Numerical Methods for Physicists

Luciano Rezzolla

- Root Finding (1)
- Linear Algebra (2)
- Interpolation and Extrapolation of functions (1)
- Integration of Functions (1)
- Random Numbers and Montecarlo Techniques (1)
- Solution of Ordinary Differential Equations (1)
- Solution of Partial Differential Equations
  - Initial Value Problems
    * Hyperbolic Equations (3)
    * Parabolic Equations (1)
  - Boundary Value Problems
    * Elliptic Equations (2)
- Fourier Transforms (1)
- Data Analysis Techniques (1)

The course is developed over 15 lectures and aims at providing the student with many of the "tools" frequently used in the solution of physical problems. The course is meant to be an applied course, in which the actual programming is a key feature. For this reason, each lecture will have one or more exercises involving the solution of a specific physical problem via the use of numerical codes implementing the techniques discussed in the lecture. The codes can be written in any of the following languages: fortran77, fortran90, fortran95, C, C++, however no specific computational knowledge is necessary. Each exercise must be completed before the subsequent lecture in the course starts. The schedule of the lectures can be found at http://www.sissa.it/ap/schedule.html

Textbooks

- Numerical Recipes, W. H. Press et al., Cambridge Univ. Press, 1992;

1The number in brackets refers to the number of lectures covering the subject. Note: each lecture is given over two hours
Before entering in the details of root finding, it is important to discuss a number of general but fundamental concepts in Numerical Analysis.

Computers store numbers with finite precision: i.e. with fixed number of bits or bytes.

Numbers are also distinguished in terms of the data type:

- **INTEGERS**: representation is exact within the range of representation; arithmetics of integers is exact (modulus the remainder in a division).

- **FLOATING-POINT**: a number represented as

  \[ s \times M \times B^{e-E} \]

  - \( s \): sign
  - \( M \): mantissa
  - \( B \): base
  - \( e-E \): bias in the exponent
The smallest no depends on a number of things: machine type, computer used etc.

In general the smallest and largest numbers on a 16-bit machine are:

Single precision:
- $10^{-38} - 10^{+38}$
- $10^{-108} - 10^{+308}$

Double precision:
- $10^{-32} - 10^{+38}$
- $10^{-308} - 10^{+308}$

$\epsilon_M$ is a measure of relative precision and not of absolute magnitude.
Now some definitions:

\[ E_m: \text{Machine accuracy: } \text{the smallest floating point}\]

\[ (fp) \text{ number which, when }\]

\[ \text{edited to } 1.0 \text{ produces a }\]

\[ \text{result different from } 1.0\]

\[ 1+1=2(1+E_m)\]

\[ (\times) \]

**NOTE:** this is not the smallest \( fp \) number that can be represented on a machine and which depends on how many bits there are in the exponent. Rather, \( E_m \) depends on how many bits there are in the mantissa. \( \star \) can be \( \approx 10^{-8} \) (7 digits) or \( 10^{-17} \)

\[ E_r: \text{ROUND OFF ERROR: } \text{is the error introduced}\]

\[ \text{in an operation between two}\]

\[ (fp) \text{ nos. It reflects the error in the least significant fig. of the mantissa.}\]

If there are \( N \) operations one would expect that

\[ E_r \approx N E_m \]

but in most cases roundoff errors compensate themselves at least in part so that

\[ E_r \approx \sqrt{N} E_m \]

in general

Obviously \( E_r = E_m \) if \( N = 1 \)
\[ (\text{numerical solution}) = (\text{exact solution}) + |\varepsilon_t| \]

Note: I have used numerical and not computed because \( \varepsilon_t \) has nothing to do with the use of computers.

\[ f(x) = f(x_0) + f'(x_0)(x-x_0) + O(\Delta x^2) \]

\( \varepsilon_t \) is totally under the programmer's control and should be monitored to verify that the numerical implementation is a faithful representation of the math.

**STABLE & UNSTABLE METHODS:**

It is often the case that the errors introduced in an operation with floating-point numbers are not just round-off. There are algorithms that although apparently innocuous introduce errors that are then magnified and grow unbounded until they destroy the solution. This topic will be further discussed in the solution of hyperbolic PDEs.
\[ f(x) = ax + b \]
Root finding problem \( \Leftrightarrow \) finding the root of
\[ f(x) = 0 \]
where
\[ x = \text{ vector of } N \text{ independent variables } \]
\[ f = \text{ } M \text{ functions } \]

The problem is much simpler if \[ N = 1 = M \]

In this case, the "bracketing" technique represents an obvious approach.

(*) Except for linear problems, root finding proceeds by iteration: ie starting from some suitable trial solution, use an algorithm that will improve it till a converge criterion is satisfied.
"There are no good general methods for solving systems of more than one nonlinear equations."
Rule of thumb: "before you look for a root, make sure you will find at least one!"

Use all possible information to identify the neighborhood of a root.

More on Bracketing

Let \( f = f(x) \), \( x \in \mathbb{R} \)

\( x_0 \) is bracketed in \( [x_1, x_2] \) iff

\[ f(x_1) f(x_2) < 0 \]

Notes: \( f(x_1) f(x_2) < 0 \) is not a necessary, nor a sufficient condition for the existence of a root. \( x_0 \in [x_1, x_2] \) \( x_0 \) root \( \Rightarrow \) \( f(x_1) f(x_2) < 0 \)

2 roots but \( f(x_1) f(x_2) > 0 \)

Two roots

\( f(x_1) f(x_2) = 0 \)

\( x_1 = x_2 \)
b) \( f(x_1)f(x_2) < 0 \) \( \Rightarrow \) there is a root

\[ f(x) = \frac{1}{x - x_0} \]

**Suggestion**: always try to plot the function!

**How do we bracket?**

\[ \Delta x_2 = \frac{(b - a)}{N} \]

\[ x_1 = a \]

\[ f_1 = f(a) \]

\[ x_2 = x_1 + \Delta x \]

\[ f_2 = f(x_2) \]

\[ f_1 f_2 < 0 \]

\[ n = n + 1 \]

\[ x_0, n = x_1 \]

\[ x_{b, n} = x_2 \]

Increment a counter and save interval.
If $f \epsilon / \epsilon_0 = 10^{-7}$

$N \sim 23$
The simple idea of bracketing is behind the simplest root finding algorithm: **Bisection**

Very simple idea: given an interval in which the root is bracketed, \([x_1, x_2]\), evaluate the function at the midpoint and use it to replace any part of the interval in which the function has the same sign.

**Pros:** bisection cannot fail if the root is bracketed.

**Cons:** Not very efficient algorithm, i.e. the rate of convergence is linear.

Let \(e_n\) be deviation from \(x_0\) at each iteration

\[
e_n = \frac{1}{2} e_{n-1} = 2^{-n} e_0.
\]

If we want a final tolerance \(\varepsilon_f\) we will need a number of iterations

\[
N = -\log_2 \left( \frac{\varepsilon_f}{e_0} \right) = - \log \left( \frac{\varepsilon_f/e_0}{\log_2 e} \right)
\]
In general, a method is said to have a rate of convergence of order $m$ if

$$
\epsilon_{n+1} = \text{const.} \times (\epsilon_n)^m
$$
Q: What is a reasonable $E_f$?

$E_f \approx E_{\text{in}} \left( \frac{1}{(x_1 + x_2)} \right)$

the smaller the bracketing range the better the tolerance

machine accuracy

Also $E_f = \frac{x_2 - x_1}{x_0}$

measure how close you are to $x_0$

Q: Why is not $E_f \approx 8$?

A: By def $E_f = E_{\text{in}}$ will not work. As $E_f \approx E_{\text{in}}$ you start picking up the round-off error.

Let $f(x_1)f(x_2) < 0$

$f(x_1) < 0$?

YES ($f(x_2) > 0$)

NO ($f(x_2) < 0$)

$x = x_2$

$x = x_1$

$\Delta x = x_2 - x_1$

$\Delta x = \Delta x/2$

$x_m = x + \Delta x$

$f(x_m)$

YES

update $x_m$

$x = x_m$

NO

$\Delta x < \varepsilon$? or $f(x_m) = 0$
To improve on convergence rate one can use one of these alternative methods

I) SECANT METHOD

1) Let $x_0$ be a root, $x_0 \in [x_1, x_2]$

a) approximate the function as a line through $x_1$ and $x_2$

$$\frac{f(x_0) - f(x_1)}{x_0 - x_1} = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

b) take $f(x_0) = 0$ as the best approximation to the root, i.e.

$$x_0 = x_1 + (x_2 - x_1) \frac{f(x_1)}{f(x_1) - f(x_2)}$$

2) Iterate

The secant method is very fast with convergence rate

$$\lim_{n \to \infty} \frac{|\varepsilon_{n+1}|}{|\varepsilon_n|} \leq \text{const} \cdot 1.681$$
The secant method has a potential drawback since it retains always one of the two initial bracketing limits (i.e., $x_1$ or $x_2$). This procedure is dangerous as it does not check whether the root is bracketed!
The search is made in three intervals

<table>
<thead>
<tr>
<th>Iteration no</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>X₄</th>
<th>X₃</th>
<th>X₂</th>
<th>bracketed</th>
<th>X₃</th>
<th>X₁</th>
<th>X₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>X₁</td>
<td>X₂</td>
<td>X₃</td>
<td>bracketed</td>
<td>X₃</td>
<td>X₁</td>
<td>X₄</td>
</tr>
<tr>
<td>N</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>X₁</td>
<td>X₃</td>
<td>X₄</td>
<td>not bracketed</td>
<td>X₃</td>
<td>X₁</td>
<td>X₄</td>
</tr>
</tbody>
</table>
II) **FALSE POSITION**

solves this problem by making sure the root is always bracketed!

Q: How?

check the condition on the cross product

a) Let \( x_0 \) be a root, \( x_0 \in [x_1, x_2] \)
   Let \( f(x_1) < 0 \), \( f(x_2) > 0 \)

b) evaluate \( x_0 \) through a secant

\[
x_0 = x_1 + (x_2 - x_1) \frac{f(x_1)}{f(x_1) - f(x_2)}
\]

c) Evaluate \( f_0 = f(x_0) \)

\[ f(x_2) > 0 \text{ by assumption} \]

If \( f_0 < 0 \), then \( f_0 \cdot f_2 < 0 \implies \text{retain } x_2 \)

If \( f_0 > 0 \), then \( f_0 \cdot f_1 < 0 \implies \text{retain } x_1 \)

In other words: if they have the same sign the secant is not useful.
PROS: False position guarantees bracketing; this condition might be necessary for other reasons.

CONS: Less efficient; it is superlinear but the exact order is not possible to determine.

Q: Are there bad situations for BISEC and FPS?

very slow convergence!
$x$ is a quadratic function of $y$.

Estimate of the root.
We have seen that the secant and the false position methods generally converge faster than bisection which has a linear convergence rate. On the other hand, bisection cannot fail!

There is a method that provides the "certainty" of bisection with a superlinear convergence. This is BRENT's method (Van Vlijngaerder-Dekker-Brent).

The superlinear convergence rate is reached by using an inverse quadratic interpolation among three points and estimates the root as the location at which the interpolating function vanishes.

Let the 3 points be

\[(a, f(a) = f_a); (b, f(b) = f_b); (c, f(c) = f_c)\]

then the inverse quadratic formula gives

\[x = \frac{(y-f_a)(y-f_b)c + (y-f_b)(y-f_c)b}{(f_c-f_a)(f_c-f_b)} + \frac{(y-f_c)(y-f_a)b}{(f_b-f_c)(f_b-f_a)}\]

Setting \(y = 0\)

\[x = b + \frac{p}{q}\]
where

\[ P = S \left[ T (R-T)(C-6) - (1-R)(6-a) \right] \]
\[ Q = (T-1)(R-1)(S-1) \]

end

\[ R = \frac{f_b}{f_c} ; \quad S = \frac{f_b}{f_a} ; \quad T = \frac{f_a}{f_c} \]

Of course it may happen that the correction is not small (e.g. \( Q \gg 1 \)) and the guessed root is out of the bracketing values. In this case, Brent's method takes a bisection step.

* Brent's Method is a recommended general use root finder*
A very celebrated root-finding algorithm is the **NEWTON–RAPHSON** method.

Its power (as well as its major shortcomings) come from the use of function \( f(x) \) and function derivative \( f'(x) \) at a given point \( x \).

Graffically, given a point \( x \) and a function there evaluated \( f(x) \), draw the tangent to the function at \( x \) and extend it to the intersection with the \( x \)-axis. The intersection will represent the new best estimate.

Taylor expand \( f \) around \( x \):

\[
f(x_0) = f(x) - f'(x)(x-x_0) + O((x-x_0)^2)
\]

\[
f(x_0) = 0 \quad \Rightarrow \quad x - x_0 = -\frac{f(x)}{f'(x)} \quad \Rightarrow
\]

\[
x_0 = x - \frac{f(x)}{f'(x)}
\]

Current best estimate → correction
Why is NR so powerful? Which is the root
Consider a point $x$ and Taylor expand in its neighborhood

$$f(x+\epsilon) = f(x) + \epsilon f'(x) + \frac{\epsilon^2}{2} f''(x) + O(\epsilon^3)$$

$$f'(x+\epsilon) = f'(x) + \epsilon f''(x) + O(\epsilon^2)$$

NR formula says

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \Rightarrow \epsilon_{i+1} = \epsilon_i - \frac{f(x_i)}{f'(x_i)}$$

$x$ is a root

$$\epsilon_{i+1} = \epsilon_i - \frac{f(x_i) + \epsilon_i f'(x_i) + \epsilon_i^2/2 f''(x)}{f'(x_i) + \epsilon_i f''(x)}$$

$$= \left[ \epsilon_i f'(x) + \epsilon_i^2 f''(x) - f(x) - \epsilon_i f'(x) - \frac{\epsilon_i^2 f''(x)}{2} \right] \cdot \frac{1}{f(x) + \epsilon_i f''(x)}$$

$$= \frac{\epsilon_i^2 f''(x)}{2} \cdot \frac{1}{f'(x) \left( 1 + \epsilon_i \frac{f''(x)}{f'(x)} \right)}$$

$$\approx \frac{\epsilon_i^2 f''(x)}{2} \cdot \frac{1}{f'(x)} \cdot \left( 1 - \epsilon_i \frac{f''(x)}{f'(x)} \right)$$

$$\epsilon_{i+1} \approx \frac{\epsilon_i^2 f''(x)}{2} f'(x) + O(\epsilon_i^3)$$
NR is quadratically convergent!

The no of significant figures doubles at each step. Of course one needs to know \( f'(x) \). If no analytic expression is available, then

\[
f'(x) \approx \frac{f(x+\delta x) - f(x)}{\delta x} + o(\delta x)
\]

but in this case the convergence rate is reduced to \( \leq \sqrt{2} \).

NR has also the problem that since it does not use any bracketing, it might well get completely lost.

No convergence

NR should be used together with a bisection algorithm!
\[ f(x) = f(x_0) + f'(x_0)(x - x_0) = 0 \]

\[ x = \frac{f(x_0)}{f'(x_0)} x_0 + x_0 = x_0 + \frac{f(x)}{f'(x_0)} \]

\[ \delta x_k = -f_i(x_0) \]

\[ \frac{d}{dx_k} (x_0 - x_k) = -f_i(x_0) = \delta_{ik} (x_0 - x_k) \]

\[ x_k = \left( f_i(x_0) + \delta_{ik} x_0 \right) / \delta_{ik} \]

\[ = x_0 + \frac{f_i(x_0)}{\delta_{ik}} \]
MULTIDIMENSIONS

Root finding in multidimensions and for nonlinear equations is considerably more difficult! There is NO GENERAL METHOD.

NR's extension is powerful but might fail spectacularly

\[ f_i(x_1, x_2, \ldots, x_n) = 0 \]


Set \( f(\bar{x}) = 0 \) be the set of eqs to solve

set of initial guesses = \( (\bar{x}_0 - \bar{x}_k) \)

Expand around \( \bar{x}_0 \)

\[ f_i(\bar{x}) = f_i(\bar{x}_0) + J_{ik}(\bar{x}_0) \Delta x_k + O(\Delta x_k^2) \]

where

\[ J_{ik}(\bar{x}_0) = \frac{\partial f_i}{\partial x_k} : Jacobian matrix \]

Imposing \( f(\bar{x}) = 0 \Rightarrow f_i(\bar{x}) = 0 \)

\[ J_{ik} \Delta x_k = -f_i(\bar{x}) \]

so that \((\bar{x}_0)\) can be obtained through a matrix inversion of \( J_{ik} \Delta x_k \)

\[ \bar{x}_k = \bar{x}_0 + \frac{f_i(\bar{x}_0)}{J_{ik}} \]
If $J_{ik}$ is not available analytically, then

$$J_{ik} \approx \frac{f_i(x_k + \delta x_k) - f_i(x_k)}{\delta x_k}$$
The solution of nonlinear elliptic equations might need to go through a discretization procedure which at the end produces a system of linear algebraic equations.

In general, one needs to solve $N$ equations in $N$ unknowns of the type:

\[
\sum_{j=1}^{N} a_{ij} x_j = b_i \quad \text{for } i = 1, \ldots, N
\]

or, equally,

\[
A \cdot x = b
\]

Cramer's rule, which is fine for analytic cases, is not good as the number of operations grows rapidly with $N$ and is almost as $N!$.

We need efficient methods, where the efficiency is defined as:

1) How fast the method is (i.e., how many operations)

2) How accurate the solution is (check if the round-off error is growing excessively)
Given a system of eqs

\[
\begin{align*}
q_{11}x_1 + q_{12}x_2 + & \cdots + q_{1N}x_N = b_1 \\
q_{21}x_1 + q_{22}x_2 + & \cdots + q_{2N}x_N = b_2 \\
& \quad \vdots \\
q_{N1}x_1 + q_{N2}x_2 + & \cdots + q_{NN}x_N = b_N \\
\end{align*}
\]

or

\[
\sum_{j=1}^{N} a_{ij}x_j = b_i \quad i = 1, \ldots, N
\]

or

\[
A \cdot x = b
\]

There is a well known procedure in the case in which \( |A| \neq 0 \) (i.e. the coefficients matrix has non-zero determinant).

In this case: Cramer's Rule

\[
x_j = \frac{\sum_{i=1}^{N} c_{ij} b_i}{|A|} = \frac{c_{j1}b_1 + c_{j2}b_2 + \cdots + c_{jn}b_n}{|A|}
\]

where
C_{ij} is the cofactor of the a_{ij} element

C_{ij} = (-1)^{i+j} M_{ij}

minor: determinant of A_{ij} when the i-th row and the j-th column are removed.

If we refer to the system of eqs (1)-(3)

\[ A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \]

\[ x_1 = \frac{\sum i \cdot C_{i1} b_i}{|A|} = \frac{(C_{11} b_1 + C_{21} b_2 + C_{31} b_3)}{|A|} \]

\[ |A| = a_{11} (a_{22} a_{33} - a_{23} a_{32}) - a_{12} (a_{21} a_{33} - a_{31} a_{23}) + a_{13} (a_{21} a_{32} - a_{31} a_{22}) \]

\[ C_{11} = (-1)^{12} M_{11} = a_{22} a_{33} - a_{23} a_{32} \]

\[ C_{21} = (-1)^{2} M_{21} = -a_{12} a_{33} + a_{13} a_{32} \]
This is all very nice but very expensive too!

The no of operations scales roughly like

\[
(\text{No operations}) \sim (N+1)!
\]

If \( N = 4 \) \( (N+1)! \sim 120 \)

We clearly need more efficient methods.

When solving a system of linear algebraic eqs we need to look at the matrix of coefficients.

In this case there are two possible choices:
a) filled but not large (few elements are zero)

b) sparse but large $\sim 10^4$ (few elements are not zero)

In case a) we can use **DIRECT** methods or Gaussian elimination methods.

In case b) we can use **ITERATIVE** methods (we will look at these when discussing PDEs).

Direct methods are devised by Gauss to eliminate one by one the unknowns. It is based on an analytic strategy.

It is often convenient to calculate the residual of the present solution of the linear algebraic eq.

$$ r = b - A \cdot x $$

Note that the residual needs not to be zero but just smaller than the truncation error:

$$ r < \epsilon_T $$
Example

Let \( x_t \) be the exact solution \((\text{true})\)

\[
A x_t = b \iff \begin{cases}
A x_t - b = 0 \\
A x_c - b = 0
\end{cases}
\]

then

\[
\varepsilon = A (x_t - x_c) \Rightarrow x_t - x_c = A^{-1} \varepsilon
\]

Note that if the matrix is **ill-conditioned** (ie one element is much larger than the others) the residual might not be a good measure of the accuracy.

Let's have a closer look at the Gauss elimination with an example of a 3 eqs system! ie \( A \) is a 3x3 matrix \( b \) is \((3x1)\):

\[
\begin{align*}
\alpha_{11} x_1 + \alpha_{12} x_2 + \alpha_{13} x_3 &= b_1 \quad (1) \\
\alpha_{21} x_1 + \alpha_{22} x_2 + \alpha_{23} x_3 &= b_2 \quad (2) \\
\alpha_{31} x_1 + \alpha_{32} x_2 + \alpha_{33} x_3 &= b_3 \quad (3)
\end{align*}
\]

The direct method I will discuss is based on two distinct steps:

1) **Gauss elimination**

2) **Back-substitution**
Let's divide eq (1) by \( a_{11} \) i.e.

\[
\frac{x_1 + \frac{a_{12}}{a_{11}} x_2 + \frac{a_{13}}{a_{11}} x_3 = \frac{b_1}{a_{11}}}{a_{11}}
\]

Then

1. \( a_{21}(4) = b_2 - a_{21} \frac{b_1}{a_{11}} \equiv b_2^{(1)} \) (5)
2. \( a_{31}(4) = b_3 - a_{31} \frac{b_1}{a_{11}} \equiv b_3^{(1)} \) (6)

Eqs (5) and (6) do not contain \( x_1 \).

We have eliminated \( x_1 \).

(5) and (6) \( \Leftrightarrow \)

\[
\begin{align*}
\text{\( a_{12}^{(1)} \) } x_2 + \text{\( a_{23}^{(1)} \) } x_3 &= \text{\( b_2^{(1)} \) } \\
\text{\( a_{32}^{(1)} \) } x_2 + \text{\( a_{33}^{(1)} \) } x_3 &= \text{\( b_3^{(1)} \) }
\end{align*}
\]

Where

\[
\begin{align*}
\text{\( a_{22}^{(1)} \) } &= a_{22} - a_{21} \frac{a_{12}}{a_{11}}; \quad \text{\( a_{23}^{(1)} \) } &= a_{23} - a_{21} \frac{a_{13}}{a_{11}} \\
\text{\( a_{32}^{(1)} \) } &= a_{32} - a_{31} \frac{a_{12}}{a_{11}}; \quad \text{\( a_{33}^{(1)} \) } &= a_{33} - a_{31} \frac{a_{13}}{a_{11}}
\end{align*}
\]

I can now proceed in a similar way if \( a_{22}^{(1)} \neq 0 \) (if not I just change row or column).
\[
\text{dividing (5') by } a_{22}^{(1)} \text{ I get}
\]
\[
\Rightarrow x_2 + \frac{a_{23}^{(1)}}{a_{22}^{(1)}} x_3 = \frac{b_2^{(1)}}{a_{22}^{(1)}} \quad (7)
\]
\[
(6') - a_{32}^{(1)} (7) = b_3^{(1)} - \frac{b_2^{(1)}}{a_{22}^{(1)}} \equiv b_3^{(2)} \quad (8)
\]

If I define
\[
a_{33}^{(2)} = a_{33}^{(1)} - a_{32}^{(1)} \frac{a_{23}^{(1)}}{a_{22}^{(1)}}
\]

then (8) \Rightarrow
\[
a_{33}^{(2)} x_3 = b_3^{(2)} \quad (9)
\]

Once \( x_3 \) is known, we can calculate \( x_2 \) and \( x_1 \) through **BACK-SUBSTITUTION**

If we look at eqs (4), (7), (9) we have a set of eqs with matrix of coefficients

\[
\begin{pmatrix}
1 & \frac{a_{12}}{a_{11}} & \frac{a_{12}}{a_{11}} \\
0 & 1 & a_{23}^{(1)} \\
0 & 0 & 1
\end{pmatrix}
\]

This is an **UPPER TRIANGULAR MATRIX** or **U-matrix**

Building the U-matrix represents the operation of the **Gaussian elimination**
The generic algorithm is then:

\[ A \cdot x = b \Rightarrow A^{(n)} \cdot x = b^{(n)} \]

\[ a_{11} a_{12} \ldots a_{1n} \]

\[ a_{21} \]

\[ a_{22} \]

\[ \vdots \]

\[ a_{nn} \]

this row is filled first

second row to be filled

A \[ \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \]

\[ a_{kk} \]

A \[ \begin{pmatrix} \ldots & \cdots & \cdots & \cdots \\ \cdots & \ldots & \cdots & \cdots \\ \cdots & \cdots & \ldots & \cdots \\ \cdots & \cdots & \cdots & a_{nn} \end{pmatrix} \]

last element to be calculated

\[ a_{ij} = a_{ij} - M_{ik} a_{kj} \]

where

\[ M_{ij} = a_{ik} / a_{kk} \]

\[ i, j = k+1, \ldots, N \]

\[ k = 1, \ldots, N-1 \]

and

\[ b^{(k+1)}_i = b^{(k)}_i - M_{ik} b^{(k)}_k \]

\[ i = k+1, \ldots, N \]

If we call

\[ U = A^{(n)} \]

\[ g = b^{(n)} \]

note that the multiplier does not contain information on the RHS and refers to the previous elimination
Then the transformation has led to the system

\[ U \cdot x = g \]

\[ A^{(n)} \cdot x = b^{(n)} \]

\[ A \cdot x = b \]

so that the solution

\[ x_N = \frac{g_N}{U_{NN}} \]

\[ x_k = \frac{1}{U_{kk}} \left( g_k - \sum_{j=k+1}^{N} U_{kj} x_j \right) \]

\[ k = N-1, N-2, \ldots, 1 \]

Because new coefficients are calculated, there is the problem of allocating new memory for them.

\( \text{Q: Where should I store the multipliers?} \)

\( \text{A: In the lower part of the upper triangular matrix and call this a } L \text{-matrix} \)

\( \text{ie those of the matrix } U \)
Gaussian elimination with back substitution leads the way to another approach in the solution of linear algebraic eqns.

**LU DECOMPOSITION**

\[ A \cdot x = b \]  

Suppose we are able to decompose \( A \) as

\[ A = L \cdot U \]

\[ A \cdot x = (L \cdot U) \cdot x = L \cdot (U \cdot x) = b \]

The initial problem (*) is then recast as

\[ \begin{align*}
\text{(*) } \iff \\
L \cdot y &= b \\
U \cdot x &= y
\end{align*} \]

In other words, we have broken up the linear system in two successive ones.

