

↑

The last term approximating U_{xxx} makes PDE very stiff at high wavenumbers.

To remove this instability for high wavenumbers we replace the last term with:

$$\sin(v^3\Delta t) \approx v^3\Delta t + 0(\Delta t^3) \quad \text{as } \Delta t \to 0 \text{ this is satisfied}$$

Using $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots$ and re-solve with this approximation:

$$U_j^{k+1} = U_j^{k-1} + 2\Delta t U_j^k F^{-1}(ivF(U)) + 2F^{-1}(-i\sin(v^3\Delta t)F(U))$$

$\Rightarrow$ This numerical solution is stable!
(See Fornberg and Whitham, Philos. Trans. Roy. Soc. London (1974))
Again use the same initial conditions:

$$\begin{aligned} U_j^{-1} &= U(x_j, -\Delta t) = U^0\left(x + \frac{A}{3}\Delta t\right) \\ &= f\left(x + \frac{A}{3}\Delta t\right) \\ &= A\text{sech}^2(\sqrt{\frac{A}{12}}\left(x + \frac{A}{3}\Delta t - \pi\right)) \end{aligned}$$

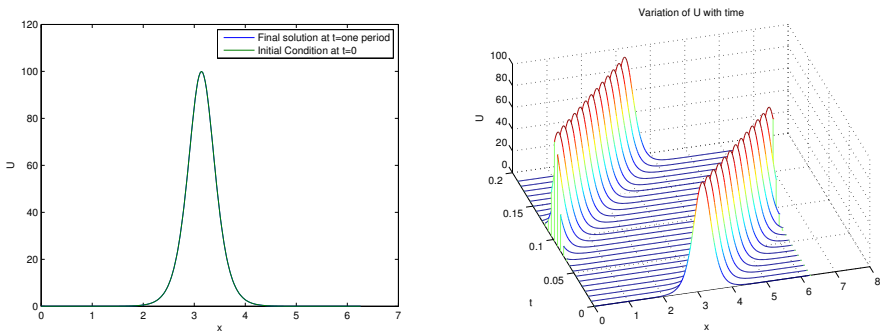


Figure 12.3: Initial conditions and final solution after one period in (a) and solution for nonlinear KdV equation using a modified spectral method in (b)

Figure 12.3 shows that the numerical solution for the KdV equation is stable using a spectral method where we have modified the U_{xxx} term causing instabilities.

The method of integrating factors can also be used to remove the instability due to U_{xxx} term (see Trefethen).

12.3.3 Interacting Solitons

The matlab code is **InteractingSoliton.m**.

When 2 solitons travelling at different speeds collide their waveform maintains same size, shape and speed but the smaller (and slower) soliton is backward shifted and the taller (and faster) soliton is forward shifted.

To show this feature of solitons we begin with intial conditions of two solitons with speeds of $V = 2A/3$ and $V = A/3$:

$$U(x,0) = f(x) = A\text{sech}^2\left(\sqrt{\frac{A}{12}}(x - \frac{3\pi}{2})\right) + 2A\text{sech}^2\left(\sqrt{\frac{2A}{12}}(x - \frac{\pi}{2})\right)$$

where $A = 100$.

Figure 12.5 shows the numerical solution for the KdV equation for 2 interacting solitons using the modified spectral method. In figure 12.5(a) we plot the initial conditions and final solution after one period. We see that after the interaction the smaller soliton is backward shifted and taller soliton forward shifted in time.

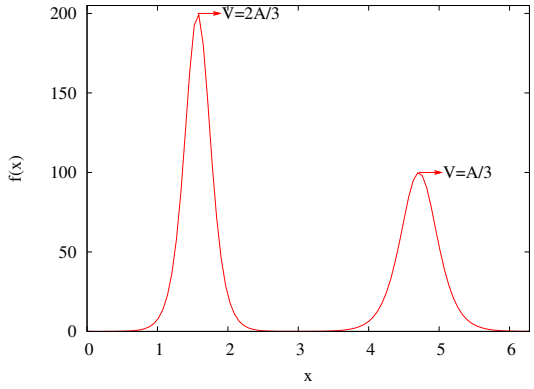


Figure 12.4: The initial conditions $U(x,0) = f(x)$ is plotted to show the 2 solitions and their speeds

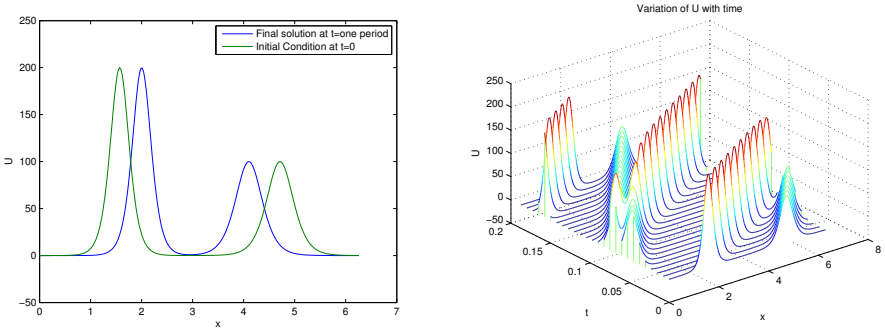


Figure 12.5: Initial conditions and final solution after one period in (a) and solution for nonlinear KdV equation for two interacting solitons in (b)