**Q:** Where's the advantage?

**A:** The advantage is apparent when the decomposition of the initial matrix \( A \) is in two triangular matrices. In that case, in fact, algorithms similar to the ones of back substitution can be used.
let's go back to the previous matrix $A$

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} = \begin{pmatrix} d_{11} & 0 & 0 \\ d_{21} & d_{22} & 0 \\ d_{31} & d_{32} & d_{33} \end{pmatrix} = \begin{pmatrix} B_{11} & B_{12} & B_{13} \\ 0 & B_{22} & B_{23} \\ 0 & 0 & B_{33} \end{pmatrix}$$

$L \cdot y = b \iff$

$$\begin{pmatrix} d_{11} & 0 & 0 \\ d_{21} & d_{22} & 0 \\ d_{31} & d_{32} & d_{33} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$\Rightarrow$

$$y_1 = b_1 / d_{11}$$

$$y_2 = \frac{1}{d_{22}} \left( b_2 - y_1 \cdot d_{21} \right) = \frac{1}{d_{22}} \left( b_2 - b_1 \cdot \frac{d_{21}}{d_{11}} \right)$$

$$y_3 = \frac{1}{d_{33}} \left( b_3 - y_2 \cdot d_{32} - y_1 \cdot d_{31} \right)$$

In general:

$$y_i = \frac{b_i}{d_{ii}} \quad \left[ \frac{y_j}{d_{ii}} \right]_{j=1}^{i-1} = \frac{b_i - \sum_{j=1}^{i-1} \alpha_{ij} y_j}{\alpha_{ii}}$$

Calculate first $y_1$, then $y_2$... and finally $y_n$. 


Similarly, once I know all the $y$s
I can use the previous algorithm

$$U \cdot X = Y$$

$$\begin{pmatrix}
\beta_{11} & \beta_{12} & \beta_{13} \\
0 & \beta_{22} & \beta_{23} \\
0 & 0 & \beta_{33}
\end{pmatrix}
\begin{pmatrix}
X_1 \\
X_2 \\
X_3
\end{pmatrix}
= 
\begin{pmatrix}
Y_1 \\
Y_2 \\
Y_3
\end{pmatrix}$$

$$X_3 = \frac{Y_3}{\beta_{33}}$$

$$X_2 = \frac{1}{\beta_{22}} \left( Y_2 - X_3 \beta_{23} \right)
= \frac{1}{\beta_{22}} \left( Y_2 - \frac{Y_3 \beta_{23}}{\beta_{33}} \right)$$

$$X_1 = \frac{1}{\beta_{11}} \left( Y_1 - X_2 \beta_{12} - X_3 \beta_{13} \right)$$

or in general

$$X_N = \frac{Y_N}{\beta_{NN}}$$

$$X_i = \frac{1}{\beta_{ii}} \left[ Y_i - \sum_{j=i+1}^{N} \beta_{ij} X_j \right] \quad i = N-1, N-2, \ldots 1$$

First calculate $X_N$, then $X_{N-1}$ and finally $X_1$. 
Q: How do we do the LU decomposition in practice?

\[ L \cdot U = A \]

\[
\begin{pmatrix}
\alpha_{11} & 0 & 0 & \cdots & 0 \\
\alpha_{21} & \alpha_{22} & 0 & \cdots & 0 \\
\alpha_{31} & \alpha_{32} & \alpha_{33} & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\alpha_{n1} & \cdots & \cdots & \cdots & \alpha_{nn}
\end{pmatrix}
\begin{pmatrix}
\beta_{11} & \beta_{12} & \beta_{13} & \cdots & \beta_{1n} \\
0 & \beta_{22} & \beta_{23} & \cdots & \beta_{2n} \\
0 & 0 & \beta_{33} & \cdots & \beta_{3n} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & \beta_{nn}
\end{pmatrix}
= 
\begin{pmatrix}
\alpha_{11} & \alpha_{12} & \alpha_{13} & \cdots & \alpha_{1n} \\
\alpha_{21} & \alpha_{22} & \alpha_{23} & \cdots & \alpha_{2n} \\
\alpha_{31} & \alpha_{32} & \alpha_{33} & \cdots & \alpha_{3n} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\alpha_{n1} & \cdots & \cdots & \cdots & \alpha_{nn}
\end{pmatrix}
\]

\[ \iff \]

Set of eqs of the type

\[ \alpha_{ij} \beta_{jj} + \ldots = a_{ij} \]

\[ \text{the no of terms depends in the values of } ij \]

In particular:

\[ i < j \quad \alpha_{ij} \beta_{jj} + \alpha_{i2} \beta_{2j} + \ldots + a_{ij} \beta_{ij} = a_{ij} \]

\[ i = j \quad \alpha_{ii} \beta_{jj} + \ldots = a_{ii} = a_{ii} \] (*)

\[ i > j \quad \alpha_{ij} \beta_{jj} + \alpha_{i2} \beta_{2j} + \ldots + a_{ij} \beta_{jj} = a_{ij} \]

\[
\begin{pmatrix}
\alpha_{11} & 0 & 0 \\
\alpha_{21} & \alpha_{22} & 0 \\
\alpha_{31} & \alpha_{32} & \alpha_{33}
\end{pmatrix}
\begin{pmatrix}
\beta_{11} & \beta_{12} & \beta_{13} \\
0 & \beta_{22} & \beta_{23} \\
0 & 0 & \beta_{33}
\end{pmatrix}
\]

\[
\begin{pmatrix}
\alpha_{11} & \alpha_{12} & \alpha_{13} \\
\alpha_{21} & \alpha_{22} & \alpha_{23} \\
\alpha_{31} & \alpha_{32} & \alpha_{33}
\end{pmatrix}
\begin{pmatrix}
\beta_{11} & \beta_{12} & \beta_{13} \\
0 & \beta_{22} & \beta_{23} \\
0 & 0 & \beta_{33}
\end{pmatrix}
\]
represent $N^2$ eqs in $N^2 + N$ unknowns because of the diagonal terms $\alpha_{ii}$ and $\beta_{ii}$ more unknowns than eqs.

Since the system is underdetermined we are free to choose $N$ of the unknowns arbitrarily and solve for the others.

A possible choice is $\alpha_{ii} = 1$, $i = 1, \ldots, N$.

With this choice we can then use CROUT's ALGORITHM

\[ \alpha_{ii} = 1 \Rightarrow \beta_{ii} = \alpha_{ii} \]

choose $j = 2, 3, \ldots, N$ then

\[ \begin{align*}
\beta_{ij} &= \alpha_{ij} - \sum_k \alpha_{ik} \beta_{kj} \\
(\text{where } i &= 1 \Rightarrow \beta_{ij} = \alpha_{ij}) \\
\alpha_{ij} &= \frac{1}{\beta_{ij}} \left( \alpha_{ij} - \sum_k \alpha_{ik} \beta_{kj} \right) \\
(\text{for } i &= j + 1, j + 2, \ldots, N)
\end{align*} \]

Note that in (i) and (ii) all the $\alpha$s and $\beta$s on the RHS are already known because computed in a previous loop.
Example \( \beta_{ij} = a_{ij} - \frac{i-1}{i} \alpha_{ik} \beta_{kj} \)

- \( i=1, j=1 \)
  \[ \beta_{11} = a_{11} \]

- \( i=1, j=2 \)
  \[ \beta_{12} = a_{12} \]
  \[ \alpha_{12} = \frac{1}{\beta_{11}} (a_{12} - a_{11} \beta_{12}) \]

- \( i=2, j=1 \)
  \[ \beta_{21} = a_{21} - (\alpha_{21} \beta_{11}) \]
  \[ \alpha_{21} = \frac{1}{\beta_{11}} (a_{21} - \alpha_{11} \beta_{12}) \]

- \( i=2, j=2 \)
  \[ \beta_{22} = a_{22} - (\alpha_{22} \beta_{12}) \]
  \[ \alpha_{22} = \frac{1}{\beta_{22}} (a_{22} - \alpha_{22} \beta_{12}) \]

- \( i=3, j=1 \)
  \[ \beta_{31} = a_{31} - (\alpha_{31} \beta_{11} + \alpha_{32} \beta_{21}) \]
  \[ \alpha_{31} = \frac{1}{\beta_{31}} (a_{31} - \alpha_{31} \beta_{21}) \]

- \( i=3, j=2 \)
  \[ \beta_{32} = a_{32} - (\alpha_{31} \beta_{12} + \alpha_{32} \beta_{22}) \]
The decomposition is said to be "in place" in the sense that as the $a_i$s and $f_i$s are calculated, the $a_i$s can be removed so that at the end the $a_i$s and $f_i$s fill a single matrix of dimension $N \times N$ replacing $A$.

$$A \rightarrow L \cdot U \rightarrow \begin{pmatrix}
    a_{11} & a_{12} & \cdots & a_{1N} \\
    a_{21} & a_{22} & \cdots & a_{2N} \\
    a_{31} & a_{32} & a_{33} & \cdots & a_{3N} \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    a_{N1} & a_{N2} & \cdots & a_{NN} & a_{NN}
\end{pmatrix}$$
Q: What's the computational cost? 

C: Cost \( \approx \frac{N^3}{3} \approx 300 \) if \( N=10 \)

Q: Give \( Ax = b \) why not solve \( x = A^{-1}b \)

The problem is that matrix inversion is much more expensive since it involves solving \( AA^{-1} = I \)

As a comparison, Cramer's rule computational cost is \( (N+1)! \)

\( N=10 \)

\( \approx 3.9 \times 10^7 \)

In case of ill-conditioned matrices there are "curing" techniques which are based on pivoting or equilibration and which try to re-normalize the elements of the matrix.
The basic problem is well-known.

Given the values of a function \( f = f(x) \) in a number of given points \( (x_1, x_2, \ldots, x_n) \rightarrow f(x_1), f(x_2), \ldots, f(x_n) \) find

- \( f(x) \) with \( x \in [x_1, x_n] \): INTERPOLATION
- \( f(x) \) with \( x \notin [x_1, x_n] \): EXTRAPOLATION

Both interpolation and extrapolation must model the function in between or beyond the known points in terms of MODEL functions; i.e., functions that are sufficiently general so as to be able to approximate a large class of functions.
Good and widely used model functions are:

- Polynomials
- Rational functions (better for functions with poles)
- Trigonometric functions

Although one might choose the right modelling function, there might still be intrinsic problems related to pathological functions. E.g.

\[ f(x) = 3x^2 + \frac{1}{\ln[(11-x)^2]} + 1 \]

This is basically a parabola but have a closer look for \( x \approx 11 \)

In other words, interpolation and extrapolation is well posed problem if the interpolated function is smooth among the tabulated points.

When this condition is not maintained, interpolation and extrapolation are not reliable!

It is important to monitor the error made to check anomalies.

Hereafter what we say about interpolation is also valid to extrapolation.
Conceptually, the interpolation process has two stages:

i) find an interpolating function to the data points provided \( x_1, x_2, \ldots, x_N \)

ii) evaluate the function at the target point \( x \)

In practice, however, it's preferable to combine i) and ii) into a single step and use expressions that evaluate \( f(x) \) directly from the \( (x_1, \ldots, x_N) \) and \( f(x_1), \ldots, f(x_N) \).

In other words, one starts from evaluating \( f(x) \) where \( x \) is the nearest neighbouring point and then add successive corrections to improve the estimate. In general this takes \( O(N^2) \) operations.
Interpolation scheme can also be distinguished in:

A) **LOCAL**: the coefficients of the interpolating polynomial are evaluated from the nearest neighbours.

B) **GLOBAL**: the coefficients are determined non-locally (e.g., by using information on all the points).

A) is **more efficient** but might introduce discontinuities in the first or higher derivatives.

B) is **more elaborate and expensive** but provides continuous derivatives $\Rightarrow$ Spline interpolation.

Spline might also be at the origin of instabilities: one must be careful, especially in hyperbolic eqs. (*)
Consider polynomials as interpolating functions.

Q: What is a "good" order of the polynomial if polynomials are used for modelling the function?

A: It depends on the function.

If there are sharp corners (i.e., large high-order derivatives), then low-order polynomials are better.

If the function is smooth, high-order polynomials are better.
If \{x_1, x_2, \ldots, x_N\} pts are available with \{f(x_1), f(x_2), \ldots, f(x_N)\} then the max order for the polynomial is

\[ N_{\text{max}} = N_{\text{pts}} - 1 \]

However, generally \( N = 2 - 3 \) is a good approximation and one should rarely go past \( N = 4 - 5 \).

Let's consider now a method for polynomial interpolation.

Given a set of \( N \) points \( x_1, x_2, \ldots, x_N \) and the corresponding values of the function \( y_1, y_2, \ldots, y_N \) (where \( y_i = f(x_i) \)), then the value of interpolating polynomial of order \( N-1 \) in \( x \) is given by Lagrange's formula

\[
y(x) = \frac{(x-x_2)(x-x_3)\ldots(x-x_N)y_1 + \ldots}{(x_1-x_2)(x_1-x_3)\ldots(x_1-x_N)}
\]

\[
= \frac{(x-x_1)(x-x_2)\ldots(x-x_{N-1})y_N}{(x_1-x_1)(x_1-x_2)\ldots(x_1-x_{N-1})}
\]
N=2, interpolating function is a straight line

\[ y(x) = \frac{(x-x_2)}{(x_1-x_2)} y_1 + \frac{(x-x_1)}{(x_2-x_1)} y_2 \]

\[ \Delta = \frac{1}{\Delta} \left[ (x-x_2) y_1 + (x-x_1) y_2 \right] \]

Taupongye's formula is fine but it is a bit awkward to implement and it does not give an estimate of the error.

Usually one then uses Neville's algorithm which is based on a recursive relation - polynomial of order zero

\[ x_1 \rightarrow y_1 = P_1 \]
\[ x_2 \rightarrow y_2 = P_2 \]
\[ x_3 \rightarrow y_3 = P_3 \]
\[ x_4 \rightarrow y_4 = P_4 \]

where polynomial of order m

\[ P(x) = \frac{(x-x_{i+m}) P_{i(i+1)...(i+m-1)} + (x_i-x) P_{i+1(i+2)...(i+m)}}{x_i-x_{i+m}} \]

\[ m = N_{pts} - 1 \]
In the case of 4 points

\[ P_{234}(x) = \frac{(x-x_4)P_{122} - (x_1-x)P_{234}}{x_1-x_4} \]

where

\[ P_{12}(x) = \frac{(x-x_2)P_{11} - (x_1-x)P_{22}}{x_1-x_2} \quad (i=1, m=2) \]

\[ P_{23} = \frac{(x-x_3)P_{22} - (x_2-x)P_{34}}{x_2-x_3} \quad (i=2, m=3) \]

and where

\[ P_{12}(x) = \frac{(x-x_2)y_1 - (x_1-x)y_2}{x_1-x_2} \]

\[ P_{23}(x) = \frac{(x-x_3)y_2 - (x_2-x)y_3}{x_2-x_3} \quad (i=2, m=2) \]

In other words

\[ \{y_1, y_2, y_3, y_4\} \rightarrow P_{12}, P_{23} \rightarrow P_{123}, P_{234} \rightarrow P(x) \]

\[ \{x_1, x_2, x_3, x_4\} \]
In other words $P_i \ldots (i+m)$ is constructed from $P_{i+1} \ldots (i+m-1)$ and $P_{i+1} \ldots (i+m)$.

Why is NEVILLE'S algorithm convenient?
- easy to implement for any arbitrary order
- keeps track of the differences between parents and daughters $\Rightarrow$ error check:

Define
\[
\begin{cases}
D_{m,i} = P_i \ldots (i+m) - P_{i+1} \ldots (i+m) \\
C_{m,i} = P_i \ldots (i+m) - P_i \ldots (i+m-1)
\end{cases}
\]

then an improvement of (*) is given by
\[
\begin{align*}
D_{m+1,i}(x) &= \frac{(x_{i+m+1} - x)(C_{m+1,i} - D_{m,i})}{x_i - x_{i+m+1}} \\
C_{m+1,i}(x) &= \frac{(x_i - x)(C_{m+1,i} - D_{m,i})}{x_i - x_{i+m+1}}
\end{align*}
\]

polynomials of higher order

So that $P(x) = \sum_i (C_i, D_i)$

In other words, it's trivial to increase the order of the polynomial. This is cumbersome with Lagrange's formula.
An alternative and very powerful interpolation algorithm is offered by the CUBIC SPLINE interpolation.

Given \( y_i = y(x_i) \), \( i = 1, \ldots, N \) set of points set of tabulated values of a function

the linear interpolation formula can be written as

\[
y = Ay_j + By_{j+1}
\]

where \( A = A(x) \), \( B = B(x) \) and

\[
A = \frac{x_{j+1} - x}{x_{j+1} - x_j} \quad \text{and} \quad B = 1 - A = \frac{x - x_j}{x_{j+1} - x_j}
\]

Now, suppose we want write \( y \) as an interpolating cubic polynomial. We then need to add two more coefficients and two more functional values which are cubic in \( x \):

\[
y = Ay_j + By_{j+1} + Cy_{j}'' + Dy_{j+1}'' \quad (*)
\]

\( y_j'', y_{j+1}'' \) : coefficients at this stage

\( C = O(x^3) \), \( D = O(x^3) \), \( A = O(x) \), \( B = O(x) \)
We now impose that $c$ and $D$ are chosen so that

$$y(x=x_j) = y_j$$

$$y(x=x_{j+1}) = y_{j+1}$$

This interpolating function must pass in $y_j$ and $y_{j+1}$, $A(x_j) = 1$, $B(x_{j+1}) = 1$.

A possible choice is then

$$C = \frac{1}{6} \left( A^3 - A \right) \left( x_{j+1} - x_j \right)$$

$$(\star)$$

$$D = \frac{1}{6} \left( B^3 - B \right) \left( x_{j+1} - x_j \right)$$

Note $C(x_j) = 0 = C(x_{j+1})$.

But what are $y''_j$ and $y''_{j+1}$?

Take 1st and 2nd derivative of $(\star)$, then $(\star \star)$

$$\frac{dy}{dx} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{3A^2}{2} (x_{j+1} - x_j) y''_j + \frac{3B^2}{2} (x_{j+1} - x_j) y''_{j+1}$$

$$A y''_j + B y''_{j+1} = 1$$

$$\frac{d^2y}{dx^2} = -6AA' (x_{j+1} - x_j) y''_j + \frac{6BB'}{6} (x_{j+1} - x_j) y''_{j+1}$$

$$= A y''_j + B y''_{j+1}$$
(...*) shows that the coefficients \( y''_j \) and \( y''_{j+1} \) are

\[
y''_j = \frac{d^2 y}{dx^2} \bigg|_{x=x_j} \quad (A=1, \quad B=0)
\]

\[
y''_{j+1} = \frac{d^2 y}{dx^2} \bigg|_{x=x_{j+1}} \quad (A=9, \quad B=1)
\]

Moreover, \( \frac{dy}{dx} \) is continuous across the interval \((x_{j-1}, x_j), (x_j, x_{j+1})\) since it has been obtained analytically through \( dy/dx \). In principle, we can give ANY value to \( y''_j, y''_{j+1} \), but this will not enforce the most important property of a spline: ie continuity of the first derivative!

This is done by imposing that

\[
\left. \frac{dy}{dx} \right|_{x_j^-} = \left. \frac{dy}{dx} \right|_{x_j^+} \quad \text{\( \circ \)}
\]

Using \( x_j, x_{j-1} \)

This then gives a set of \( N-2 \) equations in the \( N \) unknowns \( y''_j \), \( j = 1, \ldots, N \)

\[
\frac{x_j - x_{j-1}}{6} \cdot y''_{j-1} + \frac{x_{j+1} - x_j}{3} \cdot y''_j + \frac{x_{j+1} - x_{j+1}}{6} \cdot y''_{j+1} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}}
\]

(*) point \( j \) requires information from \( j \neq 1 \)
The system of eqs. (1) can be seen as a tridiagonal system of eqs. of N-2 and LU decomposition which can be easily solved (cf. system of linear equations). Of course we need to provide the 2 missing eqs and these provide the bcs for

\[ y''_1 \quad \text{and} \quad y''_N. \]

Possible choices are

a) \( y''_1 = y''_N = 0 \) : Natural spline

b) Use \( y''_1 \) and \( y''_N \) to set specific values of \( y'_1 \) and \( y'_N \)
In other words, the logic behind the cubic spline is the following:

a) I can make a cubic pass through two points \( x_j, x_{j+1} \) but need to provide further information, e.g., through the second derivatives.

\[
(c_{j+1} - c_j) \frac{(x_{j+1} - x_j)^2}{6} = (b^3 - b) \frac{(c_{j+1} - c_j)^2}{6}
\]

b) If I use \( C = \frac{(A^2 - A)(x_{j+1} - x_j)^2}{6} \)

d) Then the actual value of the second derivative is left undetermined. Of course, different values will refer to different cubic curves.

c) I can eliminate this degeneracy by requiring that the first derivatives are continuous. To impose this I need to solve \( N-2 \) algebraic equations which provide the "globality."
MULTIDIMENSIONAL INTERPOLATION

Very frequent problem in multidimensional simulations. Should not be considered as infrequent problem!

In general, for an n-dimensional problem we have:
- \( n \) 1-dimensional vectors of coordinates:
  
  \[
  \begin{align*}
  x_1 &= (x_{11}, x_{12}, x_{13}, \ldots, x_{1n}) \\
  x_2 &= (x_{21}, x_{22}, \ldots, x_{2n}) \\
  \vdots \\
  x_n &= (x_{n1}, x_{n2}, \ldots, x_{nn})
  \end{align*}
  \]

- and an \( nxn \) matrix of function values at the different coordinate locations \( y = y(x_1, x_2, \ldots, x_n) \). The problem is then to calculate \( \tilde{y} = \tilde{y}(x_1, x_2, x_3, \ldots, x_n) \).

Hereafter we will consider a 2-dim problem in a CARTESIAN coordinate system.
In 2-dim we will have a 2x2 array of functional values

\[ Y(j, k) \]

where \( j \) and \( k \) are indices of the 2nd and 1st coordinates, respectively.

and two 1-D vectors of coordinates

\[ X_1(1:N), X_2(1:N) \] representing the intervals \( I_1 \) and \( I_2 \)

We want to calculate \( \bar{y} = y(\bar{x}_1, \bar{x}_2) \) where

\( \bar{x}_1 \in I_1, \bar{x}_2 \in I_2 \)

1. First of all we need to isolate the grid square where \((\bar{x}_1, \bar{x}_2)\) falls.

\[ \begin{array}{cccc}
1 & 2 & 3 & 4 \\
\hline
4 & & & \\
\hline
1 & & \ast X_1, X_2 & \\
\end{array} \]

1: \( X_1(j), X_2(k) \)
2: \( X_1(j+1), X_2(k) \)
3: \( X_1(j+1), X_2(k+1) \)
4: \( X_1(j), X_2(k+1) \)

\[ y_1 = Y(j, k) \]
\[ y_2 = Y(j+1, k) \]
\[ y_3 = Y(j+1, k+1) \]
\[ y_4 = Y(j, k+1) \]
• Next, we need the normalized weights

\[ t = \frac{x_1 - x_1(j)}{x_1(j+1) - x_1(j)} \]

\[ u = \frac{x_2 - x_2(k)}{x_2(k+1) - x_2(k)} \]

\[ D_1 \]
\[ D_2 \]

• Finally, we decide what kind of interpolation scheme we want to use.

Let's first consider **bilinear interpolation** i.e. first interpolate linearly in one direction and then in the other.

The formula to use is then:

\[
\begin{align*}
\hat{y} &= y(x_1, x_2) \\
&= (1-t)(1-u)y_1 + t(1-u)y_2 + tu y_3 + (1-t)u y_4
\end{align*}
\]

This is just like in 1-dim where functional value is multiplied by the non-adjacent interval (weight)
\[ y(x) = y_1 + \gamma y_2 \]
\[ \gamma \gamma_2 \]
\[ x_1 x \]
\[ x_2 \]

In 2-dim the multiplying weight is given by the area of the rectangle of dimensions \( u \times t \) or \( (1-u)x \times t \).

Bilinear interpolation is usually sufficient especially if the function is smooth and the grid rather fine.

Possible improvements upon BILINEAR go into two directions and respond to two different needs:

- Need for higher accuracy: \( \Rightarrow \) higher order methods: BICUBIC INTERP.

- Need for continuity of derivatives: \( \Rightarrow \) BICUBIC SPLINE.
Just as in cubic interpolations, we here need 4 coefficients associated to each grid-point and these are the functional values and the values of the derivatives and of the mixed derivatives $\Rightarrow$ 4×4 matrix

\[
\begin{array}{cccc}
  & 1 & 2 & 3 & 4 \\
\hline
x & & & & \\
y & & & & \\
y_1 &=& \frac{\partial y}{\partial x_1} & & \\
y_2 &=& \frac{\partial y}{\partial x_2} & \times & \\
y_{12} &=& \frac{\partial^2 y}{\partial x_1 \partial x_2} & \times & \\
y_{112} &=& \frac{\partial^3 y}{\partial x_1^2 \partial x_2} & \times & \\
\end{array}
\]

In practice it is possible to express the interpolated functional value as

\[
y(x_1, x_2) = \sum_{i=1}^{4} \sum_{j=1}^{4} c_{ij} \cdot t^{i-1} \cdot u^{j-1}
\]

where $u$ and $t$ are the normalized weights defined before.

The $c_{ij}$ are computed through suitable combinations of $y, y_1, y_2, y_{12}$ and are just numerical coefficients.
What if \( y_{11}, y_{12} \) and \( y_{112} \) are not known analytically?

Compute them numerically, e.g., for central expressions

\[
\frac{\partial y}{\partial x_1} \bigg|_{j,k} = Y_1(j,k) = \frac{Y(j+1,k) - Y(j-1,k)}{x_1(j+1) - x_1(j-1)} + O(\Delta x_1^2)
\]

\[
\frac{\partial y}{\partial x_2} \bigg|_{j,k} = Y_2(j,k) = \frac{Y(j,k+1) - Y(j,k-1)}{x_2(k+1) - x_2(k-1)} + O(\Delta x_2^2)
\]

and similarly for \( \frac{\partial^2 y}{\partial x_1 \partial x_2} \bigg|_{j,k} \)

Notes

- The bicubic interpolation guarantees that
  1) the values of the function and of the specified derivatives are reproduced exactly at the gridpoints
  2) The function and the derivative change smoothly in passing from one grid square to another one

- The smoothness has nothing to do with the accuracy in determining the derivatives. Improving derivatives only improves accuracy.
BICUBIC SPLINE

basically

It's the same as bicubic interpolation and can be seen as a hinting of...

Indeed the interpolating formula is the same but the values of the derivatives are determined "globally" through 1-dim splines.
INTEGRATION OF FUNCTIONS (or QUADRATURE)

It has a long history in numerical analysis, especially because computing integrals is/was more difficult than calculating derivatives.

If $F$ as a problem can be recast in the problem of integrating a differential equation

$$J = \int_{a}^{b} f(x) dx$$

$$\frac{dy}{dx} = f(x) \quad \text{with} \quad \text{bc's}$$

$$y(a) = 0 \quad ; \quad y(b) = I$$

but we will here treat

QUADRATURE $= \sum \text{i} \cdot C_i f(x=x_i)$

between $a \& b$ with $x_i \in [a,b]$ $C_i \in \mathbb{R}$

In doing so we should evaluate $I$ as accurately as possible with the smallest no of function evaluations. In doing so we will consider methods of increasing order which does not necessarily mean increasing accuracy.
The easiest approach to the problem is that of going to
FORMULAS FOR EQUALLY SPACED ABSIISAS
Some stuff:
Consider set of abscissas

\( x_1, \ldots, x_N \)

\( x_i = x_0 + ih \quad i = 0, \ldots, N-1 \)

set of functional values
\( f_i = f(x_i) \)

We will consider
(Closed: Newton-Cotes)
CLOSED FORMULAS: if we use \( f_0 \) for \( f_N \)

OPEN FORMULAS: if we don't use \( f_0 \) for \( f_N \)

Useful if \( f \) has an
integrable singularity
at \( x_0 \) or \( x_N \)
**Newton-Cotes Closed Formulas**

\[ \int_{x_1}^{x_2} f(x) \, dx = h \left[ \frac{1}{2} f_1 + \frac{1}{2} f_2 \right] + o \left( h^3 f'' \right) \]

- \( f'' \) is the second derivative evaluated somewhere in \([x_1, x_2]\). Prove it!

**Proof:**

\[ I = \int_{x_1}^{x_2} f(x) \, dx = f_1 h + f_1' \frac{h^2}{2} + o(h^3 f'') \]

\[ = f_2 h - f_2' \frac{h^2}{2} + o(h^3 f'') \]

\[ \Rightarrow 2I = h (f_1 + f_2) + \frac{h^2}{2} (f_1' - f_2') + o(h^3) \]

\[ \Rightarrow \]

\[ f_1 = f + \frac{h-e}{2} f'' + o(h^3) \]

\[ f_2 = f + (h-e) f' + \frac{(h-e)^2}{2} f'' + o(h^3) \]

Taking the derivative:

\[ f_1' = e f'' + \frac{e^2}{2} f''' + o(h^3) \]

\[ f_2' = (h-e) f'' + \frac{(h-e)^2}{2} f''' + o(h^3) \]

\[ \Rightarrow f_1' - f_2' = (2e-h) f'' + o(h^3) \]
\[ 2I = h \left( f_1 + f_2 \right) + \frac{h^2}{2} (2l - h) f'' + O(h^4 f^{(4)}) \]

\[ \Rightarrow I = h \left( \frac{1}{2} f_1 + \frac{1}{2} f_2 \right) + O(h^3 f'') \]

Q: Is there a class of functions for which this formula is exact?

A: Yes: being a 2-point formula is exact for polynomials of degree up to 1 (in this case \( f''(x) = 0 \) \( \Rightarrow \) the error term is zero).

Q: If we use a 3-point formula will we get exact expression for a polynomial of order 2?

A: No: A lucky cancellation makes Simpson’s formula exact up to a polynomial of order 3, included.

* Simpson’s formula

\[ \int_{x_1}^{x_3} f(x) \, dx = h \left[ \frac{1}{3} f_1 + \frac{4}{3} f_2 + \frac{1}{3} f_3 \right] + O(h^5 f^{(4)}) \]
Simpson's \( \frac{3}{8} \) rule (4 pts formula)

\[
\int_{x_1}^{x_4} f(x) \, dx = h \left[ \frac{3}{8} f_1 + \frac{9}{8} f_2 + \frac{9}{8} f_3 + \frac{3}{8} f_4 \right] + O(h^5 f^{(4)})
\]

Exact for polynomials of order 3 included (no cancellation).

Of course, for polynomials of order higher than 3, a 4pts formula is more accurate than a 3pts one.
Bode's rule (5 pts formula)

\[ \int_{x_1}^{x_5} f(x) \, dx = h \left[ \frac{14}{45} f_1 + \frac{64}{45} f_2 + \frac{24}{45} f_3 + \frac{64}{45} f_4 + \frac{14}{45} f_5 \right] + O(h^7 f^{(7)}) \]

Exact for polynomial of order up to 5 included.

Once we have these closed formulas for a small set of points we can iterate the calculation of the integral as a sum of small integrals

\[ I = \sum_i I_i \quad \text{where} \quad I_i = \int_{x_{i-1}}^{x_i} f(x) \, dx \]

In this case we talk of Extended Closed Forms

\[ \int_{x_1}^{x_N} f(x) \, dx = h \left[ \frac{1}{2} f_1 + f_2 + f_3 + \ldots + f_{N-1} + \frac{1}{2} f_N \right] + O \left( \left( \frac{b-a}{N^2} \right)^2 f^{(1)} \right) \]

What is the error?

We know that the trapezoidal formula has an error

\[ \varepsilon = O(h^3 f^{(1)}) \]

in the case of our extended formulas

\[ \varepsilon' = N \varepsilon = O \left( N h^3 f^{(1)} \right) = O \left( N \left( \frac{b-a}{N^2} \right)^3 f^{(1)} \right) = O \left( \left( \frac{1}{N^2} \right)^3 f^{(1)} \right) \]

Useful notation because our usually holds a & b fixed
\( I_n = \int f(x) \, dx = h_n \left[ \frac{1}{2} f_1 + \frac{1}{2} f_2 \right] \)

\( h_n = \left( \frac{N-1}{N} \right) h_{n-1} \)

\( h_2 = \frac{1}{2} h_1 \)

\( h_3 = \frac{3}{2} h_2 = \frac{3}{2} h_1 \)

\( I_1 = \frac{h_1}{2} (f_1 + f_2) \)

\( I_2 = h_2 (\frac{1}{2} f_1 + f_{1/2} + \frac{1}{2} f_2) = h_1 f_{1/2} + I_1 \)

\( I_3 = h_3 (\frac{1}{2} f_1 + f_{1/2} + f_{3/4} + \frac{1}{2} f_2) = I_2 + h_3 (f_{3/4} + f_{3/4}) \)

\( I_4 = h_4 (\frac{1}{2} f_1 + f_{1/2} + f_{3/4} + f_{5/8} + f_{1/2} + \cdots \frac{1}{2} f_2) \)

= \( I_3 + (f_{3/8} + f_{3/8} + f_{5/8} + f_{5/8}) h_4 \)

The no. of operations is not reduced but the algorithm is much simpler.

The relevance of this algorithm is that it cannot be extended to other formulas (e.g. Simpson). In other words, the trapezoidal rule lends itself to a numerical implementation.
Similarly we can derive Simpson’s extended formula:

\[
\int_{x_i}^{x_N} f(x) \, dx = h \left[ \frac{1}{3} f_1 + \frac{4}{3} f_2 + \frac{2}{3} f_3 + \frac{4}{3} f_4 + \ldots \right] + O\left(\frac{1}{N^4}\right)
\]

Let’s consider now the application to actual calculations; the guiding logic is the same: optimize computational cost and accuracy.

In this respect, the trapezoidal formula is very good as it can be used iteratively. In fact at each iteration one can simply double the no. of intervals and add the new function evaluation.

\[
\begin{array}{cccccc}
V_0 & 3/8 & 3/8 & 1/6 & 1/6 & 1/6 \\
V_2 & 3/4 & 3/4 & 1/6 & 1/6 & 1/6 \\
V_4 & & & & & \\
\end{array}
\]

N=1, N=2, N=3, N=4

Total after N=4

These are (\*), known already

9 function eval.
Note that the first order term is $\propto Be h^2$; why not $h^3$?

\[
\varepsilon = \frac{(a-b)^3 f''}{N^2}
\]

\[
h^2 (a-b) f'' \propto h^2
\]

\[
(a-b) = Nh \iff N = \frac{(a-b)}{h}
\]
So one can write an elementary routine which can be called iteratively and which continuously updates the value of the integral.

In NR this routine is **TRAPZD**

The trapezoidal extended rule is useful also because of the peculiar nature of the leading error term.

Recall that

\[ \int_a^b f(x) \, dx = h \left[ \frac{f_1}{2} + f_2 + \cdots + f_{N-1} + \frac{f_N}{2} \right] + O\left(\frac{(b-a)^3}{N^2}\right) \]

where the error term can be written as

\[ O\left(\frac{(b-a)^3}{N^2}\right) = -\frac{B_2}{2!} h^2 (f'_{N} - f'_{1}) - \cdots - \frac{B_{2k}}{(2k)!} h^{2k} (f^{(2k)}_{N} - f^{(2k)}_{1}) \]

where

\[ \frac{t}{e^t - 1} = \sum_{n=0}^{\infty} \frac{B_n}{n!} t^n \quad (\text{all odd } B_n \text{ is zero except } B_1 = -\frac{1}{2}) \]

**Note that** \((\Delta)\) **is not a convergent expansion but only an asymptotic expansion because the Bernoulli nos become very large**

The asymptotic convergence comes from the fact that \(\varepsilon_N < 2\varepsilon_{N-1}\)

error in the expansion with \(N\) terms
If I have used $2N$ points, I can then have two possible errors

$$
\varepsilon = \begin{cases} 
0(1/N^2) & \text{no sub.} \\
O(1/N^4) & \text{if the subtraction is made}
\end{cases}
$$

Q: Why do I need the $\frac{1}{3}$ coefficient in (*)?

exact integral

$$
\delta_N = I + O(1/N^2)
$$

$$
S_{2N} = I + O(1/4N^2)
$$

$$
4S_{2N} - S_N = 4I - I + O(1/N^4)
\Rightarrow \quad I = \frac{1}{3} (4S_{2N} - S_N) + O(1/N^4)
\quad \text{q.e.d.}
$$
The important aspect is that they depend on even powers of $h$

$$
\epsilon = O(h^{2k}) = O\left(\frac{1}{N^{2k}}\right)
$$

and there are no other contributions say $O(N^3)$, $O(N^5)$ as in Simpson's rule.

Why is this important?

Let $S_N, S_{2N}$ the evaluation of an integral over $N$ and $2N$ points

Then

$$
\epsilon(S_N) = 4 \epsilon(S_{2N}) \quad (\epsilon = O(1/N^2))
$$

$$
\Rightarrow \text{we can define a new evaluation of the integral}
$$

$$
S^* = \frac{4}{3} S_{2N} - \frac{1}{3} S_N = O(1/N^4)
$$

(1)

The reason for this is that by subtracting we get rid of the error $O(1/N^2)$ and are therefore (since there is no $O(1/N^3)$ error) left only with the $O(1/N^4)$ one.

Indeed one can avoid using Simpson's rule and use (1) and still obtain
IMPROPER INTEGRALS

An integral can be defined improper when:

(a) The integrand is finite at one of the limiting values but cannot be evaluated there. For example, \( \frac{\sin(x)}{x} \) at \( x = 0 \).

(b) Has upper (lower) limit as \( \pm \infty \).

(c) It has an integrable singularity at either limit (e.g., \( x^{-1/2} \) at \( x = 0 \)).

(d) It has an integrable singularity somewhere between the limits.

Note: Improper integrals are calculable, but if the integral is infinite, then it is just impossible (no way out!)

A good general purpose algorithm is given by the midpoint formula:

\[
\int_{x_1}^{x_n} f(x) \, dx = h \left[ f_{3/2} + f_{5/2} + f_{7/2} \ldots + f_{n-3/2} + f_{n-1/2} \right] + O\left(\frac{1}{N^2}\right)
\]
\[ I = \int + O\left( h^2 f''\right) \]

\[ \tilde{I} = \int + O\left( \frac{h^2}{8} f''\right) \]

\[ \tilde{I} - I = O\left( \frac{h^3 f''}{8} \right) - O\left( h^3 f''\right) \]

\[ = O\left( \frac{3h^3 f'''}{8} \right) \]

\[ 2\tilde{I} - I = \tilde{I} + \tilde{I} - I \]

\[ = \int + O\left( \frac{h^2}{8} f''\right) + O\left( \frac{7h^2 f'''}{8} \right) \]

\[ - \int + O\left( h^2 f''\right) \]
Indeed (i) can be seen as a "double" use of the extended trapezoidal formula:

\[
\int_{x_1}^{x_N} f(x) \, dx = h \left( \frac{1}{2} f_1 + f_2 \cdots + f_{N-1} + \frac{1}{2} f_N \right) - I
\]

\[
= \frac{h}{2} \left( \frac{1}{2} f_1 + f_2 + f_N \right) - f_{N-1} + f_{N-\frac{1}{2}} + \frac{1}{2} f_N
\]

Let \( \overline{I} \) be the exact integral, then:

\[
\overline{I} - I = o \left( h^3 f^{(n)} \right)
\]

\[
\overline{I} = I + o \left( h^3 f^{(n)} \right)
\]

\[
2\overline{I} - I = h \left( \frac{1}{2} f_1 + f_2 \cdots + \frac{1}{2} f_N \right) -
\]

\[
\frac{h}{2} f_1 + \frac{1}{2} f_N
\]

\[
= h \left( f_{N-\frac{1}{2}} + f_{N-\frac{1}{2}} \cdots + f_{N-\frac{1}{2}} \right)
\]

\[
= I + o \left( h^3 f^{(n)} \right)
\]

\[\text{qed}\]
Notes

1) If \( f(x) \) is a polynomial, then

\[
\int_{a}^{b} W(x) f(x) \, dx = \sum_{j} W_{j} f_{j}
\]

ie the expression is exact. This can be proven by exploiting the properties of orthogonal polynomials.

2) In general

\[
\int_{a}^{b} W(x) f(x) \, dx = \sum_{j} W_{j} f_{j} + O(\sqrt[2k]{N})
\]

ie one obtains an order that is twice that of the Newton-Cotes formulas seen so far for the same \( N \) of function evaluations.

3) High order \( \Rightarrow \) high accuracy

... only if \( f(x) \) is a sufficiently smooth function does this happen
GAUSSIAN QUADRATURES

The basic idea is to choose not only the coefficients but also the location of the abscissae that make your integral as accurate as possible.

\[ \int_a^b W(x) f(x) \, dx \approx \sum_j w_j f(x_j) \]

suitably chosen

\[
(*)
\]

According to the choice of \( W(x) \) we have different types of Gaussian quadratures:

1. \( W(x) = 1 \) \( \quad x \in [-1, 1] \)
   GAUSS--LEGENDRE

2. \( W(x) = (1-x^2)^{-1/2} \) \( \quad x \in [-1, 1] \)
   GAUSS--CHEBYSHEV

3. \( W(x) = x^{\mu} e^{-x} \) \( \quad x \in [0, \infty] \)
   GAUSS--LAGUERRE
Example

\[ W(x) = 1 \] \text{ with } 10 \text{ points}

Then

\[ W_1 = 0.299 \ldots = W_9 \]
\[ W_2 = 0.269 \ldots = W_8 \]
\[ W_3 = 0.219 \ldots = W_7 \]
\[ W_4 = 0.149 \ldots = W_6 \]
\[ W_5 = 0.666 \ldots \]

\[ x_1 = 0.148 = x_9 \]
\[ x_2 = 0.433 = x_8 \]
\[ x_3 = 0.679 = x_7 \]
\[ x_4 = 0.865 = x_6 \]
\[ x_5 = 0.973 \]

Note that

\[ \int_{a}^{b} W(x) f(x) \, dx = \sum_{j=1}^{N} g(x_j) v_j = \sum_{j} v_j g(x_j) \]

\[ g(x) = W(x) f(x) \]
\[ v_j = \frac{W_j}{W(X_j)} \]

ie one can play around to reformulate the integral in terms of the density weighting function.

Ex \[ g(x) \] is integrand,
\[ g(x) \] is expressed as \[ W(x)f(x) \]

\[ \int_{a}^{b} g(x) \, dx = \int_{a}^{b} W(x) f(x) \, dx = \sum_{j} W_j p_j = \sum_{j} \frac{W_j}{\text{int.}} g_j = \sum_{j} v_j g_j \]
GQ is related to orthonormal polynomials

\[ \langle x_j \rangle = \text{zero of polynomials} \]

\[ w_j = \text{coefficients that provide the correct quadrature} \]

In this case, \( w_j = w_i / W(x_j) \) is not defined

& polynomials such that

\[ \langle f^* g \rangle = \int_{a}^{b} w(x) f(x) g(x) \, dx = 0 \]

\[ \langle f f \rangle = \int_{a}^{b} w(x) f(x) f(x) \, dx = 1 \]

**MULTIDIMENSIONAL QUADRATURES**

There are two main possibilities

A) boundary simple: then break up the problem in many successive 1D-integrals

\[ \Sigma = \int_{x_1}^{x_2} \int_{y_1}^{y_2} \int_{z_1}^{z_2} f(x,y,z) \, dx \, dy \, dz \]
Thus I can compute

\[
G(x, y) = \int_{x_1}^{x_2} f(x, y, z) \, dz
\]

and

\[
H(x) = \int_{y_1}^{y_2} G(x, y) \, dy
\]

once \( x \) is fixed you know the relevant \( y_2 \) and \( y_1 \)

so that

\[
I = \int_{x_1}^{x_2} H(x) \, dx
\]

3) the boundary is complicated: use Monte Carlo techniques

6 next lecture
Introduction to Random numbers:

Q: What is a random number sequence produced by a computer?

After all a computer will produce an output following a completely deterministic algorithm.

In reality all computer generated random numbers are "pseudo-random" in the sense that given an initial seed, the computer will always produce the same sequence.

We here need a pragmatic approach to this issue:

A sequence of numbers will be considered random if it "appears" relative to a number of statistical tests aimed at ferreting out any correlation.

In other words:

what is random for an application may not be random enough for another
\[ p(x) : \text{ prob. in } x \]
\[ \int_x^{x+dx} p(x) \, dx : \text{ between } x, x+dx \]
\[ \int_a^b p(x) \, dx = (\text{Total probability}) \leq 1 \]

If all \( x \) are equally probable, then
\[ p(x) = k = \text{const} \]
\[ \int_0^1 p(x) \, dx = k (b-a) = 1 \]
\[ \Rightarrow k = \frac{1}{b-a} \]

\[ p(x) = \frac{1}{b-a} \]
\[ P(X) = \int_x^{x+dx} \frac{dx}{b-a} \]

If \( b-a = 1 \), \( \Rightarrow k = 1 \); \( p(x) = 1 \)
Random Nos are classified according to their 
**UNIFORM DEVIATES** distribution functions

Def: Random numbers that lie within a given range (e.g. \(x \in [9, 17]\)) and that are equally probable.

\[
\begin{align*}
\int_{x}^{x+dx} p(x) \, dx & : \text{prob. of generating a number between } x \text{ and } x+dx \\
& = \text{const}
\end{align*}
\]

The most diffused and efficient generators of uniform deviates are the **LINEAR CONGRUENTIAL GENERATORS** (LCG).

A sequence of \(m\) integers \(I_1, I_2, \ldots, I_{m-1}\) is produced by the recurrence relation:

\[
I_{j+1} = a \cdot I_j + c \quad \text{mod} (m)
\]

- **multiplier** (integer) \(a, c, m\) are suitably fixed and \(m\) is a large integer.
- \(I_j\) needs to be provided and is referred to as **SEED** and the sequence will repeat itself with a
\[ I_{j+1} \mod (m) \]

means that I consider

\[ I_{j+1} = \frac{(a \cdot I_j + c)}{m} - \text{INT}\left[\frac{(a \cdot I_j + c)}{m}\right] \]

This is the output I've indicated with \( r \)

\[ I_j = 3 \]
\[ a = 2 \]
\[ c = 1 \]

\[ I_{j+1} = 7 \quad m = 3 \]

\[ I_{j+1} \mod (m) = \frac{7}{3} - \text{INT}\left(\frac{7}{3}\right) \]

\[ = 2.\overline{3} - 2 = 0.\overline{3} \]
LCGs are fast (simple algorithms) but have problems of sequential correlation on successive calls.

Take \( k \) random numbers \( x_1, \ldots, x_k \) \( \in [0,1] \).

Do this a large number of times and plot them in a \( k \)-dimensional space.

The space will not be filled uniformly but points will cluster on \((k-1)\)-dimensional planes and there are at most \( m^\frac{1}{k} \) such planes, where \( m \) is the largest integer available.

Take \( m = 2^{32} \) and \( k = 3 \) (triplets).

The number of planes is \( m^{\frac{1}{3}} \approx 1600 \).

If one is looking at a physical process that occurs on a large portion of the total volume, discretization problems are potentially high.
How to proceed?

Never use a random number generator which has not been statistically tested

PORTABLE (machine independent) GENERATORS

There are general available but I personally suggest to look at those of NRs

In particular:

\[
\begin{align*}
\text{rand} & : \quad \text{very basic, period } = 2^{31} - 2 \sim 10^9 \\
& \quad \text{suffers of low numbers, serial correlations, it's a good benchmark}
\end{align*}
\]

\[
\begin{align*}
\text{rand1} & : \quad \text{statistically better behaved than rand, but smaller period } \approx 10^8 \\
\text{rand2} & : \quad \text{almost a "perfect" generator, period } \approx 10^{18} \approx \text{infinity on present machines}
\end{align*}
\]

\[
\begin{align*}
\text{rangel} & : \quad \text{quick and dirty} \\
\text{range2} & : \quad \text{periods } \sim 10^4 - 10^6
\end{align*}
\]
Random numbers are the stepping stones of Monte Carlo techniques where large numbers of "attempts" are necessary.

**TIME is an important aspect**

**Generator**

<table>
<thead>
<tr>
<th>ran 0</th>
<th>ran1</th>
<th>ran 2</th>
<th>ranqd 1</th>
<th>ranqd2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.0</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.3</td>
<td>1.0</td>
</tr>
</tbody>
</table>

useful if generators random numbers are not very frequently used

V very good compromise (best buy?)

Q: What should we use random numbers for? Isn't that a waste of technology?
A: No! There are uncountable situations in which one may want to explore a random realization of a scenario.

Random numbers can be used to build

A: Deviates

- Uniform
- Non-Uniform
  - Gaussian (Normal)
  - Binomial (Poisson)
  - Exponential

B: Monte Carlo Integration.
(6)

$p(x)$ is the prob. distrib. function

$$\Rightarrow \int_a^b p(x) \, dx = P: \text{prob. of getting } x \text{ in } [a,b]$$

on the other hand $P$ is such that

$$\frac{dP}{dx} = p(x) \Rightarrow$$

$$dP = p(x) \, dx \quad \text{and hence}$$

$$dP \quad \text{which is the prob. of finding } x \text{ in } [x, x+dx] \text{ is simply given by } p(x) \, dx$$

(...)

$p(x) \, dx = dx \quad x \in [0, 1]$ from the normalization constraint

$$\int_0^1 p(x) \, dx = 1 = \int_0^1 p(x) \, dx = k \int_0^1 \, dx = K \Rightarrow k = 1$$

$q(x)$
Generation of \textit{Deviates}: probably disturbed

Two basic methods for doing this:

1) Transformation Method: Exponential and Normal Deviates, fastest

2) Rejection Method: more powerful, general techniques slower

1) Transformation Method

Let \( p(x) \, dx \): prob. of choosing \( x \) in the interval \( x, x+dx \)

so that in the case of uniform deviates

\[
\int_0^1 p(x) \, dx = 1
\]

Suppose now that you don't want the \( p(x) \) to be uniform but rather follow a certain functional dependence

\[
y = y(x)
\]
Then, the transformation laws of probability of correlated events guarantees that
\[ |p(y) dy| = |p(x) dx| \]
\[ p(x) = \left| \frac{dx}{dy} \right| p(y) \]
\[ p(x) = \left| \frac{dy}{dx} \right| p(y) \]

Here is the trick: If I can build a uniform deviate, I can build any deviate with given functional dependence if this is known.

Example: Exponential deviate

Suppose you want the distribution function to be an inverse exponential: ie \( p(y) = e^{-y} \)

Q: How do I build this deviate using a uniform deviate?

A: I know that \( p(y) dy = \left| \frac{dx}{dy} \right| dy = e^{-y} dy \)

\[ \frac{dx}{dy} = e^{-y} \Rightarrow y = -\ln x + \text{const} \]

Random nos. are distributed so that there are many at small \( y \) and few at large \( y \).
(2)
In the previous example:

\[ f(y) = e^{-y} \]

(3)
In the previous example:

\[ F(y) = x = e^{-y} \]

\[ \int f(y) \, dy = \int e^{-y} \, dy = e^{-y} \]
In other words, to have an exponential deviate I can start from a uniform deviate and then impose

$$X_{ED} = -\ln X_{UP}$$

\[\text{inverse exponential deviate}\]

\[\text{uniform deviate}\]

**Example**

```python
function expdev (seed)
    tmp = ran1 (seed)  # uniform deviate
    expdev  = -log (tmp)  # exponential deviate
    return expdev
end function
```

In general, complicated arbitrary function representing the desired non-uniform deviate

Let $P(y) = f(y)$

$\frac{d x}{dy} = P(y) = f(y)$ \Rightarrow define \( x \) as

$$x := \int f(y) \, dy = F(y)$$

so that

Inverse function

$$y = F^{-1}(x)$$

Relation between uniform deviate $x$ and non-uniform $y$

Drawback: in order to have a deviate of dependence $f(y)$ I need to calculate, either numerically...
or analytically, $F^{-1}(x)$: the inverse of the integral of $f(y)$

For some functions (e.g., Gaussian), this is possible. Let's consider the problem from a geometrical point of view.

For instance, $F(y) = \int_0^y f(y) \, dy = \int_0^\infty p(y) \, dy$

Let's say $X = F(Y) = \int_0^Y f(y) \, dy$ from a uniform deviate between 0 and 1, given an $X$, find the value $Y$ such that

$\int_0^Y p(y) \, dy = X$; of course, if I know $F^{-1}$ then the task is trivial, and $Y = F^{-1}(X)$

Q: What if $F^{-1}(y)$ cannot be calculated?

2) Rejection Method: more powerful and generic since does not require the analytic knowledge of $F$ or $F^{-1}$
In practice, what we need is to generate random numbers uniformly in two dimensions (ie the region under the curve is filled uniformly with pts).

What we can do is to choose a function $f(y) > p(y)$ (always possible because $\int p(y) \, dy = 1$, hence $p$ finite).

Generating from random numbers in two dimensions below $f(y)$ is now easier because we know all of $q(y)$. One that is done we can then reject all those numbers that are below $f(y)$ but above $p(y)$.

$$\text{OVERHEAD} = \int q(y) \, dy - \int p(y) \, dy$$
The rejection method can therefore be split into the following steps:

1) Pick \( x \) from uniform distribution in \([0, 1]\)

2) Find \( \overline{y} \) s.t. \( \overline{x} = Q(\overline{y}) = \int_{0}^{\overline{y}} q(y) \, dy \)
   that is \( \overline{y} = F^{-1}(\overline{x}) \) : analytic inversion

3) Evaluate \( q(\overline{y}) \)

4) Pick \( x' \) from uniform distribution in \([0, q(\overline{y})]\)

5) Reject if \( x' < p(\overline{y}) \)
1. If the deviate uniform and not between 0 and 1, it can always be recast in this interval.

\[ p(x) = K = \text{const} \]

\[ \int_{a}^{b} p(x) \, dx = 1 \implies \int_{0}^{1} \frac{p(x)}{b-a} \, dx = 1 = \int_{a}^{b} p(x) \, dx \]

Proof:

\[ \int_{a}^{b} p(x) \, dx = (b-a)K = 1 \implies K = \frac{1}{b-a} \]

\[ \int_{0}^{1} \frac{p(x)}{b-a} \, dx = \int_{0}^{1} k \, dx = \int_{0}^{1} dx = 1 \quad \text{q.e.d.} \]

2. In the rejection method the only relevant normalization is that on \( p(y) \), i.e.

\[ \int_{y_1}^{y_2} p(y) \, dy = 1 \]

\[ \int_{y_1}^{y_2} q(y) \, dy = K > 1 \]
\( x \in [0, \alpha y_2] \)

\[ \bar{y} = F^{-1}(x) = \frac{x}{\alpha} \]
3. The rejection is much simpler if 
\[ q(y) = \text{const.} = \alpha \]

\[ \begin{array}{c}
\text{In this case} \\
1) \text{choose } y \text{ from uniform distribution} \\
in \ y_1, y_2 \quad y \in [0, y_2] \\
2) \text{calculate } p(y) \\
3) \text{pick number } f \text{ from uniform deviate between} \\
0 \text{ and } \alpha \\
4) \text{Reject if } f > p(y) \\
\end{array} \]

Note that in the first step I choose \( y \in [y_1, y_2] \) and not \( x \in [0, \alpha y_2] \); this is because
\[ \int_{y_1}^{y_2} p(y) dy = \int_0^{\alpha y_2} p(x) dx \]
4. Overhead is minimized if \( \alpha = \max(p(y)) \)

\[
\begin{align*}
\text{Percentage overhead is } & \frac{A_1 - A_2}{A_1} = 1 - \frac{A_2}{A_1} \\
3. \text{This works well unless } p(y) \text{ has large local maximum}
\end{align*}
\]

Also in this case the overhead is acceptable
In summary, the rejection method using a constant function set to the maximum of the probability is a robust and efficient method to build an arbitrary distribution function by starting from a uniform distribution.

It's the method to use!

**MONTE CARLO TECHNIQUES**: For our purposes, they are best suited to calculate integrals especially in higher dimensional spaces.

The Monte Carlo arises from the random character of the method (chance in cosmos).

The basic strategy is in determining the integral as a sum of \( N \) random evaluations of the integrand.

The fundamental theorem of MC integration is

\[
\int f \, dV = \sum_{i=1}^{N} \left( f(x_i) \pm \sqrt{\frac{\left( f^2 \right) - \left( \langle f \rangle \right)^2}{N}} \right)
\]

(1)

where

\[
\langle f \rangle = \frac{1}{N} \sum_{i=1}^{N} f(x_i) \quad \text{and} \quad \langle f^2 \rangle = \frac{1}{N} \sum_{i=1}^{N} f^2(x_i)
\]

(2)
Example
\[ I = \int \frac{dx}{1 + x^2} = \frac{\pi}{4} = 0.78540 \]

Q: Is there a way of optimizing the calculation?

A: We want to minimize the integral variance \( \langle f^2 \rangle - \langle f \rangle^2 \).

Q: How?

A: The variance is zero if \( f \) is constant and so the optimization is reached by making the integrand as close as possible to a constant function \( f(x) \) divided by a function \( w(x) \) very similar to \( f(x) \).

Proof:
\[ h(x) = \frac{f(x)}{w(x)} \]
\[ I = \int f(x) dV = \int \frac{f(x)}{w(x)} w(x) dV = \int h(x) w(x) dV \]
\[ = \int h(x) dV' \text{ where } dV' = w(x) dV \]

Let \( U' = \int dV' = \int w(x) dx = w(x) \) can be chosen so that \( U' = 1 \).
Note that I do not need to divide by $\frac{V}{V'}$ to obtain the average because the $V$ integral is made over $V'$ where

$$V' = \int \delta V' = 1 = \int \delta V$$

$$\langle f \rangle = \frac{1}{V} \int f \delta V = \frac{1}{V} \int f' \delta V'$$

$$= \int f' \delta V'$$

$$= \int f \frac{\partial V}{\partial V'} \delta V$$

$$\langle \frac{f}{w} \rangle = \frac{1}{V} \int \frac{f}{w} \delta V = \frac{1}{V} \int \frac{f'}{w} \delta V'$$

$$= \frac{1}{V'} \int \frac{f'}{w} \delta V'$$

$$= \int f' \delta V$$
Then

\[ I = \int f(x) \, dx = V' \langle h \rangle + \sqrt{\frac{\langle h^2 \rangle - \langle h \rangle^2}{N}} = V' \langle h \rangle \pm \frac{S^2}{\sqrt{N}} \]

\[ S = \frac{\langle f^2 \rangle}{w^2} - \left( \frac{\langle f \rangle}{w} \right)^2 = \int \frac{f^2}{w^2} \, w \, dV - \left( \int \frac{f}{w} \, w \, dV \right)^2 \]

\[ = \int \frac{f^2}{w} \, dV - \left( \int f \, dV \right)^2 \]

Lagrange Multiplier

\[ \frac{\delta S}{\delta w} = 0 \implies \frac{\delta}{\delta w} \left[ \int \frac{f^2}{w} \, dV - \left( \int f \, dV \right)^2 + \lambda \int w \, dV \right] \]

\[ \implies w \propto k |f| \]

\[ \Rightarrow 0 = -\frac{f^2}{w^2} + \lambda \Rightarrow w = \frac{1}{\sqrt{\lambda}} \; \; ; \; \; f^2 = \lambda w^{-2} \]

When (\cdot) is satisfied the variance is optimal and hence zero:

\[ S = \text{Optimal} = \left[ \int w \, dV - \lambda \left( \int w \, dV \right)^2 \right] = \left[ \lambda \int w \, dV - \lambda \left( \int w \, dV \right)^2 \right] = 0 \]

See exercise B

\[ w(x) = \frac{1}{3} (4 - 2x) \quad \int w(x) \, dx = 1 \]
Monte Carlo integrations are particularly useful in multidimensions, where the volume of integration is complicated. In this case

\[
I_1 = \int_{V_2} f \, dV \\
I_2 = \int_{V_2} f \, dV = \text{easily calculable}
\]

Then

\[
\frac{I_1}{I_2} = \frac{V_1}{V_2} = \frac{N_1}{N_2} \Rightarrow I_1 = \frac{N_1}{V_2} \int_{V_2} f(x) \, dV
\]

See exercise

\[
\frac{N_1}{N_2} < 1 \quad \text{such that} \quad V_1 = V_2 \frac{N_1}{N_2} \text{. In this way you can compute } V_1 \text{ and complete the integration.}
\]
Problems involving ODEs can be reduced to the study of first-order differential equations which can be solved to ordinary accuracy.

This justifies the frequent expression that the "numerical solution of an ODE is "exact".

\[
\frac{d^2y}{dx^2} + q(x) \frac{dy}{dx} = r(x) \quad \leftrightarrow \\
\begin{align*}
\frac{dy}{dx} &= z(x) \quad \text{introduction of new variables} \\
\frac{dz}{dx} &= -q(x)z(x) + r(x)
\end{align*}
\]

In general, we want to solve a set of \(N\) first-order ODEs and even more complex ODEs should be transformed in this form.

\[
\frac{dy_i}{dx} = f_i(x, y_1, \ldots, y_N) \quad i = 1, \ldots, N
\]

It's well known that the Cauchy problem in the solution of our ODE requires also the definition of boundary conditions.
According to the type of bcs specified, the solution of an ODE can be classified as:

1) **Initial Value Problem**

\[ y_i = y_i(x_{s,i}) \]

Initial x values

...and the solution is then to be found everywhere up to \( x = x_f \)

2) **Two-point boundary value problem**

(for ODE of order > 1)

\[ \tilde{y}_i = y_i(x_{s,i}) ; \quad \tilde{y}_i = y_i(x_{f,i}) \]

Initial values

\[ \tilde{y} \]

Final value

Hereafter we will deal only with IVPs.
\[ \frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim_{x \to 0} \frac{y(x_{n+1}) - y(x_n)}{x_{n+1} - x_n} \]

\[ = \lim_{h \to 0} \frac{(y_{n+1} - y_n)}{h} \]

\[ \Rightarrow \]

\[ \frac{dy}{dx} = f \implies y_{n+1} = y_n + h f + O(h^2) \]
The basic idea behind all methods to solve an ODE is discretization:

\[ \frac{dy}{dx} \rightarrow \Delta y \quad \text{and} \quad y \rightarrow y_1, \ldots, y_N \]

\[ dx \rightarrow \Delta x \quad \text{where} \quad y_j = y(x_j) \]

\[ \frac{dy}{dx} = f \rightarrow (\ast) \]

\[ y_{n+1} = y_n + h f(x_n, y_n) \]

\[ x_{n+1} = x_n + h \]  

Euler's Method

It's conceptually very important but of irrelevant usefulness.

There are 3 main classes of ODE solvers:

* **RUNGE-KUTTA** methods: based on Euler's method to evolve the solution through a number of steps which use information on the first derivative at different locations.

* **RICHARDSON EXTRAPOLATION**: uses the idea of extrapolating the computed result to the case in which the step is infinitesimal (Burlisch-Stoer).

* **PREDICTOR-CORRECTOR**: extrapolate the solution to the next step and use value of the derivative there to improve upon the prediction.
We will look in detail at Runge-Kutta. The building block is Euler’s method.

\[ y_{n+1} = y_n + hf(x_n, y_n) + O(h^2) \]

(first-order method)

This method is not accurate nor stable but establishes the basic path to follow in defining more accurate and stable methods.

**Midpoint** or 2nd order Runge-Kutta

Let 

\[ k_1 = hf(x_n, y_n) \quad \text{at initial point} \]

\[ k_2 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right) \quad \text{at midpoint} \]

then

\[ y_{n+1} = y_n + k_2 + O(h^3) \]

see graph at D6
Proof

\[ y_{n+1} = y_n + y'_n \frac{h}{2} + y''_n \frac{h^2}{2} + O(h^3) \]  
\[ y_{n+2} = y_n + y'_n \frac{h}{2} + y''_n \frac{h^2}{8} + O(h^3) \]  
\[ y_{n+1} = y_{n+1/2} + y'_{n+1/2} \frac{h}{2} + y''_{n+1/2} \frac{h^2}{8} + O(h^3) \]  

Now, (*) \[ \Rightarrow \]

\[ h y'_n = y_{n+1} - y_n - y''_n \frac{h^2}{2} + O(h^3) \]
\[ \Rightarrow \]

\[ \frac{1}{2} y_{n+1} = \frac{1}{2} y_n - y''_n \frac{h^2}{4} + y''_n h^2 + y'_{n+1/2} \frac{h}{2} + y''_{n+1/2} \frac{h^2}{8} + O(h^3) \]  
but also, taking the second derivative of (**), we have

\[ y'_{n+1/2} = y'_n + y''_n \frac{h}{2} + O(h^2) \]
\[ y''_{n+1/2} = y''_n + O(h) \]
As a result, the terms $O(h^2)$ cancel out and

$$y_{n+1} = y_n + \frac{y_n'}{2} h + O(h^3)$$

Graphically:

We can do better than a 2nd-order RK

$\Rightarrow$ 4th-order RK

The idea is always the same: evaluate the RHS (i.e., first derivative) at different locations and cancel out lower-order error terms.

In particular, we can have 2 evaluations at the starting and ending points plus two evaluations at midpoints so that
\[ k_1 = hf(x_n, y_n) \]
\[ k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \]
\[ k_3 = hf(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}) \]
\[ k_4 = hf(x_n + h, y_n + k_3) \]
\[ y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5) \]

See fig.

Q: If we maintain the same accuracy, when is a 4th-order scheme better than a 2nd-order one?

Step taken with the 4th order scheme

This is because in a 4th-order you make 4 evaluations vs the 2 of a 2nd order

Gain if \( h < 2h^2 \)

NOTE

For most of the practical uses, a 4th-order RK is the first and last word in ODE integrators. It requires 4 evaluations; higher order requires \( M+1, M+2 \) It's robust, accurate and a real workhorse. However it can be improved in two ways:

- ADAPTIVE STEP SIZE CONTROL
- HIGHER-ORDER ALGORITHMS
ASSC (Adaptive Step Size Control)

Unless one is in the need of performing the solution at equally spaced values in \( x \), the use of a constant step size might be computationally undesirable and expensive.

A very good example is given by the integration of an elliptic orbit

\[ \Omega \propto r^{-3/2} \quad \text{i.e. } \Omega \text{ varies rapidly for small } r \]

It's straightforward to realize that the solution of the ODE is much harder in the regions near the periapsis. If we set a very small step to calculate this region accurately we will waste a lot of steps in region B where we could take much fewer steps.

We need a way to understand when the step taken was of a suitable size or needed to be smaller/larger.

\( \Omega \) where \( \Omega \) is very large
Q: How do we do this?

A: An obvious way to proceed is to take the step in 2 different ways (single large step and 2 small steps) and then compare to make a decision for the following step: this is called STEP DOUBLING

\[
\begin{align*}
&x \\
&x + h \\
&x + 2h \\
&\text{4 evals}
\end{align*}
\]

In the big step we take 4 evals, in the 2 small steps \( n^2 \times 1 = 7 \) evals.

In total, 11 evaluations.

What is the overhead?

\( \approx 40\% \) not 200%.

\[
\frac{11}{8} = 1.375
\]

This is acceptable and in most cases a good investment since it allows to increase the step size in the following step, usually by an even larger amount.
1) Calculate $y_1, y_2$

2) $u, \quad \Delta = y_1 - y_2$

3) Compare $\Delta$ to $\Delta_{ref}: \frac{\Delta_{ref}}{\Delta}$

4) Set the new step as $h_{new} = h \left( \frac{\Delta_{ref}}{\Delta} \right)^{1/5}$
How do we use step-doubling? Is how can we continuously increase the step?

Let \( y_\text{E}(x+2h) \) be the value of the solution at \( x+2h \)

\[
y_{\text{E}}(x+2h) = y_{\text{E}} + (2h)^{5} k + O(h^{6})
\]

(first evol. 4 steps)

\[
y_{2}(x+2h) = y_{\text{E}} + 2 (h^{5}) k + O(h^{6})
\]

(second evol. 7 steps)

So the difference in the two numerical estimates is

\[
\Delta = y_{2} - y_{1} \propto h^{5} : \text{this is the truncation error}
\]

So that if \( \Delta_0 \) is the difference with a step \( h_0 \), we can derive the next useful step \( h_{\text{new}} \) as

\[
h_{\text{new}} = h_{0} \left( \frac{\Delta_{\text{ref}}}{\Delta_0} \right)^{1/5}
\]

where \( \Delta_{\text{ref}} \) is the required error to be produced. Clearly,

if \( \Delta_{\text{ref}} > \Delta_0 \Rightarrow h_{\text{new}} > h_0 \) : I can take a bigger step and vice versa.

In other words, the logical sequence is: (*)
\[ y_1 = y_e + K \cdot 32h^5 + O(h^6) \]
\[ y_2 = y_e + K \cdot 2h^5 + O(h^6) \]
\[ y_1 - y_2 = K \cdot 30h^5 + O(h^6) = -\Delta \]
\[ \Rightarrow k = -\frac{\Delta}{30h^5} + O(h) \]

On the other hand
\[ y_2 = y_e - \frac{\Delta}{30h^5} 2h^5 + O(h^6) \]
\[ = y_e - \frac{\Delta}{15} + O(h^6) = \]
\[ \Rightarrow \quad y_e = y_2 + \frac{\Delta}{15} + O(h^6) \]
\[ = y_2 + \frac{y_2 - y_1 + O(h^6)}{15} = \frac{14y_2 - y_1}{15} + O(h^6) \]
There are some important things to underline.

1) Once we have $y_1$ and $y_2$, we could write

$$y(x+2h) = y_2 + \frac{\Delta}{15} + O(h^6)$$

2) Imagine of one order our estimate of $y(x+2h)$. This is not as good as it seems because although more accurate, we cannot control the truncation error of (ii). We basically don't know whether it is improving the solution or not: once again, we shouldn't confuse higher order with higher accuracy and $\Delta$ may be longer than $2h^5/5!$ since we don't know $k$. 

2) Note that if we have a set of coupled 1st-order ODEs, there is going to be a $\Delta$ for each of these and of course we should make our choice in step size based on the largest of these $\Delta$s (the largest truncation error sets the pace).

3) There are different ways to prescribe a good $\Delta$ref which cannot be based on the spacing $h$ since this is what we want to determine.
a) Relative change in the solution
\[ \Delta \text{ref} = \varepsilon \cdot y \] overall tolerance level
This is ok if \( y \neq 0 \) always.

b) Bounded change
\[ \Delta \text{ref} = \varepsilon \cdot \text{max} (y) \]

c) Multi-purpose: works also if \( y \leq 0 \)
\[ \Delta \text{ref} = \varepsilon \cdot \left[ y + h \cdot f \right] \] \( \varepsilon \)

4) Never use a too small \( \Delta \text{ref} \) or you run into the problem of never moving the solution away from the initial point.
\[ y_{n+1} = y_n + \sum \text{ci} \cdot i \]

\( \text{O}(h^m) \)
The last comment on RK methods should be dedicated to the possibility of increasing the accuracy. We have already seen that if $M$ is the order of the integrator, for $M \leq 4$ are sufficient $M$ evaluations of 1st deriv.

for $M > 4$ $M+1, M+2$ are necessary.

In this context, a special mention is deserved by the Cash-Karp RK method which gives a 5th-order estimate with 6 evaluations:

$k_1 = hf(x_n, y_n)$

$k_2 = hf(x_n + a_2 h, y_n + b_{21} k_1)$

$k_6 = hf(x_n + a_6 h, y_n + b_{61} k_1 + \ldots + b_{65} k_5)$

so that

$y_{n+1} = y_n + \sum_i^6 c_i k_i + O(h^6)$

and where the: $a_j, b_{ij}, c_i$ are all tabulated and can be found on textbooks.
\[ Z_z = Z_i + h f'(x+h, z_i) + O(h^2) \]

\[ = Z_0 + hf'(x, z_0) + hf'(x+h, z_i) + O(h^2) \]

\[ = Z_0 + h \left[ f'(x, z_0) + f'(x+h, z_i) \right] + O(h^2) \]

\[ = Z_0 + 2hf'(x+h, z_i) + O(h^2) \]

\[ \text{i.e. } f'(x+h, z_i) - f'(x, z_0) = O(h^3) \]

\[ \Rightarrow h \left[ f'(x+h, z_i) + f'(x, z_0) \right] = \]

\[ = h \left[ 2f'(x+h, z_i) + O(h^3) \right] \]

\[ = 2hf'(x+h, z_i) + O(h^3) \]

\[ \text{lost because dominant error is } O(h^2) \]
Modified Midpoint Method

We basically advance the solution from an initial value $y(x)$ to a point $x + h$ where

$$H = nh$$

$n$ steps are taken to go from $x$ to $x + nh$ (see fig).

Thus $h$ is our constant step.

Then the algorithm is a generalization of the midpoint except at the edges: i.e

$Z_0 = y(x)$

$Z_1 = Z_0 + hf'(x, Z_0)$

$Z_{m+1} = Z_{m-1} + 2hf'(x + mh, Z_m)$ with $m = 1, \ldots, n-1$

and for the final point

$$y_n(x + nh) \approx y_n = \frac{1}{2} \left[ Z_n + Z_{n-1} + hf'(x + nh, Z_n) \right]$$

Since this is basically a midpoint method except at the first and last points, it is called "modified" midpoint method.

As a result, this is a 2nd-order method but for large $n$ has the advantage of a single derivative evaluation per step as compared to the two per step of the 2nd-order RK.
\[ y(n + H) = y_n + O(h^2) \]
\[ y(n + H) = y_{n/2} + O((2h)^2) = y_{n/2} + 4O(h^2) \]

\( \Rightarrow \) multiply first line by 4 and subtract second line

\[ 4y(n + H) = 4y_n + 4O(h^2) + O(h^4) \]
\[ y(n + H) = y_{n/2} + 4O(h^2) + O(h^4) \]

\( \Rightarrow \) \[ 3y(n + H) = 4y_n - y_{n/2} + O(h^4) \]

\( \Rightarrow \) \[ y(n + H) = \frac{4y_n - y_{n/2}}{3} + O(h^4) \]
The importance of the MMM is that the error terms come in even powers of $h$ so that removing an order gives a gain of 2-orders

$$y(x+h) - y_n = \sum \alpha_i h^{2i} \quad (\text{cf. trapezoidal rule for quadrature})$$

so that, for example,

$$y(x+h) \approx \frac{1}{3} \sum_{i=0}^{n} y_{2i} - y_{2i-2} + o(h^4)$$

$\frac{1}{3}$ steps

$\sum_{i=0}^{n}$ steps

ie this is 4th-order but we take less (3/2 per step $h$) derivatives.

The MMM finds its best application in the Burrisch-Shamp Method where this method is used together with Richardson extrapolation to get the final answer at $x+H$
In the BS method one follows a sequence of \( n \) points

\[ n = 2, 4, \ldots, 2^j \]

and after each successive \( n \) is tried, an extrapolation is performed and the error compared with the required accuracy.

If the latter is reached, one moves to the following \( x + h \).
Numerical Methods for the Solution of Partial Differential Equations

Luciano Rezzolla

Institute for Theoretical Physics,
Frankfurt, Germany

June 7, 2019
## Contents

1 Introduction 3  
  1.1 Discretization of differential operators and variables . . . . . . . . . . 4  
  1.2 Errors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5  
    1.2.1 Machine-precision error . . . . . . . . . . . . . . . . . . . . 5  
    1.2.2 Round-off error . . . . . . . . . . . . . . . . . . . . . . . . 6  
    1.2.3 Truncation error . . . . . . . . . . . . . . . . . . . . . . . . 6  
2 Hyperbolic PDEs: Flux Conservative Formulation 9  
3 The advection equation in one dimension (1D) 11  
  3.1 The 1D Upwind scheme: $O(\Delta t, \Delta x)$ . . . . . . . . . . . . . . . . 11  
  3.2 The 1D FTCS scheme: $O(\Delta t, \Delta x^2)$ . . . . . . . . . . . . . . . . 16  
  3.3 The 1D Lax-Friedrichs scheme: $O(\Delta t, \Delta x^2)$ . . . . . . . . . . . 18  
  3.4 The 1D Leapfrog scheme: $O(\Delta t^2, \Delta x^2)$ . . . . . . . . . . . . . . 21  
  3.5 The 1D Lax-Wendroff scheme: $O(\Delta t^2, \Delta x^2)$ . . . . . . . . . . . 23  
  3.6 The 1D ICN scheme: $O(\Delta t^2, \Delta x^2)$ . . . . . . . . . . . . . . . . 25  
    3.6.1 ICN as a $\theta$-method . . . . . . . . . . . . . . . . . . . . . 27  
    3.6.2 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31  
    3.6.3 Finite-difference stencils . . . . . . . . . . . . . . . . . . . . . . 32  
4 Dissipation, Dispersion and Convergence 35  
  4.1 On the Origin of Dissipation and Dispersion . . . . . . . . . . . . . . . . 35  
  4.2 Measuring Dissipation and Convergence . . . . . . . . . . . . . . . . . . 39  
    4.2.1 The summarising power of norms . . . . . . . . . . . . . . . . . . 39  
    4.2.2 Consistency and Convergence . . . . . . . . . . . . . . . . . . . . 40  
    4.2.3 Convergence and Stability . . . . . . . . . . . . . . . . . . . . . . 43  
5 The Wave Equation in 1D 45  
  5.1 The FTCS Scheme . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46  
  5.2 The Lax-Friedrichs Scheme . . . . . . . . . . . . . . . . . . . . . . . . . . . 47  
  5.3 The Leapfrog Scheme . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48  
  5.4 The Lax-Wendroff Scheme . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
Acknowledgments

I am indebted to the several students who have helped me with the typing of the lectures notes into at TeXformat. They are too numerous to be reported here but my special thanks go to Olindo Zanotti for his help with the hyperbolic equations and to Gregor Leiler for his help with the parabolic equations and Chapter 4.
Chapter 1

Introduction

Let us consider a partial differential equation (PDE) of second-order in two dimensions \((x, y)\), which we can write generically as

\[
a_{11} \frac{\partial^2 u}{\partial x^2} + 2a_{12} \frac{\partial^2 u}{\partial x \partial y} + a_{22} \frac{\partial^2 u}{\partial y^2} + f(x, y, u) = 0, \tag{1.1}
\]

where \(x, y\) are not all spatial coordinates and where we will assume the coefficients \(a_{ij}\) to be functions of position only, i.e., \(a_{ij} = a_{ij}(x, y)\). The PDE (1.1) is then said to be "linear with variable coefficients"\(^1\).

The traditional classification of partial differential equations is then based on the sign of the determinant \(\Delta \equiv a_{11}a_{22} - a_{12}^2\) that we can build with the coefficients of equation (1.1) and distinguishes three types of such equations. More specifically, equation (1.1) will be (strictly) hyperbolic if \(\Delta = 0\) has roots that are real (and distinct), parabolic if \(\Delta = 0\) has real but zero roots, while it will be elliptic if \(\Delta = 0\) has complex roots (see Table 1.1).

Elliptic equations, on the other hand, describe boundary value problems, or BVP, since the space of relevant solutions \(\bar{\Omega}\) depends on the value that the solution takes on its boundaries \(d\bar{\Omega}\). Elliptic equations are easily recognizable by the fact the solution does not depend on time coordinate \(t\) and a prototype elliptic equation is in fact given by Poisson equation (cf. Table 1.1).

Hyperbolic and parabolic equations describe initial value boundary problems, or IVBP, since the space of relevant solutions \(\bar{\Omega}\) depends on the value that the solution \(L\) (which we assume with compact support) takes on some initial time (see upper panel of Fig. 1.1). In practice, IVBP problems are easily recognizable by the fact that the solution will depend on the time coordinate \(t\). Very simple and useful examples of hyperbolic and parabolic equations are given by the wave equation and by the diffusion equation, respectively (cf. Table 1.1). An important and physically-based difference between hyperbolic and parabolic equations becomes apparent by considering the “characteristic velocities” associated to them. These represent the velocities at which perturbations are propagated and have \textit{finite} speeds in the case of hyperbolic equations,

\(^1\)The PDE (1.1) is said to be ”quasi-linear” if \(a_{ij} = a_{ij}(x, y, u)\)
CHAPTER 1. INTRODUCTION

<table>
<thead>
<tr>
<th>Type</th>
<th>Condition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperbolic</td>
<td>$a_{11}a_{22} - a_{12}^2 &lt; 0$</td>
<td>Wave equation: $\frac{\partial^2 u}{\partial t^2} = v^2 \frac{\partial^2 u}{\partial x^2}$</td>
</tr>
<tr>
<td>Parabolic</td>
<td>$a_{11}a_{22} - a_{12}^2 = 0$</td>
<td>Diffusion equation: $\frac{\partial u}{\partial t} = \frac{\partial}{\partial x} \left( D \frac{\partial u}{\partial x} \right)$</td>
</tr>
<tr>
<td>Elliptic</td>
<td>$a_{11}a_{22} - a_{12}^2 &gt; 0$</td>
<td>Poisson equation: $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \rho(x, y)$</td>
</tr>
</tbody>
</table>

Table 1.1: Schematic classification of a quasi-linear partial differential equation of second-order. For each class, a prototype equation is presented.

while these speeds are \textit{infinite} in the case of parabolic equations. In this way it is not difficult to appreciate that while both hyperbolic and parabolic equations describe time-dependent equations, the domain of dependence in a finite time for the two classes of equations can either be finite (as in the case of hyperbolic equations), or infinite (as in the case of parabolic equations).

1.1 Discretization of differential operators and variables

Consider, for simplicity, a generic one-dimensional IVBP that could be written as

$$L(u) - f = 0,$$  \hspace{1cm} (1.2)

where $u = u(x, t)$ and $L$ is a differential operator in the two variables $x$ and $t$ acting on $u$. One of the most used methods for the solution of such a problem is by means of \textit{finite differences}. It consists in two “discretization steps”:

- \textit{Variables discretization}: replace the function $u(x, t)$ with a discrete set of values $\{u^n_j\}$ that should approximate the pointwise values of $u$, i.e., $u^n_j \approx u(x_j, t_n)$;

- \textit{Operator discretization}: replace the continuous differential operator $L$ with a discretized one, $L_\Delta$, that when applied to the set $\{u^n_j\}$, gives an approximation to $L(u)$ in terms of differences between the various $u^n_j$.

The set of values $\tilde{u} \equiv \{u^n_j, j = 1, \ldots, J, n = 1, \ldots, N\}$ ($J$ and $N$ are the number of points considered for the space and time variable respectively) is called the grid function and will be denoted by $\tilde{u}$. After this discretization process, the problem (1.2) is replaced by

$$L_\Delta (\tilde{u}) - \tilde{f} = 0 + \epsilon_T,$$  \hspace{1cm} (1.3)

that is, a discrete representation of both the differential operator $L$ and of the variable $u$. The above equation is the \textit{discrete representation} of the problem (1.2). Note that the righ-hand-side of (1.3) is not exactly zero and it differs from it by the \textit{truncation error} $\epsilon_T$, which will be introduced in Sect. 1.2.3.
1.2. ERRORS

In the following Sections 2–7 we will concentrate on partial differential equations of hyperbolic type. Before doing that, however, it is useful to discretize the continuum space of solutions (a “spacetime” in the case of IVPBs) in spatial foliations such that the time coordinate \( t \) is constant on each slice. As shown in the lower panel of Fig. 1.1, each point \( P(x_j, t^n) \) in this discretized spacetime will have spatial and time coordinate defined as

\[
x_j = x_0 + j \Delta x, \quad j = 0, \pm 1, \ldots, \pm J,
\]

\[
t^n = t^0 + n \Delta t, \quad n = 0, \pm 1, \ldots, \pm N,
\]

(1.4)

where \( \Delta t \) and \( \Delta x \) are the increments between two spacelike and timelike foliations, respectively. In this way we can associate a generic solution \( u(x, t) \) in the continuum spacetime to a set of discretized solutions \( u^m_i \equiv u(x_i, t^m) \) with \( i = \pm I, \ldots, \pm 1, 0 \) and \( m = \pm M, \ldots, \pm 1, 0 \) and \( I \leq J; \ M \leq N \). Clearly, the number of discrete solutions to be associated to \( u(x, t) \) will depend on the properties of the discretized spacetime (i.e., on the increments \( \Delta t \) and \( \Delta x \)) which will also determine the truncation error introduced by the discretization.

Once a discretization of the spacetime is introduced, finite difference techniques offer a very natural way to express a partial derivative (and hence a partial differential equation). The basic idea behind these techniques is that the solution of the differential equation \( u(x_j, t^n + \Delta t) \) at a given position \( x_j \) and at a given time \( t^n \) can be Taylor-expanded in the vicinity of \((x, t^n)\). Under this simple (and most often reasonable assumption), differential operators can be substituted by properly weighted differences of the solution evaluated at different points in the numerical grid. In the following Section we will discuss how different choices in the way the finite-differencing is made will lead to numerical algorithms with different properties.

1.2 Errors

Errors are a natural and inevitable heritage of numerical analysis and their presence is not a nuisance as long their origing is well determined and under control. Three main errors will be discussed repeatedly in these notes and we briefly discuss them below.

1.2.1 Machine-precision error

The machine-precision error reflects the precision of the machine used and can be expressed in terms of the equality

\[
fp(1.0) = fp(1.0) + \epsilon_m,
\]

(1.5)

where \( fp(1.0) \) is the floating-point description of the number 1. Stated differently, the machine-precision error reflects the ability of the machine to distinguish two floating point numbers and is therefore related to the number of significant figures used in the mantissa.
1.2.2 Round-off error

The round-off error is the accumulation of machine-precision errors as a result of \( N \) floating point operations. Because of the random nature in which machine-precision errors add-up, this error can be estimated to be

\[
\epsilon_{\text{RO}} \approx \sqrt{N}\epsilon_m .
\]

(1.6)

Clearly, when performing a numerical computation one should restrict the number of operations such that \( \epsilon_{\text{RO}} \) is below the error at which the results needs to be determined.

1.2.3 Truncation error

The truncation error is fundamentally different from the previous two types of errors in that it is not dependent on the machine used but it reflects the human decision made in discretizing the continuum problem. Mathematically it can therefore be expressed as

\[
L(u) - f = L_\Delta(\tilde{u}) - \tilde{f} + \epsilon_T .
\]

(1.7)

Since the truncation error is totally under the human judgment, its measure is essential to guarantee that the discretization operation has been made properly and that the discretized problem is therefore a faithful representation of the continuum one, modulo the truncation error.
1.2. ERRORS

Figure 1.1: Upper panel: Schematic distinction between IVBPs and BVPs. Lower panel: Schematic discretization of a hyperbolic IVBP.
Chapter 2

Hyperbolic PDEs: Flux Conservative Formulation

It is often the case, when dealing with hyperbolic equations, that they can be formulated through conservation laws stating that a given quantity "u" is transported in space and time and is thus locally "conserved". The resulting "law of continuity" leads to equations which are called conservative and are of the type

\[ \frac{\partial u}{\partial t} + \nabla \cdot F(u) = 0 , \tag{2.1} \]

where \( u(x, t) \) is the density of the conserved quantity, \( F \) the density flux and \( x \) a vector of spatial coordinates. In most of the physically relevant cases, the flux density \( F \) will not depend explicitly on \( x \) and \( t \), but only implicitly through the density \( u(x, t) \), i.e., \( F = F(u(x, t)) \). The vector \( F \) is also called the conserved flux and takes this name from the fact that in the integral formulation of the conservation equation (2.1), the time variation of the integral of \( u \) over the volume \( V \) is indeed given by the net flux of \( u \) across the surface enclosing \( V \).

Generalizing expression (2.1), we can consider a vector of densities \( U \) and write a set of conservation equations in the form

\[ \frac{\partial U}{\partial t} + \nabla \cdot F(U) = S(U) . \tag{2.2} \]

Here, \( S(U) \) is a generic "source term" indicating the sources and sinks of the vector \( U \). The main property of the homogeneous equation (2.2) (i.e., when \( S(U) = 0 \)) is that the knowledge of the state-vector \( U(x, t) \) at a given point \( x \) at time \( t \) allows to determine the rate of flow, or flux, of each state variable at \( (x, t) \).

Conservation laws of the form given by (2.1) can also be written as a quasi-linear form

\[ \frac{\partial U}{\partial t} + A(U) \frac{\partial U}{\partial x} = 0 , \tag{2.3} \]

where \( A(U) \equiv \partial F/\partial U \) is the Jacobian of the flux vector \( F(U) \).
The use of a conservation form of the equations is particularly important when dealing with problems admitting shocks or other discontinuities in the solution, e.g., when solving the hydrodynamical equations. A non-conservative method, i.e., a method in which the equations are not written in a conservative form, might give a numerical solution which appears perfectly reasonable but then yields incorrect results. A well-known example is offered by Burger’s equation, i.e., the momentum equation of an isothermal gas in which pressure gradients are neglected, and whose non-conservative representation fails dramatically in providing the correct shock speed if the initial conditions contain a discontinuity. Moreover, since the hydrodynamical equations follow from the physical principle of conservation of mass and energy-momentum, the most obvious choice for the set of variables to be evolved in time is that of the conserved quantities. It has been proved that non-conservative schemes do not converge to the correct solution if a shock wave is present in the flow, whereas conservative numerical methods, if convergent, do converge to the weak solution of the problem.

In the following, we will concentrate on numerical algorithms for the solution of hyperbolic partial differential equations written in the conservative form of equation (2.2). The advection and wave equations can be considered as prototypes of this class of equations in which with \( S(U) = 0 \) and will be used hereafter as our working examples.
Chapter 3

The advection equation in one dimension (1D)

A special class of conservative hyperbolic equations are the so-called advection equations, in which the time derivative of the conserved quantity is proportional to its spatial derivative. In these cases, $F(U)$ is diagonal and given by

$$F(U) = vI \cdot U ,$$

(3.1)

where $I$ is the identity matrix.

Because in this case the finite-differencing is simpler and the resulting algorithms are easily extended to more complex equations, we will use it as our “working example”. More specifically, the advection equation for $u$ we will consider hereafter has, in 1D, the simple expression

$$\frac{\partial u}{\partial t} + v \frac{\partial u}{\partial x} = 0 ,$$

(3.2)

and admits the general analytic solution $u = f(x - vt)$, representing a wave moving in the positive $x$-direction.

3.1 The 1D Upwind scheme: $O(\Delta t, \Delta x)$

We will start making use of finite-difference techniques to derive a discrete representation of equation (3.2) by first considering the derivative in time. Taylor expanding the solution around $(x_j, t^n)$ we obtain

$$u(x_j, t^n + \Delta t) = u(x_j, t^n) + \frac{\partial u}{\partial t}(x_j, t^n) \Delta t + O(\Delta t^2) ,$$

(3.3)

or, equivalently,

$$u_j^{n+1} = u_j^n + \frac{\partial u}{\partial t} |_{x_j}^{t^n} \Delta t + O(\Delta t^2) .$$

(3.4)
Isolating the time derivative and dividing by $\Delta t$ we obtain

$$\frac{\partial u}{\partial t} \bigg|_j^n = \frac{u^{n+1}_j - u^n_j}{\Delta t} + \mathcal{O}(\Delta t) .$$  \hspace{1cm} (3.5)

Adopting a standard convention, we will consider the finite-difference representation of an $m$-th order differential operator $\frac{\partial^m}{\partial x^m}$ in the generic $x$-direction (where $x$ could either be a time or a spatial coordinate) to be of order $p$ if and only if

$$\frac{\partial^m u}{\partial x^m} = \mathcal{L}_\Delta(u) + \mathcal{O}(\Delta x^p) .$$  \hspace{1cm} (3.6)

Of course, the time and spatial operators may have finite-difference representations with different orders of accuracy and in this case the overall order of the equation is determined by the differential operator with the largest truncation error.

Note also that while the truncation error is expressed for the differential operator, the numerical algorithms will not be expressed in terms of the differential operators and will therefore have different (usually smaller) truncation errors. This is clearly illustrated by the equations above, which show that the explicit solution (3.4) is of higher order than the finite-difference expression for the differential operator (3.5).

With this definition in mind, it is not difficult to realize that the finite-difference expression (3.5) for the time derivative is only first-order accurate in $\Delta t$. However, accuracy is not the most important requirement in numerical analysis and a first-order but stable scheme is greatly preferable to one which is higher order (i.e., has a smaller truncation error) but is unstable.

In way similar to what we have done in (3.5) for the time derivative, we can derive a first-order, finite-difference approximation to the space derivative as

$$\frac{\partial u}{\partial x} \bigg|_j^n = \frac{u^n_{j+1} - u^n_{j-1}}{\Delta x} + \mathcal{O}(\Delta x) .$$  \hspace{1cm} (3.7)

While formally similar, the approximation (3.7) suffers of the ambiguity, not present in expression (3.5), that the first-order term in the Taylor expansion can be equally expressed in terms of $u^n_{j+1}$ and $u^n_{j-1}$, i.e.,

$$\frac{\partial u}{\partial x} \bigg|_j^n = \frac{u^n_{j+1} - u^n_{j}}{\Delta x} + \mathcal{O}(\Delta x) .$$  \hspace{1cm} (3.8)

This ambiguity is the consequence of the first-order approximation which prevents a proper “centring” of the finite-difference stencil. However, and as long as we are concerned with an advection equation, this ambiguity is easily solved if we think that the differential equation will simply translate each point in the initial solution to the new position $x + v \Delta t$ over a time interval $\Delta t$. In this case, it is natural to select the points in the solution at the time-level $n$ that are “upwind” of the solution at the position $j$ and at the time-level $n+1$, as these are the ones causally connected with $u^{n+1}_j$. Depending then on the direction in which the solution is translated, and hence
3.1. THE 1D UPWIND SCHEME: $O(\Delta T, \Delta X)$

![Diagram showing the 1D upwind scheme](image)

Figure 3.1: Schematic diagram of an UPWIND evolution scheme.

on the value of the advection velocity $v$, two different finite-difference representations can be given of equation (3.2) and these are

1. If $v > 0$:
   \[
   \frac{u_j^{n+1} - u_j^n}{\Delta t} = -v \left( \frac{u_j^n - u_{j-1}^n}{\Delta x} \right) + O(\Delta t, \Delta x), \quad \text{if } v > 0, \quad (3.9)
   \]

2. If $v < 0$:
   \[
   \frac{u_j^{n+1} - u_j^n}{\Delta t} = -v \left( \frac{u_{j+1}^n - u_j^n}{\Delta x} \right) + O(\Delta t, \Delta x), \quad \text{if } v < 0, \quad (3.10)
   \]

respectively. As a result, the final finite-difference algorithms for determining the solution at the new time-level will have the form

1. If $v > 0$:
   \[
   u_j^{n+1} = u_j^n - \frac{v \Delta t}{\Delta x} (u_j^n - u_{j-1}^n) + O(\Delta t^2, \Delta x \Delta t), \quad \text{if } v > 0, \quad (3.11)
   \]

2. If $v < 0$:
   \[
   u_j^{n+1} = u_j^n - \frac{v \Delta t}{\Delta x} (u_{j+1}^n - u_j^n) + O(\Delta t^2, \Delta x \Delta t), \quad \text{if } v < 0. \quad (3.12)
   \]
More in general, for a system of linear hyperbolic equations with state vector $U$ and flux-vector $F$, the upwind scheme will take the form
\[
U_j^{n+1} = U_j^n \pm \frac{\Delta t}{\Delta x} \left[ F_{j+1}^n - F_j^n \right] + \mathcal{O}(\Delta t^2, \Delta x \Delta t), \tag{3.13}
\]
where the ± sign should be chosen according to whether $v > 0$ or $v < 0$. The logic behind the choice of the stencil in an upwind method is illustrated in Fig. 1.1 where we have shown a schematic diagram for the two possible values of the advection velocity.

The upwind scheme (as well as all of the others we will consider here) is an example of an explicit scheme, that is of a scheme where the solution at the new time-level $n + 1$ can be calculated explicitly from the quantities that are already known at the previous time-level $n$. This is to be contrasted with an implicit scheme in which the finite-difference representations of the differential equation has, on the right-hand-side, terms at the new time-level $n + 1$. These methods require in general the solution of a number of coupled algebraic equations and will not be discussed further here.

The upwind scheme is a stable one in the sense that the solution will not have exponentially growing modes. This can be seen through a von Neumann stability analysis, a useful tool which allows a first simple validation of a given numerical scheme. It is important to underline that the von Neumann stability analysis is local in the sense that: 

a) it does not take into account boundary effects; b) it assumes that the coefficients of the finite difference equations are sufficiently slowly varying to be considered constant in time and space (this is a reasonable assumptions if the equations are linear). Under these assumptions, the solution can be seen as a sum of eigenmodes which at each grid point have the form
\[
u_j^n = \xi^n e^{ikx_j}, \tag{3.14}
\]
where $k$ is the spatial wave number and $\xi = \xi(k)$ is a complex number.

If we now consider the symbolic representation of the finite difference equation as
\[
u_j^{n+1} = T(\Delta t^p, \Delta x^q)\nu_j^n, \tag{3.15}
\]
with $T(\Delta t^p, \Delta x^q)$ being the evolution operator of order $p$ in time and $q$ in space, it then becomes clear from (3.14) and (3.15) that the time evolution of a single eigenmode is nothing but a succession of integer powers of the complex number $\xi$ which is therefore named amplification factor. This naturally leads to a criterion of stability as the one for which the modulus of the amplification factor is always less than 1, i.e.,
\[
|\xi|^2 = \xi \xi^* \leq 1. \tag{3.16}
\]

Using (3.14) in (3.11)–(3.12) we would obtain an amplification factor
\[
\xi = 1 - |\alpha| (1 - \cos(k\Delta x)) - i\alpha \sin(k\Delta x), \tag{3.17}
\]
where
\[
\alpha \equiv \frac{v\Delta t}{\Delta x}. \tag{3.18}
\]
Its squared modulus $|\xi|^2 \equiv \xi \xi^*$ is then
\[
|\xi|^2 = 1 - 2 |\alpha| (1 - |\alpha|) (1 - \cos(k\Delta x)) \tag{3.19}
\]
3.1. THE 1D UPWIND SCHEME: $O(\Delta T, \Delta X)$

so that the amplification factor (3.19) is less than one as long as the Courant-Friedrichs-Löwy condition (CFL condition)

$$|\alpha| \leq 1,$$

is satisfied (condition (3.20) is sometimes referred to simply as the Courant condition.). Note that in practice, the CFL condition (3.20) is used to determine the time-step $\Delta t$ once $v$ is known and $\Delta x$ has been chosen to achieve a certain accuracy, i.e.,

$$\Delta t = c_{\text{CFL}} \frac{\Delta x}{|v|},$$

with $c_{\text{CFL}} < 1$ being the CFL factor. Expression (3.21) also allows a useful interpretation of the CFL condition.

From a mathematical point of view, the condition ensures that the numerical domain of dependence of the solution is larger than the physical one. From a physical point of view, on the other hand, the condition ensures that the propagation speed of any physical perturbation (e.g., the sound speed, or the speed of light) is always smaller than the numerical one $v_n \equiv \Delta x/\Delta t$, i.e.,

$$|v| = c_{\text{CFL}} \frac{\Delta x}{\Delta t} \leq v_n \equiv \frac{\Delta x}{\Delta t}.$$

Equivalently, the CFL conditions prevents any physical signal to propagate for more than a fraction of a grid-zone during a single time-step (cf. Fig. 3.2).

As a final remark it should be noted that as described so far, the upwind method will yield satisfactory results only in the case in which the equations have an obvious transport character in one direction. However, in more general situations such as a wave equation, the upwind method will not be adequate and different expressions, based on finite-volume formulations of the equations will be needed [1, 4].

Figure 3.2: Schematic diagram of Courant stable and unstable choices of time-steps $\Delta t$. The two dashed lines limit the numerical domain of dependence of the solution at $x^{n+1}$, while the shaded area represents the physical domain of dependence. Stability is achieved when the first one is larger than the second one.
3.2 The 1D FTCS scheme: $O(\Delta t, \Delta x^2)$

Let us consider again the advection equation (3.2) but we now finite difference with a more accurate approximation of the space derivative. To do this we can calculate the two Taylor expansions in $x_j \pm \Delta x$

\[
\begin{align*}
  u(x_j + \Delta x, t^n) &= u(x_j, t^n) + \frac{\partial u}{\partial x}(x_j, t^n)\Delta x + \frac{1}{2} \frac{\partial^2 u}{\partial x^2}(x_j, t^n)(\Delta x)^2 + O((\Delta x)^3), \\
  u(x_j - \Delta x, t^n) &= u(x_j, t^n) - \frac{\partial u}{\partial x}(x_j, t^n)\Delta x + \frac{1}{2} \frac{\partial^2 u}{\partial x^2}(x_j, t^n)(\Delta x)^2 + O((\Delta x)^3),
\end{align*}
\]

(3.23) (3.24)
3.2. **THE 1D FTCS SCHEME:** \( O(\Delta T, \Delta X^2) \)

Subtracting now the two expressions and dividing by \( 2\Delta x \) we eliminate the first-order terms and obtain

\[
\frac{\partial u^n}{\partial x} \bigg|_j = \frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} + O(\Delta x^2), \quad (3.25)
\]

![FTCS](image)

Figure 3.4: Schematic diagram of a FTCS evolution scheme.

Using now the second-order accurate operator (3.25) we can finite-difference equation (3.2) through the so called FTCS (Forward-Time-Centered-Space) scheme in which a first-order approximation is used for the time derivative, but a second order one for the spatial one. Using the a finite-difference notation, the FTCS is then expressed as

\[
\frac{u^{n+1}_j - u^n_j}{\Delta t} = -v \left( \frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} \right) + O(\Delta t, \Delta x^2), \quad (3.26)
\]

so that

\[
u^{n+1}_j = u^n_j - \frac{\alpha}{2} \left( u_{j+1}^n - u_{j-1}^n \right) + O(\Delta t^2, \Delta x^2 \Delta t), \quad (3.27)
\]

or more generically, for a system of linear hyperbolic equations

\[
U^{n+1}_j = U^n_j - \frac{\Delta t}{2\Delta x} \left[ F^n_{j+1} - F^n_{j-1} \right] + O(\Delta t^2, \Delta x^2 \Delta t), \quad (3.28)
\]

The stencil for the finite-differencing (3.27) is shown symbolically in Fig. 3.4.

Disappointingly, the FTCS scheme is **unconditionally unstable**: i.e., the numerical solution will be destroyed by numerical errors which will be certainly produced and grow exponentially. This is shown in Fig. 3.5 where we show the time evolution of a Gaussian using an FTCS scheme 100 gridpoints. The analytic solution at time \( t = 0.3 \) is shown with a solid line the dashed lines are used to represent the numerical solution at the same time. Note that the solution plotted here refers to a time which is 10 times smaller than the one in Fig. 3.3. Soon after \( t \approx 0.3 \) the exponentially growing modes appear, rapidly destroying the solution.

Applying the definition (3.14) to equation (3.27) and few algebraic steps lead to an amplification factor

\[
\xi = 1 - i\alpha \sin(k \Delta x). \quad (3.29)
\]
whose squared modulus is
\[ |\xi|^2 = 1 + (\alpha \sin(k \Delta x))^2 > 1 , \]
thus proving the unconditional instability of the FTCS scheme. Because of this, the FTCS scheme is rarely used and will not produce satisfactory results but for a very short timescale as compared to the typical crossing time of the physical problem under investigation.

A final aspect of the von Neumann stability worth noticing is that it is a necessary but not sufficient condition for stability. In other words, a numerical scheme that appears stable with respect to a von Neumann stability analysis might still be unstable.

### 3.3 The 1D Lax-Friedrichs scheme: \( \mathcal{O}(\Delta t, \Delta x^2) \)

A solution to the stability problems offered by the FTCS scheme was proposed by Lax and Friedrichs. The basic idea is very simple and is based on replacing, in the FTCS
3.3. THE 1D LAX-FRIEDRICHS SCHEME: $O(\Delta T, \Delta X^2)$

\[ u_j^{n+1} = \frac{1}{2} (u_{j+1}^n + u_{j-1}^n) - \frac{\alpha}{2} (u_{j+1}^n - u_{j-1}^n) + O(\Delta x^2), \quad (3.31) \]

and, for a system of linear hyperbolic equations

\[ U_j^{n+1} = \frac{1}{2} (U_{j+1}^n + U_{j-1}^n) - \frac{\Delta t}{2\Delta x} [F_{j+1}^n - F_{j-1}^n] + O(\Delta x^2). \quad (3.32) \]

Note that the truncation error in equations (3.31) and (3.32) is reported to be $O(\Delta x^2)$ and not $O(\Delta t^2, \Delta x^2\Delta t)$ because we are assuming that the CFL condition is satisfied and hence $\Delta t = O(\Delta x)$. We will maintain this assumption hereafter.

The schematic diagram of a Lax-Friedrichs evolution scheme is shown in Fig. 3.6. Perhaps surprisingly, the algorithm (3.32) is now conditionally stable as can be verified through a von Neumann stability analysis. Proceeding as done for the FTCS scheme and using (3.14) in (3.32) we would obtain an amplification factor whose modulus squared is

\[ |\xi|^2 = 1 - \sin^2(k\Delta x) \left(1 - \alpha^2\right), \quad (3.33) \]

which is $< 1$ as long as the CFL condition is satisfied.

Although not obvious, the correction introduced by the Lax-Friedrichs scheme is equivalent to the introduction of a numerical dissipation (viscosity). To see this, we rewrite (3.32) so that it clearly appears as a correction to (3.27):

\[ \frac{u_j^{n+1} - u_j^n}{\Delta t} = -v \left( \frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} \right) + \frac{1}{2} \left( \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta t} \right). \quad (3.34) \]

This is exactly the finite-difference representation of the equation

\[ \frac{\partial u}{\partial t} + v \frac{\partial u}{\partial x} = \frac{1}{2} \left( \frac{\Delta x^2}{\Delta t} \right) \frac{\partial^2 u}{\partial x^2}, \quad (3.35) \]

where a diffusion term, $\propto \frac{\partial^2 u}{\partial x^2}$, has appeared on the right hand side. To prove this we sum the two Taylor expansions (3.23)–(3.24) around $x_j$ to eliminate the first-order derivatives and obtain

\[ \left. \frac{\partial^2 u}{\partial x^2} \right|_j^n = \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2} + O(\Delta x^2), \quad (3.36) \]
where the sum has allowed us to cancel both the terms $O(\Delta x)$ and $O(\Delta x^3)$. Note that since the expression for the second derivative in (3.36) is $O(\Delta x^2)$, it is appears multiplied by $\Delta x^2/\Delta t = O(\Delta x)$ in equation (3.35), thus making the right-hand-side $O(\Delta x^3)$ overall. The left-hand-side, on the other hand, is only $O(\Delta x)$ (the time derivative is $O(\Delta x)$, while the spatial derivative is $O(\Delta x^2)$). As a result, the dissipative term goes to zero more rapidly than the intrinsic truncation error of the Lax-Friedrichs scheme, thus guaranteeing that the in the continuum limit the algorithm will converge to the correct solution of the advection equation.

![Figure 3.7](image)

**Figure 3.7**: This is the same as in Fig. 3.3 but for a Lax-Friedrichs scheme. Note how the scheme is stable but also suffers from a considerable dissipation.

A reasonable objection could be made for the fact that the Lax-Friedrichs scheme has changed the equation whose solution one is interested in [i.e., eq. (3.2)] into a new equation, in which a spurious numerical dissipation has been introduced [i.e., eq. (3.35)]. Unless $|v|\Delta t = \Delta x$, $|\xi| < 1$ and the amplitude of the wave is doomed to decrease (see Fig. 3.7).

However, such objection can be easily circumvented. As mentioned above, the dissipative term is always smaller than the truncation error thus guaranteeing the convergence to the correct solution. Furthermore, it is useful to bear in mind that the key
aspect in any numerical representation of a physical phenomenon is the determination of the length scale over which we need to achieve an accurate description. In a finite difference approach, this length scale must necessarily encompass many grid points and for which \( k \Delta x \ll 1 \). In this case, expression (3.33) clearly shows that the amplification factor is very close to 1 and the effects of dissipation are therefore small. Note that this is true also for the FTCS scheme so that on these scales the stable and unstable schemes are equally accurate. On the very small scales however, which we are not of interest to us, \( k \Delta x \sim 1 \) and the stable and unstable schemes are radically different. The first one will be simply inaccurate, the second one will have exponentially growing errors which will rapidly destroy the whole solution. It is rather obvious that stability and inaccuracy are by far preferable to instability, especially if the accuracy is lost over wavelengths that are not of interest or when it can be recovered easily by using more refined grids. This is called “consistency” of the discretized operator and will be discussed in detail in Sect. 4.2.2.

3.4 The 1D Leapfrog scheme: \( O(\Delta t^2, \Delta x^2) \)

Both the FTCS and the Lax-Friedrichs are “one-level” schemes with first-order approximation for the time derivative and a second-order approximation for the spatial derivative. In those circumstances \( v \Delta t \) should be taken significantly smaller than \( \Delta x \) (to achieve the desired accuracy), well below the limit imposed by the Courant condition.

Second-order accuracy in time can be obtained if we insert

\[
\frac{\partial u}{\partial t} \bigg|_j^n = \frac{u_j^{n+1} - u_j^{n-1}}{2\Delta t} + O(\Delta t^2), \tag{3.37}
\]

in the FTCS scheme, to find the Leapfrog scheme

\[
u_j^{n+1} = u_j^{n-1} - \alpha (u_{j+1}^{n} - u_{j-1}^{n}) + O(\Delta x^2), \tag{3.38}
\]
where it should be noted that the factor 2 in $\Delta x$ cancels the equivalent factor 2 in $\Delta t$.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3_9.png}
\caption{This is the same as in Fig. 3.3 but for a Leapfrog scheme. Note how the scheme is stable and does not suffer from a considerable dissipation even for low CFL factors. However, the presence of a little "dip" in the tail of the Gaussian for the case of $c_{CFL} = 0.5$ is the result of the dispersive nature of the numerical scheme.}
\end{figure}

For a set of linear equations, the Leapfrog scheme simply becomes
\begin{equation}
U_{j}^{n+1} = U_{j}^{n-1} - \frac{\Delta t}{\Delta x} \left[ F_{j+1}^{n} - F_{j-1}^{n-1} \right] + \mathcal{O}(\Delta x^2),
\end{equation}
and the schematic diagram of a Leapfrog evolution scheme is shown in Fig. 3.8.

Also for the case of a Leapfrog scheme there are a number of aspects that should be noticed:

- In a Leapfrog scheme that is Courant stable, there is no amplitude dissipation \( (i.e., |\xi|^2 = 1) \). In fact, a von Neumann stability analysis yields
\begin{equation}
\xi = -i\alpha \sin(k\Delta x) \pm \sqrt{1 - [\alpha \sin(k\Delta x)]^2},
\end{equation}
and so that
\begin{equation}
|\xi|^2 = \alpha^2 \sin^2(k\Delta x) + \{1 - [\alpha \sin(k\Delta x)]^2\} = 1 \quad \forall \alpha \leq 1.
\end{equation}
As a result, the squared modulus of amplification factor is always 1, provided the CFL condition is satisfied (cf. Fig. 3.11).

- The Leapfrog scheme is a two-level scheme, requiring records of values at time-steps $n$ and $n-1$ to get values at time-step $n+1$. This is clear from expression (5.22) and cannot be avoided by means of algebraic manipulations.

- The major disadvantage of this scheme is that odd and even mesh points are completely decoupled (see Fig. 8).

In principle, the solutions on the black and white squares are identical. In practice, however, their differences increase as the time progresses. This effect, which becomes evident only on timescales much longer than the crossing timescale, can be cured either by discarding one of the solutions or by adding a dissipative term of the type

$$\ldots + \epsilon (u_{j+1}^n - 2u_{j+1}^{n+1} + u_{j+1}^{n+1}),$$

in the right-hand-side of (5.17), where $\epsilon \ll 1$ is an adjustable coefficient.

### 3.5 The 1D Lax-Wendroff scheme: $O(\Delta t^2, \Delta x^2)$

The Lax-Wendroff scheme is the second-order accurate extension of the Lax-Friedrichs scheme. As for the case of the Leapfrog scheme, in this case too we need two time-levels to obtain the solution at the new time-level.

There are a number of different ways of deriving the Lax-Wendroff scheme but it is probably useful to look at it as a combination of the Lax-Friedrichs scheme and of the Leapfrog scheme. In particular a Lax-Wendroff scheme can be obtained as
1. A Lax-Friedrichs scheme with half step:

\[
U_{n+1/2}^+ = \frac{1}{2} \left[ U_{j+1}^n + U_j^n \right] - \frac{\Delta t}{2\Delta x} \left[ F_{j+1}^n - F_j^n \right] + \mathcal{O}(\Delta x^2),
\]

\[
U_{n+1/2}^- = \frac{1}{2} \left[ U_j^n + U_{j-1}^n \right] - \frac{\Delta t}{2\Delta x} \left[ F_j^n - F_{j-1}^n \right] + \mathcal{O}(\Delta x^2),
\]

where \(\Delta t/(2\Delta x)\) comes from having used a timestep \(\Delta t/2\);

2. The evaluation of the fluxes \(F_{j+1/2}^n\) from the values of \(U_{n+1/2}^n\);

3. A Leapfrog “half-step”:

\[
U_{j+1}^{n+1} = U_j^n - \Delta t \frac{\Delta x}{2} \left[ F_{j+1/2}^{n+1/2} - F_{j-1/2}^{n+1/2} \right] + \mathcal{O}(\Delta x^2).
\]

The schematic diagram of a Lax-Wendroff evolution scheme is shown in Fig. 3.11 and the application of this scheme to the advection equation (3.2) is straightforward. More specifically, the “half-step” values can be calculated as

\[
u_{j+1/2}^{n+1/2} = \frac{1}{2} \left( u_j^n + u_{j+1}^n \right) \mp \frac{\alpha}{2} \left( u_{j+1}^n - u_j^n \right) + \mathcal{O}(\Delta x^2),
\]

so that the solution at the new time-level will then be

\[
u_j^{n+1} = \nu_j^n - \alpha \left( \nu_{j+1/2}^{n+1/2} - \nu_{j-1/2}^{n+1/2} \right) + \mathcal{O}(\Delta x^2)
\]

\[
= \nu_j^n - \frac{\alpha}{2} \left( \nu_{j+1}^n - \nu_{j-1}^n \right) + \frac{\alpha^2}{2} \left( \nu_{j+1}^n - 2\nu_j^n + \nu_{j-1}^n \right) + \mathcal{O}(\Delta x^2).
\]

where expression (3.46) has been obtained after substituting (3.44) in (3.45).

Aspects of a Lax-Wendroff scheme worth noticing are:
In the Lax-Wendroff scheme there might be some amplitude dissipation. In fact, a von Neumann stability analysis yields

\[ \xi = 1 - i \alpha \sin(k\Delta x) - \alpha^2 [1 - \cos(k\Delta x)] \] , \hspace{1cm} (3.47)

so that the squared modulus of the amplification factor is

\[ |\xi|^2 = 1 - \alpha^2 (1 - \alpha^2) [1 - \cos^2(k\Delta x)] \] . \hspace{1cm} (3.48)

As a result, the von Neumann stability criterion \(|\xi|^2 \leq 1\) is satisfied as long as \(\alpha^2 \leq 1\), or equivalently, as long as the CFL condition is satisfied. (cf. Fig. 10). It should be noticed, however, that unless \(\alpha^2 = 1\), then \(|\xi|^2 < 1\) and some amplitude dissipation is present. In this respect, the dissipative properties of the Lax-Friedrichs scheme are not completely lost in the Lax-Wendroff scheme but are much less severe (cf. Figs. 5 and 10).

The Lax-Wendroff scheme is a two-level scheme, but can be recast in a one-level form by means of algebraic manipulations. This is clear from expressions (3.46) where quantities at time-levels \(n\) and \(n+1\) only appear.

### 3.6 The 1D ICN scheme: \(O(\Delta t^2, \Delta x^2)\)

The idea behind the iterative Crank-Nicolson (ICN) scheme is that of transforming a stable implicit method, i.e., the Crank-Nicolson (CN) scheme (see Sect. 8.4.2) into an explicit one through a series of iterations. To see how to do this in practice, consider differencing the advection equation (3.2) having a centred space derivative but with the time derivative being backward centred, i.e.,

\[ \frac{u_j^{n+1} - u_j^n}{\Delta t} = -v \left( \frac{u_{j+1}^{n+1} - u_{j-1}^{n+1}}{2\Delta x} \right) . \] \hspace{1cm} (3.49)

This scheme is also known as “backward in time, centred in space” or BTCS (see Sect. 8.4.1) and has amplification factor

\[ \xi = \frac{1}{1 + i\alpha \sin k\Delta x} , \] \hspace{1cm} (3.50)

so that \(|\xi|^2 < 1\) for any choice of \(\alpha\), thus making the method unconditionally stable.

The Crank-Nicolson (CN) scheme, instead, is a second-order accurate method obtained by averaging a BTCS and a FTCS method or, in other words, equations (3.26) and (3.49). Doing so one then finds

\[ \xi = \frac{1 + i\alpha \sin k\Delta x/2}{1 - i\alpha \sin k\Delta x/2} \] \hspace{1cm} (3.51)

so that the method is stable. Note that although one averages between an explicit and an implicit scheme, terms containing \(u^{n+1}\) survive on the right hand side of equation (3.49), thus making the CN scheme implicit.
Figure 3.12: This is the same as in Fig. 3.3 but for a Lax-Wendroff scheme. Note how the scheme is stable and does not suffer from considerable dissipation even for low CFL factors. However, the presence of a little “dip” in the tail of the Gaussian for the case of $c_{\text{CFL}} = 0.5$ is the result of the dispersive nature of the numerical scheme.

The first iteration of iterative Crank-Nicolson starts by calculating an intermediate variable $(1)^{\tilde{u}}$ using equation (3.26):

$$
\frac{(1)^{\tilde{u}}_{j}^{n+1} - u_j^{n}}{\Delta t} = -v \left( \frac{u_{j+1}^{n} - u_{j-1}^{n}}{2\Delta x} \right) .
$$

Then another intermediate variable $(1)^{\bar{u}}$ is formed by averaging:

$$
(1)^{\bar{u}}_{j}^{n+1/2} := \frac{1}{2} \left( (1)^{\tilde{u}}_{j}^{n+1} + u_j^{n} \right) .
$$

Finally the timestep is completed by using equation (3.26) again with $\bar{u}$ on the right-hand side:

$$
\frac{u_{j}^{n+1} - u_j^{n}}{\Delta t} = -v \left( \frac{(1)^{\bar{u}}_{j+1}^{n+1/2} - (1)^{\bar{u}}_{j-1}^{n+1/2}}{2\Delta x} \right) .
$$
Iterated Crank-Nicolson with two iterations is carried out in much the same way. After steps (3.52) and (3.53), we calculate

\[
\frac{(2)\tilde{u}_{j}^{n+1} - u_{j}^{n}}{\Delta t} = -v \left( \frac{(1)\bar{u}_{j+1}^{n+1/2} - (1)\bar{u}_{j-1}^{n+1/2}}{2\Delta x} \right), \tag{3.55}
\]

Then the final step is computed analogously to equation (3.54):

\[
\frac{u_{j}^{n+1} - u_{j}^{n}}{\Delta t} = -v \left( \frac{(2)\bar{u}_{j+1}^{n+1/2} - (2)\bar{u}_{j-1}^{n+1/2}}{2\Delta x} \right). \tag{3.57}
\]

Further iterations can be carried out following the same logic.

To investigate the stability of these iterated schemes we compute the amplification factors relative to the different iterations to be

\[
(1)\xi = 1 + 2i\beta, \tag{3.58}
\]

\[
(2)\xi = 1 + 2i\beta - 2\beta^2, \tag{3.59}
\]

\[
(3)\xi = 1 + 2i\beta - 2\beta^2 - 2i\beta^3, \tag{3.60}
\]

\[
(4)\xi = 1 + 2i\beta - 2\beta^2 - 2i\beta^3 + 2\beta^4. \tag{3.61}
\]

where \(\beta := (\alpha/2)\sin(k\Delta x)\), and \((1)\xi\) corresponds to the FTCS scheme. Note that the amplification factors (3.58) correspond to those one would obtain by expanding equation (3.51) in powers of \(\beta\).

Computing the squared moduli of (3.58) one encounters an alternating and recursive pattern. In particular, iterations 1 and 2 are unstable (\(|\xi|^2 > 1\)); iterations 3 and 4 are stable (\(|\xi|^2 < 1\) provided \(\beta^2 \leq 1\)); iterations 5 and 6 are also unstable; iterations 7 and 8 are stable provided \(\beta^2 \leq 1\); and so on. Imposing the stability for all wavenumbers \(k\), we obtain \(\alpha^2/4 \leq 1\), or \(\Delta t \leq 2\Delta x\) which is just the CFL condition [the factor 2 is inherited by the factor 2 in equation (3.26)].

In other words, while the magnitude of the amplification factor for iterated Crank-Nicolson does approach 1 as the number of iterations becomes infinite, the convergence is not monotonic. The magnitude oscillates above and below 1 with ever decreasing oscillations. All the iterations leading to \(|\xi|^2\) above 1 are unstable, although the instability might be very slowly growing as the number of iterations increases. Because the truncation error is not modified by the number of iterations and is always \(O(\Delta t^2, \Delta x^2)\), a number of iterations larger than two is never useful; three iterations, in fact, would simply amount to a larger computational cost.

### 3.6.1 ICN as a \(\theta\)-method

In the ICN method the \(M\)-th average is made weighting equally the newly predicted solution \((M)\tilde{u}_{j}^{n+1}\) and the solution at the “old” timelevel” \(u_{n}\). This, however, can be seen as the special case of a more generic averaging of the type

\[
(M)\bar{u}_{j}^{n+1/2} = \theta (M)\tilde{u}_{j}^{n+1} + (1 - \theta)u_{j}^{n}, \tag{3.62}
\]
where $0 < \theta < 1$ is a constant coefficient. Predictor-corrector schemes using this type of averaging are part of a large class of algorithms named $\theta$-methods [10], and we refer to the ICN generalized in this way as to the “$\theta$-ICN” method.

A different and novel generalisation of the $\theta$-ICN can be obtained by swapping the averages between two subsequent corrector steps, so that in the $M$-th corrector step
\[
(M)\bar{u}_{n+1/2} = (1 - \theta) (M)\tilde{u}_{n+1} + \theta u^n ,
\]
while in the $(M+1)$-th corrector step
\[
(M+1)\bar{u}_{n+1/2} = \theta (M+1)\tilde{u}_{n+1} + (1 - \theta) u^n .
\]

Note that as long as the number of iterations is even, the sequence in which the averages are computed is irrelevant. Indeed, the weights $\theta$ and $1 - \theta$ in eqs. (3.63)–(3.64) could be inverted and all of the relations discussed hereafter for the swapped weighted averages would continue to hold after the transformation $\theta \rightarrow 1 - \theta$.

### Constant Arithmetic Averages

Using a von Neumann stability analysis, Teukolsky has shown that for a hyperbolic equation the ICN scheme with $M$ iterations has an amplification factor [13]
\[
(M)\xi = 1 + 2 \sum_{n=1}^{M} (-i\beta)^n ,
\]
where $\beta := v[\Delta t/(2\Delta x)] \sin(k\Delta x)$ \(^1\). More specifically, zero and one iterations yield an unconditionally unstable scheme, while two and three iterations a stable one provided that $\beta^2 \leq 1$; four and five iterations lead again to an unstable scheme and so on. Furthermore, because the scheme is second-order accurate from the first iteration on, Teukolsky’s suggestion when using the ICN method for hyperbolic equations was that two iterations should be used and no more [13]. This is the number of iterations we will consider hereafter.

### Constant Weighted Averages

Performing the same stability analysis for a $\theta$-ICN is only slightly more complicated and truncating at two iterations the amplification factor is found to be
\[
\xi = 1 - 2i\beta - 4\beta^2 \theta + 8i\beta^3 \theta^2 ,
\]
where $\xi$ is a shorthand for $(2)\xi$. The stability condition in this case translates into requiring that
\[
16\beta^4 \theta^4 - 4\beta^2 \theta^2 - 2\theta + 1 \leq 0 ,
\]
or, equivalently, that for $\theta > 3/8$
\[
\frac{\sqrt{\frac{1}{2} - \sqrt{2\theta - \frac{3}{4}}} - \sqrt{2\theta - \frac{3}{4}}} {2\theta} \leq \beta \leq \frac{\sqrt{\frac{1}{2} + \sqrt{2\theta - \frac{3}{4}}} - \sqrt{2\theta - \frac{3}{4}}} {2\theta} ,
\]
\(^1\)Note that we define $\beta$ to have the opposite sign of the corresponding quantity defined in ref. [13]
3.6. **THE 1D ICN SCHEME: \( O(\Delta T^2, \Delta X^2) \)**

3.6. THE 1D ICN SCHEME: \( O(\Delta T^2, \Delta X^2) \)

![Figure 3.13](image)

Figure 3.13: *Left panel:* stability region in the \((\theta, \beta)\) plane for the two-iterations \(\theta\)-ICN for the advection equation (3.2). Thick solid lines mark the limit at which \(|\xi| = 1\), while the dotted contours indicate the values of the amplification factor in the stable region. The shaded area for \(\theta < 1/2\) refers to solutions that are linearly unstable [15].

*Right panel:* same as in the left panel but when the averages between two corrections are swapped. Note that the amplification factor in this case is less sensitive on \(\theta\) and always larger than the corresponding amplification factor in the left panel.

which reduces to \(\beta^2 \leq 1\) when \(\theta = 1/2\). Because the condition (3.68) must hold for every wavenumber \(k\), we consider hereafter \(\beta := v\Delta t/(2\Delta x)\) and show in the left panel of Fig. 3.13 the region of stability in the \((\theta, \beta)\) plane. The thick solid lines mark the limit at which \(|\xi| = 1\), while the dotted contours indicate the different values of the amplification factor in the stable region.

A number of comments are worth making. Firstly, although the condition (3.68) allows for weighting coefficients \(\theta < 1/2\), the \(\theta\)-ICN is stable only if \(\theta \geq 1/2\). This is a known property of the weighted Crank-Nicolson scheme [10] and inherited by the \(\theta\)-ICN. In essence, when \(\theta \neq 1/2\) spurious solutions appear in the method [16] and these solutions are linearly unstable if \(\theta < 1/2\), while they are stable for \(\theta > 1/2\) [15]. For this reason we have shaded the area with \(\theta < 1/2\) in the left panel of Fig. 3.13 to exclude it from the stability region. Secondly, the use of a weighting coefficient \(\theta > 1/2\) will still lead to a stable scheme provided that the timestep (i.e., \(\beta\)) is suitably decreased. Finally, as the contour lines in the left panel of Fig. 3.13 clearly show, the amplification factor can be very sensitive on \(\theta\).

**Swapped weighted averages**

The calculation of the stability of the \(\theta\)-ICN when the weighted averages are swapped as in eqs. (3.63) and (3.64) is somewhat more involved; after some lengthy but straight-
forward algebra we find the amplification factor to be

\[ \xi = 1 - 2i\beta - 4\beta^2 \theta + 8i\beta^3 \theta (1 - \theta) , \quad (3.69) \]

which differs from (3.66) only in that the \( \theta^2 \) coefficient of the \( O(\beta^3) \) term is replaced by \( \theta(1 - \theta) \). The stability requirement \( |\xi| \leq 1 \) is now expressed as

\[ 16\beta^4 \theta^2 (1 - \theta)^2 - 4\beta^2 \theta (2 - 3\theta) - 2\theta + 1 \leq 0 . \quad (3.70) \]

Solving the condition (3.70) with respect to \( \beta \) amounts then to requiring that

\[ \beta \geq \frac{\sqrt{2 - 3\theta - \sqrt{4\theta - 11\theta^2 + 8\theta^3}}}{2(1 - \theta)\sqrt{2\theta}} , \quad (3.71a) \]

\[ \beta \leq \frac{\sqrt{2 - 3\theta + \sqrt{4\theta - 11\theta^2 + 8\theta^3}}}{2(1 - \theta)\sqrt{2\theta}} , \quad (3.71b) \]

which is again equivalent to \( \beta^2 \leq 1 \) when \( \theta = 1/2 \). The corresponding region of stability is shown in right panel of Fig. 3.13 and should be compared with left panel of the same Figure. Note that the average-swapping has now considerably increased the amplification factor, which is always larger than the corresponding one for the \( \theta \)-ICN in the relevant region of stability (i.e., for \( 1/2 \leq \theta \leq 1/2 \)).

\footnote{Of course, when the order of the swapped averages is inverted from the one shown in eqs. (3.63)–(3.64) the stability region will change into \( 0 \leq \theta \leq 1/2 \).}
3.6. **THE 1D ICN SCHEME: \( \mathcal{O}(\Delta T^2, \Delta X^2) \)**

### 3.6.2 Summary

In what follows I summarise the most salient aspects of the different finite-difference operators discussed so far and report, for each of them, the truncation error \( \epsilon_T \), the amplification factor \( |\xi|^2 \) and the finite-difference representation of the advection equation 3.2. I recall that \( \alpha := \frac{v\Delta t}{\Delta x} \).

| Method          | \( \epsilon_T \) | \( |\xi|^2 \) for \( (k\Delta x \ll 1) \) | finite-difference form |
|-----------------|-------------------|--------------------------------------------|-----------------------|
| Upwind          | \( \mathcal{O}(\Delta t, \Delta x) \) | \( 1 - 2|\alpha|(1 - |\alpha|) \cos(k\Delta x) \) | \( u_{j}^{n+1} = u_{j}^{n} \pm \alpha(u_{j+1}^{n} - u_{j}^{n}) \) |
| FTCS            | \( \mathcal{O}(\Delta t, \Delta x^2) \) | \( 1 + \sin^2(k\Delta x)\alpha^2 \) | \( u_{j}^{n+1} = u_{j}^{n} - \alpha(u_{j+1}^{n} - u_{j-1}^{n}) \) |
| Lax Friedrichs  | \( \mathcal{O}(\Delta t, \Delta x^2) \) | \( 1 - \sin^2(k\Delta x)(1 - \alpha^2) \) | \( u_{j}^{n+1} = \frac{1}{2}(u_{j+1}^{n} + u_{j-1}^{n}) - \frac{1}{2}\alpha(u_{j+1}^{n} - u_{j-1}^{n}) \) |
| Leapfrog        | \( \mathcal{O}(\Delta t^2, \Delta x^2) \) | \( 1 \) | \( u_{j}^{n+1} = u_{j}^{n-1} - \alpha(u_{j+1}^{n} - u_{j-1}^{n}) \) |
| Lax Wendroff    | \( \mathcal{O}(\Delta t^2, \Delta x^2) \) | \( 1 - \alpha^2(1 - \alpha^2) \sin^2(k\Delta x) \) | \( u_{j}^{n+1} = u_{j}^{n} - \frac{1}{2}\alpha(u_{j+1}^{n} - u_{j-1}^{n}) + \frac{1}{2}\alpha^2(2u_{j+1}^{n} - 2u_{j}^{n} + u_{j-1}^{n}) \) |

Table 3.1: Schematic summary of the finite-difference operators discussed so far.
3.6.3 Finite-difference stencils

In what follow I summarise the most used finite-difference stencils for derivatives of order 1 to 4.

**Finite-difference stencils for** $\partial u / \partial x$

<table>
<thead>
<tr>
<th>Type</th>
<th>Difference Stencil</th>
<th>LTE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Forward</strong></td>
<td>$(-u_j + u_{j+1}) / h$</td>
<td>$O(h)$</td>
</tr>
<tr>
<td><strong>Backward</strong></td>
<td>$(-u_{j-1} + u_j) / h$</td>
<td>$O(h)$</td>
</tr>
<tr>
<td><strong>Forward</strong></td>
<td>$(-3u_j + 4u_{j+1} - u_{j+2}) / h$</td>
<td>$O(h^2)$</td>
</tr>
<tr>
<td><strong>Backward</strong></td>
<td>$(u_{j-2} - 4u_{j-1} + 3u_j) / 2h$</td>
<td>$O(h^2)$</td>
</tr>
<tr>
<td><strong>Centered</strong></td>
<td>$(-u_{j-1} + u_{j+1}) / 2h$</td>
<td>$O(h^2)$</td>
</tr>
<tr>
<td><strong>Forward</strong></td>
<td>$(-25u_j + 48u_{j+1} - 36u_{j+2} + 16u_{j+3} - 3u_{j+4}) / 12h$</td>
<td>$O(h^4)$</td>
</tr>
<tr>
<td><strong>Backward</strong></td>
<td>$(3u_{j-4} - 16u_{j-3} + 36u_{j-2} - 48u_{j-1} + 25u_j) / 12h$</td>
<td>$O(h^4)$</td>
</tr>
<tr>
<td><strong>Centered</strong></td>
<td>$(u_{j-2} - 8u_{j-1} + 8u_{j+1} - u_{j+2}) / 12h$</td>
<td>$O(h^4)$</td>
</tr>
</tbody>
</table>

Table 3.2: Finite-difference stencils for $\partial u / \partial x$
3.6. THE 1D ICN SCHEME: $O(\Delta T^2, \Delta X^2)$

Finite-difference stencils for $\partial^2 u/\partial^2 x$

<table>
<thead>
<tr>
<th>type</th>
<th>Difference Stencil</th>
<th>LTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>forward</td>
<td>$\frac{(u_j - 2u_{j+1} + u_{j+2})}{h^2}$</td>
<td>$O(h)$</td>
</tr>
<tr>
<td>backward</td>
<td>$\frac{(u_{j-2} - 2u_{j-1} + u_j)}{h^2}$</td>
<td>$O(h)$</td>
</tr>
<tr>
<td>forward</td>
<td>$\frac{(2u_j - 5u_{j+1} + 4u_{j+2} - u_{j+3})}{h^2}$</td>
<td>$O(h^2)$</td>
</tr>
<tr>
<td>backward</td>
<td>$\frac{(-u_{j-3} + 4u_{j-2} - 5u_{j-1} + 2u_j)}{h^2}$</td>
<td>$O(h^2)$</td>
</tr>
<tr>
<td>centered</td>
<td>$\frac{(u_{j-1} - 2u_j + u_{j+1})}{h^2}$</td>
<td>$O(h^2)$</td>
</tr>
<tr>
<td>forward</td>
<td>$\frac{45u_j - 154u_{j+1} + 214u_{j+2} - 156u_{j+3} + 61u_{j+4} - 10u_{j+5}}{12h^2}$</td>
<td>$O(h^4)$</td>
</tr>
<tr>
<td>backward</td>
<td>$\frac{-10u_{j-5} + 61u_{j-4} - 156u_{j-3} + 214u_{j-2} - 154u_{j-1} + 45u_j}{12h^2}$</td>
<td>$O(h^4)$</td>
</tr>
<tr>
<td>centered</td>
<td>$\frac{(-u_{j-2} + 16u_{j-1} - 30u_j + 16u_{j+1} - u_{j+2})}{12h^2}$</td>
<td>$O(h^4)$</td>
</tr>
</tbody>
</table>

Table 3.3: Finite-difference stencils for $\partial^2 u/\partial^2 x$
Finite difference stencils for $\partial^3 u / \partial^3 x$

<table>
<thead>
<tr>
<th>type</th>
<th>Difference Stencil</th>
<th>LTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>forward</td>
<td>$(-u_j + 3u_{j+1} - 3u_{j+2} + u_{j+3}) / h^3$</td>
<td>$O(h)$</td>
</tr>
<tr>
<td>backward</td>
<td>$(-u_{j-3} + 3u_{j-2} - 3u_{j-1} + u_j) / h^3$</td>
<td>$O(h)$</td>
</tr>
<tr>
<td>forward</td>
<td>$(-5u_j + 18u_{j+1} - 24u_{j+2} + 14u_{j+3} - 3u_{j+4}) / 2h^3$</td>
<td>$O(h^2)$</td>
</tr>
<tr>
<td>backward</td>
<td>$(3u_{j-4} - 14u_{j-3} + 24u_{j-2} - 18u_{j-1} + 5u_j) / 2h^3$</td>
<td>$O(h^2)$</td>
</tr>
<tr>
<td>centered</td>
<td>$(-u_{j-2} + 2u_{j-1} - 2u_{j+1} + u_{j+2}) / 2h^3$</td>
<td>$O(h^2)$</td>
</tr>
</tbody>
</table>

Table 3.4: Finite difference stencils for $\partial^3 u / \partial^3 x$
3.6. THE 1D ICN SCHEME: $O(\Delta T^2, \Delta X^2)$

Finite difference stencils for $\partial^4 u / \partial^4 x$

<table>
<thead>
<tr>
<th>type</th>
<th>Difference Stencil</th>
<th>LTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>forward</td>
<td>$(u_j - 4u_{j+1} + 6u_{j+2} - 4u_{j+3} + u_{j+4}) / h^4$</td>
<td>$O(h)$</td>
</tr>
<tr>
<td>backward</td>
<td>$(u_{j-4} - 4u_{j-3} + 6u_{j-2} - 4u_{j-1} + u_j) / h^4$</td>
<td>$O(h)$</td>
</tr>
<tr>
<td>forward</td>
<td>$(3u_j - 14u_{j+1} + 26u_{j+2} - 24u_{j+3} + 11u_{j+4} - 2u_{j+5}) / h^4$</td>
<td>$O(h^2)$</td>
</tr>
<tr>
<td>backward</td>
<td>$(-2u_{j-5} + 11u_{j-4} - 24u_{j-3} + 26u_{j-2} - 14u_{j-1} + 3u_j) / h^4$</td>
<td>$O(h^2)$</td>
</tr>
<tr>
<td>centered</td>
<td>$(u_{j-2} - 4u_{j-1} + 6u_j - 4u_{j+1} + u_{j+2}) / h^4$</td>
<td>$O(h^2)$</td>
</tr>
</tbody>
</table>

Table 3.5: Finite difference stencils for $\partial^4 u / \partial^4 x$
Chapter 4

Dissipation, Dispersion and Convergence

We will here discuss a number of problems that often emerge when using finite-difference techniques for the solution of hyperbolic partial differential equations. In stable numerical schemes the impact of many of these problems can be suitably reduced by going to sufficiently high resolutions, but it is nevertheless important to have a simple and yet clear idea of what are the most common sources of these problems.

4.1 On the Origin of Dissipation and Dispersion

We have already seen in Chapter 3 how the Lax-Friedrichs scheme applied to a linear advection equation (3.2) yields the finite-difference expression

\[ u_j^{n+1} = \frac{1}{2}(u_{j+1}^n + u_j^n) - \frac{\alpha}{2}(u_{j+1}^n - u_j^n) + O(\Delta x^2). \]  

(4.1)

We have also mentioned how expression (4.1) can be rewritten as

\[ u_j^{n+1} = u_j^n - \frac{\alpha}{2}(u_{j+1}^n - u_j^n) + \frac{1}{2}(u_{j+1}^n - 2u_j^n + u_j^{n-1}) + O(\Delta x^2), \]  

(4.2)

to underline how the Lax-Friedrichs scheme effectively provides a first-order finite-difference representation of a non-conservative equation

\[ \frac{\partial u}{\partial t} + v \frac{\partial u}{\partial x} = \varepsilon_{LF} \frac{\partial^2 u}{\partial x^2}, \]  

(4.3)

that is an advection-diffusion equation in which a dissipative term

\[ \varepsilon_{LF} := \frac{v \Delta x^2}{2\Delta t}, \]  

(4.4)

is present. Given a computational domain of length \( L \), this scheme will therefore have a typical diffusion timescale \( \tau \simeq L^2/\varepsilon_{LF} \). Clearly, the larger the diffusion coefficient, the faster will the solution be completely smeared over the computational domain.
In a similar way, it is not difficult to realise that the upwind scheme
\[ u_{j}^{n+1} = u_{j}^{n} - \alpha \left( u_{j}^{n} - u_{j-1}^{n} \right) + \mathcal{O}(\Delta x^2) , \]  
(4.5)
provides a first-order accurate (in space) approximation to equation (3.2), but a second-order approximation to equation
\[ \frac{\partial u}{\partial t} + v \frac{\partial u}{\partial x} = \varepsilon_{uw} \frac{\partial^2 u}{\partial x^2} , \]  
(4.6)
where
\[ \varepsilon_{uw} := \frac{v \Delta x}{2} . \]  
(4.7)
Stated differently, also the upwind method reproduces at higher-order an advection-diffusion equation with a dissipative term which is responsible for the gradual dissipation of the advected quantity \( u \). This is shown in Fig. 4.2 for a wave packet (i.e., a periodic function embedded in a Gaussian) propagating to the right and where it is important to notice how the different peaks in the packet are advected at the correct speed, although their amplitude is considerably diminished.

In Courant-limited implementations, \( \alpha = |v| \Delta t / \Delta x < 1 \) so that the ratio of the dissipation coefficients can be written as
\[ \frac{\varepsilon_{LF}}{\varepsilon_{uw}} = \frac{1}{\alpha} \geq 1 , \quad \text{for} \quad \alpha \in [0, 1] . \]  
(4.8)
In other words, while the upwind and the Lax-Friedrichs methods are both dissipative, the latter is generically more dissipative despite being more accurate in space. This can be easily appreciated by comparing Figs. 3.3 and 3.7 but also provides an important rule: a more accurate numerical scheme is not necessarily a preferable one.

A bit of patience and a few lines of algebra would also show that the Lax-Wendroff scheme for the advection equation (3.2) [cf. eq. (3.46)]
\[ u_{j}^{n+1} = u_{j}^{n} - \alpha \left( u_{j+1}^{n} - u_{j-1}^{n} \right) + \frac{\alpha^2}{2} \left( u_{j+1}^{n} - 2u_{j}^{n} + u_{j-1}^{n} \right) + \mathcal{O}(\Delta x^2) . \]  
(4.9)
provides a first-order accurate approximation to equation (3.2), a second-order approximation to an advection-diffusion equation with dissipation coefficient \( \varepsilon_{LW} \), and a third-order approximation to equation
\[ \frac{\partial u}{\partial t} + v \frac{\partial u}{\partial x} = \varepsilon_{LW} \frac{\partial^2 u}{\partial x^2} + \beta_{LW} \frac{\partial^3 u}{\partial x^3} , \]  
(4.10)
where
\[ \varepsilon_{LW} := \frac{\alpha v \Delta x}{2} , \quad \beta_{LW} := -\frac{v \Delta x^2}{6} \left( 1 - \alpha^2 \right) . \]  
(4.11)
As mentioned in Section 3, the Lax-Wendroff scheme retains some of the dissipative nature of the originating Lax-Friedrichs scheme and this is incorporated in the dissipative term proportional to \( \varepsilon_{LW} \). Using expression (4.9), it is easy to deduce the
4.1. ON THE ORIGIN OF DISSIPATION AND DISPERSION

Figure 4.1: Time evolution of a wave-packet initially centred at $x = 0.5$ computed using a Lax-Friedrichs scheme with $C_{CFL} = 0.75$. The analytic solution at time $t = 2$ is shown with a solid line the dashed lines are used to represent the numerical solution at the same time. Note how dissipation reduces the amplitude of the wave-packet but does not change sensibly the propagation of the wave-packet.

magnitude of this dissipation and compare it with the equivalent one produced with the Lax-Friedrichs scheme. A couple of lines of algebra show that

$$
\varepsilon_{\text{LW}} = \alpha^2 \varepsilon_{\text{LF}} \ll \varepsilon_{\text{LF}},
$$

(4.12)

thus emphasizing that the Lax-Wendroff scheme is considerably less dissipative than the corresponding Lax-Friedrichs.

The simplest way of quantifying the effects introduced by the right-hand-sides of equations (4.3), (4.6), and (4.10) is by using a single Fourier mode with angular frequency $\omega$ and wavenumber $k$, propagating in the positive $x$-direction, i.e.,

$$
u(x, t) = e^{i(kx - \omega t)}.
$$

(4.13)

It is then easy to verify that in the continuum limit

$$
\frac{\partial u}{\partial t} = -i\omega u, \quad \frac{\partial u}{\partial x} = iku, \quad \frac{\partial^2 u}{\partial x^2} = -k^2 u, \quad \frac{\partial^3 u}{\partial x^3} = -ik^3 u.
$$

(4.14)
Figure 4.2: Time evolution of a wave-packet initially centred at $x = 0.5$ computed using a Lax-Wendroff scheme with $C_{CFL} = 0.75$. The analytic solution at time $t = 2$ is shown with a solid line the dashed lines are used to represent the numerical solution at the same time. Note how the amplitude of the wave-packet is not drastically reduced but the group velocity suffers from a considerable error.

In the case in which the finite difference scheme provides an accurate approximation to a purely advection equation, the relations (4.14) lead to the obvious dispersion relation $\omega = vk$, so that the numerical mode $\tilde{u}(x, t)$ will have a solution

$$\tilde{u}(x, t) = e^{ik(x-vt)}$$

representing a mode propagating with phase velocity $c_p := \omega/k = v$, which coincides with the group velocity $c_g := \partial \omega/\partial k = v$.

However, it is simple to verify that the advection-diffusion equation approximated by the Lax-Friedrichs scheme (4.3), will have a corresponding solution

$$\tilde{u}(x, t) = e^{-\varepsilon_{LF}k^2t}e^{ik(x-vt)}$$

thus having, besides the advective term, also an exponentially decaying mode. Similarly, a few lines of algebra are sufficient to realise that the dissipative term does not couple with the advective one and, as a result, the phase and group velocities remain
4.2. MEASURING DISSIPATION AND CONVERGENCE

the same and \( c_p = c_g = v \). This is clearly shown in Fig. 4.1 which shows how the wave packet is sensibly dissipated but, overall, maintains the correct group velocity.

Finally, it is possible to verify that the advection-diffusion equation approximated by the Lax-Wendroff scheme (4.10), will have a solution given by

\[
\tilde{u}(x, t) = e^{-\varepsilon_{LW} k^2 t} e^{ik[x-(v+\beta_{LW} k^2)t]},
\]

where, together with the advective and (smaller) exponentially decaying modes already encountered before, there appears also a dispersive term \( \sim \beta_{LW} k^2 t \) producing different propagation speeds for modes with different wavenumbers. This becomes apparent after calculating the phase and group velocities which are given by

\[
c_p = \frac{\omega}{k} = v + \beta_{LW} k^2, \quad \text{and} \quad c_g = \frac{\partial \omega}{\partial k} = v + 3\beta_{LW} k^2,
\]

and provides a simple interpretation of the results shown in Fig. 4.2.

4.2 Measuring Dissipation and Convergence

From what discussed so far it appears clear that one is often in need of tools that allow a rapid comparison among different evolution schemes. One might be interested, for instance, in estimating which of two methods is less dissipative or whether an evolution scheme which is apparently stable will eventually turn out to be unstable. In what follows we discuss some of these tools and how they can be used to ascertain a fundamental property of the numerical solution: its convergence

4.2.1 The summarising power of norms

A very useful tool that can be used in this context is the calculation of the “norms” of the quantity we are interested in. In the continuum limit the \( p\)-norm is defined as

\[
\|u\|_p = \left( \frac{1}{b-a} \int_a^b |u(x, t)|^p dx \right)^{1/p}.
\]

and has the same dimensions of the originating quantity \( u(x, t) \). The extension of this concept to a discretised space and time is straightforward and yields the commonly used norms

\[
1\text{-norm} :: \quad ||u||(t^n) = \frac{1}{N} \sum_{j=1}^{N} |u^n_j|,
\]

\[
2\text{-norm} :: \quad ||u||^2(t^n) = \frac{1}{N} \left( \sum_{j=1}^{N} (u^n_j)^2 \right)^{1/2},
\]

\[
p\text{-norm} :: \quad ||u||^p(t^n) = \frac{1}{N} \left( \sum_{j=1}^{N} (u^n_j)^p \right)^{1/p},
\]

\[
\infty\text{-norm} :: \quad ||u||_{\infty}(t^n) = \max_{j=1,\ldots,N} (|u^n_j|).
\]
In the case of a scalar wave equation (see Sect. 5 for a discussion), the 2-norm has a physical interpretation and could be associated to the amount of energy contained in the numerical domain; its conservation is therefore a clear signature of a non-dissipative numerical scheme.

Figure 4.3: Time evolution of the logarithm of the 2-norms for the different numerical schemes discussed so far. Sommerfeld outgoing boundary conditions were used in this example.

Fig. 4.2 compares the 2-norms for the different numerical schemes discussed so far and in the case in which Sommerfeld outgoing boundary conditions were used. Note how the FTCS scheme is unstable and that the errors are already comparable with the solution well before a crossing time. Similarly, it is evident that the use of Sommerfeld boundary conditions allows a smooth evacuation of the energy in the wave from the numerical grid after $t \sim 6$.

4.2.2 Consistency and Convergence

Consider therefore a PDE of the type

$$\mathcal{L}(u) - f = 0,$$  \hspace{1cm} (4.24)
where \( \mathcal{L} \) is a second-order differential quasi-linear operator [cf. eq. (1.1)]. Let also \( \mathcal{L}_\Delta \) be the discretized representation of such continuum differential operator and \( \epsilon = O(\Delta x^p, \Delta t^q) \) the associated truncation error, i.e.,

\[
\mathcal{L}_\Delta (u^n_i) - f^n_i = 0 + O(\Delta x^p, \Delta t^q) .
\]

(4.25)

For compactness let us assume that largest contribution to the truncation error can be expressed simply as \( \epsilon = Ch^p = O(h^p) \) where \( h \) corresponds to either the spatial or time discretization and \( C \) is a real constant coefficient. The finite-difference representation \( \mathcal{L}_\Delta \) is said to be consistent if

\[
\lim_{h \to 0} \epsilon = 0 ,
\]

(4.26)

Let \( u(x, t) \) represent the exact solution to a PDE and \( \tilde{u} \) the exact solution of the finite-difference equation that approximates the PDE with a truncation error \( O(\Delta x^p, \Delta t^q) \). The finite-difference equation is said to be convergent when the truncation error tends to zero as a power of \( p \) in \( \Delta x \) and a power of \( q \) in \( \Delta t \), namely

\[
\lim_{h \to 0} \epsilon = \Delta x^p + \Delta t^q ,
\]

(4.27)

Note that this condition is much more severe that the simple requirement that the truncation error will tend to zero as \( \Delta x \) and \( \Delta t \) tend to zero. The latter condition, in fact, does not ensure that the numerical solution is approaching the exact one at the expected rate, that is the rate determined by the truncation error and consequent to the choice of the given finite-difference representation of the continuum differential operator.

Since checking convergence essentially amounts to measuring how the truncation error changes with resolution, it is useful to define a local (i.e., pointwise) deviation from the exact solution \( u \) at \( x = x_j \) as

\[
\epsilon_j(h) = u_j^{(h)} - u(x_j)
\]

(4.28)

be the magnitude of the largest truncation error (and which could be either in space or in time) associated to the numerical solution \( u_j^{(h)} \) obtained with grid spacing \( h \). If the numerical method used is \( p \)-th order accurate, then

\[
\epsilon_j(h) = Ch^p + O(h^{p+1}) ,
\]

(4.29)

where \( C \) is a constant real coefficient. A different solution computed with a grid spacing \( k \) will have at the same spatial position \( x_j \) a corresponding truncation \( \epsilon_j(k) \) error, so that error ratio will be

\[
R_j(h, k) := \frac{\epsilon_j(h)}{\epsilon_j(k)} ,
\]

(4.30)

and the “numerical” local convergence order, that is the order of convergence as measured from the two numerical solutions at \( x_j \) will be

\[
\hat{p} := \frac{\log R_j(h, k)}{\log(h/k)} .
\]

(4.31)
In the rather common case in which \( k = h/2 \), expressions (4.30) reduce to
\[
R_j(h, h/2) = 2^\tilde{p} ,
\]
and the overall order of accuracy is measured numerically as \( \tilde{p} = \log_2(R) \). As we will discuss in the following Section, the discrete representation of the continuum equations is said to be convergent if and only if \( \tilde{p} = p \), i.e., if
\[
\lim_{h \to 0} \tilde{p} := \frac{\log(\epsilon)}{\log(CH)} = p .
\]
(4.32)

Stated differently, convergence requires not only that the error is decreasing and thus that the method is consistent (see Sect. 4.2.3) but that it is decreasing at the expected rate.

In general there will be a minimum resolution, say \( h_{\text{min}} \), below which the truncation error will dominate over the others, e.g., round-off error. Clearly, one should expect convergence only for \( h < h_{\text{min}} \) and the solution in this case is said to be in a convergent regime.

What discussed so far assumes the knowledge of the exact solution, which, in general, is not available. This does not represent a major obstacle and the convergence test can still be performed by simply employing a third numerical evaluation of the solution. This is referred to as a “self-convergence” test and exploits the fact that the difference between two numerical solutions does not depend on the actual exact solution
\[
\epsilon_j(h) - \epsilon_j(k) = \epsilon_j(h) - u(x_j) - (\epsilon_j(k) - u(x_j)) = \epsilon_j(h) - \epsilon_j(k) ,
\]
where of course the two solutions \( u_j^{(h)} \) and \( u_j^{(k)} \) should be evaluated at the same grid-point \( x_j \). If one of the numerical solutions is not available at such a point (e.g., because the spacing used is not uniform) a suitable interpolation is needed and attention must be paid that the error it introduces is much smaller than either \( \epsilon_j(h) \) or \( \epsilon_j(k) \) in order not to spoil the convergence test.

With (4.29) in mind and using three different numerical solutions \( u_j^{(h)} \), \( u_j^{(k)} \), \( u_j^{(l)} \) with grid spacings such that \( h > k > l \), the numerical error ratio is then defined as
\[
R_j(h, k; l) := \frac{u_j^{(h)} - u_j^{(l)}}{u_j^{(k)} - u_j^{(l)}} = \frac{\epsilon_j(h) - \epsilon_j(l)}{\epsilon_j(k) - \epsilon_j(l)} = \frac{h^\tilde{p} - l^\tilde{p}}{k^\tilde{p} - l^\tilde{p}} ,
\]
(4.33)
where the numerical solution \( u_j^{(l)} \) with the associated error \( \epsilon_j(l) \) has the role of “reference” solution since it is the one with the smallest error. In the common case in which \( k = h/2 \) and \( l = k/2 = h/4 \), the error ratio assumes the simple expression
\[
R(h, h/2; h/4) = 2^\tilde{p} - 1 ,
\]
so that the computed overall accuracy order is \( \tilde{p} = \log_2(R + 1) \).

As a final comment we note that all what discussed so far for a local convergence analysis can be extended to a global evaluation of the truncation error and this amounts to essentially replacing all the error estimates discussed above with the corresponding \( p \)-norms.
4.2.3 Convergence and Stability

We conclude this Chapter with an important theorem that brings together many of the different concepts exposed so far and provides a unique interpretation for the interplay between consistency, convergence and stability. We have seen in the previous Section that a finite-difference representation is said to be consistent if

\[ \lim_{h \to 0} \epsilon = 0, \]  

and it will be said to be convergent if

\[ \lim_{h \to 0} \tilde{p} := \frac{\log(\epsilon)}{\log(CH)} = p. \]  

(4.35)

Clearly, also for a convergent solution \( \epsilon \to 0 \) as \( h \to 0 \); however, conditions (4.26) and (4.32) underline that while a convergent solution is also consistent, the latter is not necessarily true. Stated differently, while there are infinite consistent representation of the differential operator, only one will be convergent.

There are numerous ways in which a consistent representation of a differential operator may not be convergent and in large majority of the cases the lack of convergence is related to a programming error (or “bug”). Because of this, convergence tests represent the most efficient if not the only way of validating that the discrete form of the equations represents a faithful representation of the continuum ones (and hence of picking out bugs!).

The knowledge of convergence has also another rewarding aspect and this is beautifully summarised in the following theorem:

**Theorem** Given a properly posed initial-value problem and a finite difference approximation to it that satisfies the consistency condition, stability is the necessary and sufficient condition for convergence.

This theorem, known as the “Lax equivalence theorem”, is very powerful as it shows that for an initial-value problem which has been discretised with a consistent finite-difference operator, the concept of stability and convergence are interchangeable. In general, therefore, proving that the numerical solution is convergent will not only validate that the discrete form of the equations represents a faithful representation of the continuum ones, but also that the solution will be bounded at all times.
Chapter 5

The Wave Equation in 1D

The numerical solution of the wave equation offers a good example of how a higher-order (in space and time) PDE can be easily solved numerically through the solution of a system of coupled 1st-order PDEs.

In one spatial dimension (1D) the wave equation has the general form:

\[
\frac{\partial^2 u}{\partial t^2} = v^2 \frac{\partial^2 u}{\partial x^2},
\]  

(5.1)

where, for simplicity, we will assume that \( v \) is constant (i.e., \( v \neq v(x) \)), thus restricting our attention to linear problems. It is much more convenient to rewrite (5.1) as a system of coupled first-order conservative PDE. For this we set

\[
r = v \frac{\partial u}{\partial x},
\]

(5.2)

\[
s = \frac{\partial u}{\partial t},
\]

(5.3)

so that (5.1) can be rewritten as a system of three coupled, first-order differential equations

\[
\begin{align*}
\frac{\partial r}{\partial t} &= v \frac{\partial u}{\partial x}, \\
\frac{\partial s}{\partial t} &= v \frac{\partial r}{\partial x}, \\
\frac{\partial u}{\partial t} &= s,
\end{align*}
\]

where it should be noted that the equations have the time derivative of one variable that is proportional to the space derivative of the other variable. This breaks the advective nature of the equation discussed in the previous Chapter and will prevent, for instance, the use of an upwind scheme.

In vector notation the system (5.4) can be symbolically written as

\[
\frac{\partial U}{\partial t} + \frac{\partial F(U)}{\partial x} = 0,
\]

(5.4)
Figure 5.1: Plot of the time evolution of the wave equation when the FTCS scheme is used. The initial conditions were given by a Gaussian centered at $x = 5$ with unit variance and are shown with the dotted line. Note the growth of the wave crests and the appearance of short wavelength noise. When this happens, the numerical errors have grown to be comparable with the solution which will be rapidly destroyed.

where

$$ U = \begin{pmatrix} r \\ s \end{pmatrix}, \quad \text{and} \quad F(U) = \begin{pmatrix} 0 & -v \\ -v & 0 \end{pmatrix} U. \quad (5.5) $$

5.1 The FTCS Scheme

As mentioned in the previous Chapter, the upwind method cannot be applied to the solution of the wave equation and the simplest, first-order in time method we can use for the solution of the wave equation is therefore given by the FTCS scheme. Applying it to the first-order system (5.4) and obtain

$$ r_j^{n+1} = r_j^n + \frac{\alpha}{2} (s_j^{n+1} - s_j^n) + O(\Delta x^2), \quad (5.6) $$

$$ s_j^{n+1} = s_j^n + \frac{\alpha}{2} (r_j^{n+1} - r_j^n) + O(\Delta x^2), \quad (5.7) $$
Once the value of \( s_{n+1}^j \) has been calculated, the value of \( u \) can be integrated in time according to equation (5.3) so that
\[
u_{n+1}^j = u_{n}^j + \Delta t s_{n+1}^j + O(\Delta x^2),
\] (5.8)
where it should be noted that \( u_{n+1}^j \) has the same truncation error of \( r_{n+1}^j \) and \( s_{n+1}^j \).

Of course, we do not expect that the FTCS scheme applied to the solution of the wave equation will provide a stable evolution and this is clearly shown in Fig. 5.1 which reports the solution of equations (5.6), (5.6) and (5.8) having as initial conditions a Gaussian centered at \( x = 5 \) with unit variance. Different lines show the solution at different times and is apparent how the initial profile splits into two part propagating in two opposite directions. During the evolution, however, the error grows (note that the peaks of the two packets increase with time) and in about one crossing time the short wavelength noise appears (this is shown by the small sharp peaks produced when the wave has left the numerical grid). When this happens, the numerical errors have grown to be comparable with the solution, which will be rapidly destroyed.

5.2 The Lax-Friedrichs Scheme

As done in the previous Section, we can apply the Lax-Friedrichs scheme to the solution of the wave equation through the first-order system (5.4) and easily obtain
\[
r_{j+\frac{1}{2}}^{n+1} = \frac{1}{2} (r_{j+1}^n + r_{j-1}^n) + \frac{\alpha}{2} (s_{j+1}^n - s_{j-1}^n) + O(\Delta x^2),
\] (5.9)
\[
s_{j+\frac{1}{2}}^{n+1} = \frac{1}{2} (s_{j+1}^n + s_{j-1}^n) + \frac{\alpha}{2} (r_{j+1}^n - r_{j-1}^n) + O(\Delta x^2),
\] (5.10)

Also in this case, once the value of \( s_{n+1}^j \) has been calculated, the value for \( u_{n+1}^j \) can be computed according to (5.8).

The solution of equations (5.9), (5.9) and (5.8) with the same initial data used in Fig. 5.1 is shown in Fig. 5.2. Note that we encounter here the same behaviour found in the solution of the advection equation and in particular it is apparent the progressive diffusion of the two travelling packets which spread over the numerical grid as they propagate. As expected, the evolution is not stable and no error growth is visible many crossing times after the wave has left the numerical grid.

5.3 The Leapfrog Scheme

We can adapt the Leapfrog scheme to equations (5.4) for the solution of the wave equation in one dimension, centring variables on appropriate half-mesh points
\[
r_{j+\frac{1}{2}}^{n+1} := v \frac{\partial u^n}{\partial x}_{j+\frac{1}{2}} = v \frac{u_{n+1}^j - u_{n}^j}{\Delta x} + O(\Delta x),
\] (5.11)
\[
s_{j+\frac{1}{2}}^{n+1} := \frac{\partial u^n}{\partial t}_{j} = \frac{u_{n+1}^j - u_{n}^j}{\Delta t} + O(\Delta t),
\] (5.12)
Figure 5.2: The same as in Fig. 5.1 but when the Lax-Friedrichs scheme is used. Note the absence of the late time instabilities but also the effects of the numerical diffusion that widens and lowers the wave fronts.

and then considering the Leapfrog representation of equations (5.4)

\[ r_{j + \frac{1}{2}}^{n+1} = r_{j + \frac{1}{2}}^n + \alpha \left( s_{j+1}^{n+\frac{1}{2}} - s_{j}^{n+\frac{1}{2}} \right) + \mathcal{O}(\Delta x^2), \quad (5.13) \]

\[ s_j^{n+\frac{1}{2}} = s_j^{n-\frac{1}{2}} + \alpha \left( r_{j+\frac{1}{2}}^n - r_{j-\frac{1}{2}}^n \right) + \mathcal{O}(\Delta x^2), \quad (5.14) \]

As in the previous examples, the new value for the wave variable \( u \) is finally computed after the integration in time of \( s \). Here however, to preserve the second-order accuracy in time it is necessary to average the time derivative \( s \) between \( n \) and \( n + 1 \) to obtain

\[ u_j^{n+1} = u_j^n + \frac{\Delta t}{2} \left( s_{j}^{n+1} + s_{j-1}^{n} \right) + \mathcal{O}(\Delta x^2) = u_j^n + \frac{\Delta t}{2} s_{j}^{n+1/2} + \mathcal{O}(\Delta x^2). \quad (5.15) \]

A simple substitution of (5.11) and (5.12) into (5.13) and (5.14) shows how the Leapfrog representation of the wave equation is nothing but its second-order differenc-
5.3. THE LEAPFROG SCHEME

Figure 5.3: The same as in Fig. 5.1 but when the Leapfrog scheme is used. Note the absence of the late time instabilities and of the effects of the numerical diffusion.

\[
\frac{u_j^{n+1} - 2u_j^n + u_j^{n-1}}{\Delta t^2} = v^2 \left( \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2} \right) + O(\Delta t^2, \Delta x^2),
\]

(5.16)

so that the solution at the new time-level is

\[
\begin{align*}
    u_j^{n+1} &= \alpha^2 u_{j+1}^n + 2u_j^n (1 - \alpha^2) + \alpha^2 u_{j-1}^n - u_j^{n-1} + O(\Delta x^4). \\
\end{align*}
\]

(5.17)

Note that as formulated in (5.17), the Leapfrog scheme has been effectively recast into a “one-level” scheme.

The solution of equations (5.17) and (5.15) with the same initial data used in Fig. 5.1 is shown in Fig. 5.3. Note that we do not encounter here a significant amount of diffusion for the two travelling wave packets. As expected, the evolution is stable and no error growth is visible many crossing times after the wave has left the numerical grid.
5.4 The Lax-Wendroff Scheme

Also in the case, the application of this scheme to our system of equations (5.4) is straightforward. We can start with the time evolution of the variable \( r \) to obtain

\[
\begin{align*}
  r_j^{n+1} &= r_j^n + \alpha \left( s_{j+1/2}^{n+1/2} - s_{j-1/2}^{n+1/2} \right) + O(\Delta x^2),
\end{align*}
\]

(5.18)

where the terms in the spatial derivatives are computed as

\[
\begin{align*}
  s_{j+1/2}^{n+1/2} &= \frac{1}{2} \left( s_j^n + s_{j+1}^n \right) + \alpha \left( r_{j+1}^n - r_j^n \right) + O(\Delta x^2),
  \quad \text{(5.19)}
\end{align*}
\]

\[
\begin{align*}
  s_{j-1/2}^{n+1/2} &= \frac{1}{2} \left( s_j^n + s_{j-1}^n \right) + \alpha \left( r_j^n - r_{j-1}^n \right) + O(\Delta x^2).
  \quad \text{(5.20)}
\end{align*}
\]

As done for the advection equation, it is convenient not to use equations (5.18) and (5.19) as two coupled but distinct equations and rather to combine them into two “one-level” evolution equations for \( r \) and \( s \)

\[
\begin{align*}
  r_j^{n+1} &= r_j^n + \alpha \left[ \frac{1}{2} (s_{j+1}^n - s_{j-1}^n) + \frac{\alpha}{2} (r_{j+1}^n - 2r_j^n + r_{j-1}^n) \right] + O(\Delta x^2),
  \quad \text{(5.21)}
\end{align*}
\]

\[
\begin{align*}
  s_j^{n+1} &= s_j^n + \alpha \left[ \frac{1}{2} (r_{j+1}^n - r_{j-1}^n) + \frac{\alpha}{2} (s_{j+1}^n - 2s_j^n + s_{j-1}^n) \right] + O(\Delta x^2).
  \quad \text{(5.22)}
\end{align*}
\]

The solution of equations (5.21), (5.22) and (5.15) with the same initial data used in Fig. 5.1 is shown in Fig. 5.4. Note that we do not encounter here a significant amount of diffusion for the two travelling wave packets. As expected, the evolution is stable and no error growth is visible many crossing times after the wave has left the numerical grid.
5.4. THE LAX-WENDROFF SCHEME

Figure 5.4: The same as in Fig. 5.1 but when the Lax-Wendroff scheme is used. Note the absence of the late time instabilities and of the effects of the numerical diffusion.
CHAPTER 5. THE WAVE EQUATION IN 1D
Chapter 6

Boundary Conditions

Unavoidable and common to all the numerical schemes discussed so far is the problem of treating the solution on the boundaries of the spatial grid as the time evolution proceeds. Let \( i \) be the first gridpoint and \( J \) the last one. It is clear from equations (3.26), (5.16), (5.21) and (5.22) that the new solution at the boundaries of the spatial grid (i.e., \( u^{n+1}_i, u^{n+1}_{J+1} \)) is undetermined as it requires the values \( u^n_0, u^n_{J+1} \). The most natural boundary conditions for the evolution of a wave equation are the so called Sommerfeld boundary conditions (or radiative boundary conditions) which will be discussed in the following Section. Other boundary conditions of general interest are:

- **Dirichlet-type** boundary conditions: values of the relevant quantity are imposed at the boundaries of the numerical grid. These values can be either functions of time or be held constant (cf. boundary conditions for boundary value problems);

  - "Periodic" boundary conditions: assume that the numerical domain is topologically connected in a given direction; this is often used in cosmological simulations (and "videogames").

- **von Neumann-type** boundary conditions: values of the derivatives of the relevant quantity are imposed at the boundaries of the numerical grid. As for Dirichlet, these values can be either functions of time or be held constant (cf. boundary conditions for boundary value problems);

  - "Reflecting" boundary conditions: mimic the presence of a reflecting boundary, i.e., of a boundary with zero transmission coefficient;

  - "Absorbing" boundary conditions: mimic the presence of an absorbing boundary, i.e., of a boundary with unit transmission coefficient;

6.1 Outgoing Wave BCs: the outer edge

A scalar wave outgoing in the positive \( x \)-direction is described by the advection equation:

\[
\frac{\partial u}{\partial t} + v \frac{\partial u}{\partial x} = 0
\]  

(6.1)
A finite-difference, first-order accurate representation of equation (6.1) which is centered in both time (at \( n + \frac{1}{2} \)) and in space (at \( j + \frac{1}{2} \)) is given by (see Fig. 3.11)

\[
\frac{1}{2\Delta t} \left[ (u_{j+1}^{n+1} + u_j^{n+1}) - (u_{j+1}^n + u_j^n) \right] = -\frac{v^2}{2\Delta x} \left[ (u_{j+1}^{n+1} + u_j^n) - (u_j^{n+1} + u_j^n) \right]
\]

and which leads to

\[
u_j^{n+1} (1 + \alpha) = u_j^{n+1} (-1 + \alpha) + u_{j+1}^{n+1} (1 - \alpha) + u_j^n (1 + \alpha) \tag{6.2}
\]

Expression (6.2) can also be written as

\[
u_j^{n+1} = u_j^n - u_j^{n+1} Q + u_{j+1}^n Q, \tag{6.3}
\]

where

\[Q := \frac{1 - \alpha}{1 + \alpha}. \tag{6.4}\]

The use of expression (6.3) for the outermost grid point where the wave is outgoing will provide first-order accurate and stable boundary conditions. Note, however, that (6.3) is a discrete representation of a physical condition which would transmit the wave without reflection. In practice, however, a certain amount of reflection is always produced (the transmission coefficient is never exactly one); the residual wave is then transmitted back in the numerical box. A few reflections are usually sufficient to reduce the wave content to values below the machine accuracy.
6.2  Ingoing Wave BCs: the inner edge

Similarly, a scalar wave outgoing in the negative $x$-direction (or ingoing in the positive one) is described by the advection equation:

$$\frac{\partial u}{\partial t} - \frac{\partial u}{\partial x} = 0$$  \hspace{1cm} (6.5)

Following the same procedure discussed before, the algorithm becomes:

$$u^{n+1}_j \left(1 + \frac{\Delta t}{\Delta x}\right) = -u^{n+1}_{j+1} \left(1 - \frac{\Delta t}{\Delta x}\right) + u^{n+1}_{j} \left(1 + \frac{\Delta t}{\Delta x}\right) + u^{n}_j \left(1 - \frac{\Delta t}{\Delta x}\right)$$

Then

$$u^{n+1}_j = u^n_j - u^{n+1}_{j+1}Q + u^n_jQ,$$  \hspace{1cm} (6.6)

where $Q$ is the same quantity as for the out-going wave. If we use equations (6.3) and (6.6) to evolve the solution at time-step $n+1$ at the boundary of our spatial grid, we are guaranteed that our profile will be completely transported away, whatever integration scheme we are adopting (Leapfrog, Lax-Wendroff etc.).

6.3  Periodic Boundary Conditions

These are very simple to impose and if $j$ is between 1 and $J$, they are given simply by

$$u^{n+1}_1 = u^{n+1}_{J-1}, \quad u^{n+1}_J = u^{n+1}_2.$$  \hspace{1cm} (6.7)

In the case of a Gaussian leaving the center of the numerical grid, these boundary conditions effectively produce a reflection. The boundary conditions (6.7) force to break the algorithm for the update scheme excluding the first and last points that need to be computed separately. An alternative procedures consists of introducing a number of “ghost” gridpoints outside the computational domain of interest so that the solution is calculated using always the same stencil for $j = 1, 2, \ldots, J$ and exploiting the knowledge of the solution also at the ghost gridpoints, e.g., 0 and $J+1$.

In the case there is only one ghost gridpoint at either edge of the 1D grid, the boundary conditions are simply given by

$$u^{n+1}_0 = u^{n+1}_J, \quad u^{n+1}_{J+1} = u^{n+1}_1.$$  \hspace{1cm} (6.8)
Chapter 7

The wave equation in two spatial dimensions (2D)

We will now extend the procedures studied so far to the case of a wave equation in two dimensions

$$\frac{\partial^2 u}{\partial t^2} = v^2 \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right). \quad (7.1)$$

As for the one-dimensional case, also in this case the wave equation can be reduced to the solution of a set of three first-order advection equations

$$\frac{\partial r}{\partial t} = v \frac{\partial s}{\partial x}, \quad (7.2)$$
$$\frac{\partial l}{\partial t} = v \frac{\partial s}{\partial y}, \quad (7.3)$$
$$\frac{\partial s}{\partial t} = v \left( \frac{\partial r}{\partial x} + \frac{\partial l}{\partial y} \right), \quad (7.4)$$

once the following definitions have been made

$$r = v \frac{\partial u}{\partial x}, \quad (7.5)$$
$$l = v \frac{\partial u}{\partial y}, \quad (7.6)$$
$$s = \frac{\partial u}{\partial t}. \quad (7.7)$$

In vector notation the system can again be written as

$$\frac{\partial U}{\partial t} + \nabla F(U) = 0, \quad (7.8)$$
where

\[ U = \begin{pmatrix} r \\ l \\ s \end{pmatrix}, \quad \text{and} \quad F(U) = \begin{pmatrix} -v & 0 & 0 \\ 0 & -v & 0 \\ 0 & 0 & -v \end{pmatrix} \cdot U = -v \begin{pmatrix} r \\ l \\ s \end{pmatrix}, \]

provided we define

\[ \nabla := \begin{pmatrix} 0 & 0 & \frac{\partial}{\partial x} \\ 0 & 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & 0 \end{pmatrix}. \]

The finite-difference notation should also be extended to account for the two spatial dimension and we will then assume that \( u_{n,i,j} := u(x_i, y_j, t^n) \).

### 7.1 The Lax-Friedrichs Scheme

We can look at the system of equations (7.2) and (7.3) as a set of two equations to be integrated with the procedures so far developed in one-dimension. Furthermore, we need to solve for Eq. (7.4) which can be written as

\[ \frac{\partial s}{\partial t} = \frac{-\partial F_x}{\partial x} - \frac{\partial F_y}{\partial y} \]  

once we identify \( F_x \) with \(-vr\) and \( F_y \) with \(-vl\).

The Lax-Friedrichs scheme for this equation is just the generalisation of the 1D expressions discussed so far and yields

\[ s_{i,j}^{n+1} = \frac{1}{4} \left[ s_{i+1,j}^n + s_{i-1,j}^n + s_{i,j+1}^n + s_{i,j-1}^n \right] - \frac{\Delta t}{2\Delta x} \left[ (F_x^n)_{i+1,j} - (F_x^n)_{i-1,j} \right] - \frac{\Delta t}{2\Delta y} \left[ (F_y^n)_{i,j+1} - (F_y^n)_{i,j-1} \right], \]

\[ \xi = \frac{1}{2} \left[ \cos(k_x \Delta x) + \cos(k_y \Delta y) \right] - i[\alpha_x \sin(k_x \Delta x) + \alpha_y \sin(k_y \Delta y)] , \]

and where \( \alpha_x = \frac{v_x}{\Delta x} \) and \( \alpha_y = \frac{v_y}{\Delta y} \).
7.2. THE LAX-WENDROFF SCHEME

The 2D generalisation of the one-dimensional scheme (3.43) is also straightforward and can be described as follows:

where

\[ \alpha_x := \frac{v_x \Delta t}{\Delta x}, \quad \alpha_y := \frac{v_y \Delta t}{\Delta x}. \]  \tag{7.14}

Stability is therefore obtained if

\[ \frac{1}{2} - (\alpha_y^2 + \alpha_x^2) \geq 0, \]  \tag{7.15}

or, equally, if

\[ \Delta t \leq \frac{\Delta x}{\sqrt{2(v_x^2 + v_y^2)}}, \]  \tag{7.16}

Expression (7.16) represents the 2D extension of the CFL stability condition. In general, for a \( N \)-dimensional space, the CFL stability condition can be expressed as

\[ \Delta t \leq \min \left( \frac{\Delta x_i}{\sqrt{N \bar{v}}} \right), \]  \tag{7.17}

where \( i = 1, \ldots, N \) and \( \bar{v} := (\sum_{i=1}^{N} v_i^2)^{1/2} \). Note, in 2D, the appearance of an averaging coefficient \( 1/4 \) multiplying the value of the function at the time-level \( n \).

### 7.2 The Lax-Wendroff Scheme

The 2D generalisation of the one-dimensional scheme (3.43) is also straightforward and can be described as follows:
1. Compute \( r, l \) and \( s \) at the half-time using a half-step Lax-Friedrichs scheme

\[
\begin{align*}
  r_{i,j}^{n+\frac{1}{2}} &= \frac{1}{2} \left[ \left( r_{i+1,j}^n + r_{i-1,j}^n \right) + \alpha_x \left( s_{i+1,j}^n - s_{i-1,j}^n \right) \right], \\
  r_{i,j}^{-\frac{1}{2}} &= \frac{1}{2} \left[ \left( r_{i,j}^n + r_{i-1,j}^n \right) + \alpha_x \left( s_{i,j}^n - s_{i-1,j}^n \right) \right], \\
  l_{i,j}^{n+\frac{1}{2}} &= \frac{1}{2} \left[ \left( l_{i+1,j}^n + l_{i,j}^n \right) + \alpha_y \left( s_{i+1,j}^n - s_{i,j}^n \right) \right], \\
  l_{i,j}^{n-\frac{1}{2}} &= \frac{1}{2} \left[ \left( l_{i,j}^n + l_{i-1,j}^n \right) + \alpha_y \left( s_{i,j}^n - s_{i-1,j}^n \right) \right], \\
  s_{i,j}^{n+\frac{1}{2}} &= \frac{1}{2} \left[ \left( s_{i,j+1}^n + s_{i,j-1}^n \right) + \alpha_x \left( r_{i,j+1}^n - r_{i,j-1}^n \right) + \frac{\alpha_y}{2} \left( l_{i,j+1}^n - l_{i,j-1}^n \right) \right], \\
  s_{i,j}^{n-\frac{1}{2}} &= \frac{1}{2} \left[ \left( s_{i,j}^n + s_{i,j-1}^n \right) + \alpha_x \left( r_{i,j}^n - r_{i,j-1}^n \right) + \frac{\alpha_y}{2} \left( l_{i,j}^n - l_{i,j-1}^n \right) \right], \\
  s_{i,j+\frac{1}{2}} &= \frac{1}{2} \left[ \left( s_{i+1,j}^n + s_{i,j}^n \right) + \frac{\alpha_x}{2} \left( r_{i+1,j}^n - r_{i,j}^n \right) + \alpha_y \left( l_{i+1,j}^n - l_{i,j}^n \right) \right], \\
  s_{i,j-\frac{1}{2}} &= \frac{1}{2} \left[ \left( s_{i,j}^n + s_{i-1,j}^n \right) + \frac{\alpha_x}{2} \left( r_{i,j}^n - r_{i-1,j}^n \right) + \alpha_y \left( l_{i,j}^n - l_{i-1,j}^n \right) \right],
\end{align*}
\]

where \( \alpha_x := v \Delta t / \Delta x \) and \( \alpha_y := v \Delta t / \Delta y \).

2. Evolve \( r, l \) and \( s \) to the time-level \( n+1 \) using a half-step Leapfrog scheme

\[
\begin{align*}
  r_{i,j}^{n+1} &= r_{i,j}^n + \alpha_x \left( s_{i+\frac{1}{2},j}^{n+\frac{1}{2}} - s_{i-\frac{1}{2},j}^{n+\frac{1}{2}} \right), \\
  l_{i,j}^{n+1} &= l_{i,j}^n + \alpha_y \left( s_{i,j+\frac{1}{2}}^{n+\frac{1}{2}} - s_{i,j-\frac{1}{2}}^{n+\frac{1}{2}} \right), \\
  s_{i,j}^{n+1} &= s_{i,j}^n + \alpha_x \left( r_{i+\frac{1}{2},j}^{n+\frac{1}{2}} - r_{i-\frac{1}{2},j}^{n+\frac{1}{2}} \right) + \alpha_y \left( l_{i,j+\frac{1}{2}}^{n+\frac{1}{2}} - l_{i,j-\frac{1}{2}}^{n+\frac{1}{2}} \right).
\end{align*}
\]

3. Update \( u \) to the time-level \( n+1 \), i.e.,

\[
u_{i,j}^{n+1} = u_{i,j}^n + \frac{\Delta t}{2} \left( s_{i,j}^{n+1} + s_{i,j}^n \right).
\]
7.3. THE LEAPFROG SCHEME

Alternatively, steps 1. and 2. can be combined analytically to yield the direct integration of \( r, l \) and \( s \) from level \( n \) to level \( n + 1 \) as

\[
r_{i,j}^{n+1} = r_{i,j}^{n} + \alpha_x \left[ \frac{1}{2} (s_{i+1,j}^{n} - s_{i-1,j}^{n}) + \frac{\alpha_x}{2} (r_{i+1,j}^{n} - 2r_{i,j}^{n} + r_{i-1,j}^{n}) \right],
\]

\[
l_{i,j}^{n+1} = l_{i,j}^{n} + \alpha_y \left[ \frac{1}{2} (s_{i,j+1}^{n} - s_{i,j-1}^{n}) + \frac{\alpha_y}{2} (l_{i,j+1}^{n} - 2l_{i,j}^{n} + l_{i,j-1}^{n}) \right],
\]

\[
s_{i,j}^{n+1} = s_{i,j}^{n} + \alpha_x \left[ \frac{1}{2} (r_{i+1,j}^{n} - r_{i-1,j}^{n}) + \frac{\alpha_x}{2} (s_{i+1,j}^{n} - 2s_{i,j}^{n} + s_{i-1,j}^{n}) \right] + \alpha_y \left[ \frac{1}{2} (l_{i,j+1}^{n} - l_{i,j-1}^{n}) + \frac{\alpha_y}{2} (s_{i,j+1}^{n} - 2s_{i,j}^{n} + s_{i,j-1}^{n}) \right],
\]

(7.30)

(7.31)

(7.32)

Such a form of the equations is simpler to implement and allows a transparent distinction between the advective and the dissipative terms produced by the presence of second spatial-derivatives terms. More specifically, the system above can be written in a compact form as

\[
r_{i,j}^{n+1} = r_{i,j}^{n} + \frac{\alpha_x}{2} D_x s_{i,j}^{n} + \frac{\alpha_x^2}{2} D_{xx} r_{i,j}^{n},
\]

\[
l_{i,j}^{n+1} = l_{i,j}^{n} + \frac{\alpha_y}{2} D_y s_{i,j}^{n} + \frac{\alpha_y^2}{2} D_{yy} l_{i,j}^{n},
\]

\[
s_{i,j}^{n+1} = s_{i,j}^{n} + \frac{\alpha_x}{2} D_x r_{i,j}^{n} + \frac{\alpha_y}{2} D_y l_{i,j}^{n} + \frac{\alpha_x^2}{2} D_{xx} s_{i,j}^{n} + \frac{\alpha_y^2}{2} D_{yy} s_{i,j}^{n},
\]

(7.33)

(7.34)

(7.35)

where we have omitted for compactness the spatial indices and we have used the compact notation

\[
D_x \phi_{i,j} := \phi_{i+1,j} - \phi_{i-1,j}, \quad D_y \phi_{i,j} := \phi_{i,j+1} - \phi_{i,j-1},
\]

\[
D_{xx} \phi_{i,j} := \phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}, \quad D_{yy} \phi_{i,j} := \phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}.
\]

(7.36)

(7.37)

Expressions (7.33)-(7.37) can be easily extended to higher spatial dimensions and readily implemented in a recursive loop.

7.3 The Leapfrogs Scheme

The 2D generalisation of the one-dimensional scheme (5.16) is less straightforward, but not particularly difficult. As in one dimension, we can start by rewriting directly the finite-difference form of the wave equation as

\[
\frac{u_{i,j}^{n+1} - 2u_{i,j}^{n} + u_{i,j}^{n-1}}{\Delta t^2} = v^2 \left( \frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{\Delta x^2} \right) + v^2 \left( \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{\Delta y^2} \right)
\]
so that, after some algebra, we obtain the explicit form

\[ u_{i,j}^{n+1} = \alpha^2 \left[ u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n \right] + 2u_{i,j}^n (1 - 2\alpha^2) - u_{i,j}^{n-1}. \] (7.38)

The stencil relative to the algorithm (7.38) is illustrated in Fig. 7.2.

Figs. 7.3 and 7.4 show the solution of the wave equation in 2D using the scheme (7.38) and imposing Sommerfeld outgoing-wave boundary conditions at the edges of the numerical grid.

Radically different appears the evolution when reflective boundary conditions are imposed, as it is illustrated in Figs 4. Note that the initial evolution (i.e., for which the effects of the boundaries are negligible) is extremely similar to the one shown in Figs. 4, but becomes radically different when the wavefront has reached the outer boundary. As a result of the high (but not perfect!) reflectivity of the outer boundaries, the wave is “trapped” inside the numerical grid and bounces back and forth producing the characteristic interference patterns.
Figure 7.3: Plot of the time evolution of the wave equation when the Leapfrog scheme in 2D is used and Sommerfeld boundary conditions are imposed. Snapshots at increasing times are illustrated in a clockwise sequence.
Figure 7.4: Plot of the time evolution of the wave equation when the Leapfrog scheme in 2D is used and Reflecting boundary conditions are applied. Snapshots at increasing times are illustrated in a clockwise sequence.
Figure 7.5: Plot of the time evolution of the wave equation when the Leapfrog scheme in 2D is used and Reflecting boundary conditions are applied.
Figure 7.6: Plot of the time evolution of the 2-norm when the Leapfrog scheme in 2D is used. Note the radically different behaviour between Sommerfeld and reflecting boundary conditions.
Chapter 8

Parabolic PDEs

8.1 Diffusive problems

The inclusion of viscosity in the description of a fluid leads to non trivial complications in the numerical solution of the hydrodynamic equations. From an analytical point of view, the resulting equations are no longer purely hyperbolic PDE’s but rather mixed hyperbolic-parabolic PDE’s. This means that the numerical method used to solve them must necessarily be able to cope with the parabolic part of the equations. It is therefore convenient to fully understand the prototypical parabolic equation, the one-dimensional diffusion equation, both analytically and numerically, before attempting to solve any mixed hyperbolic-parabolic PDE.

8.2 The diffusion equation in 1D

The description of processes like the heat conduction in a solid body or the spread of a dye in a motionless fluid is given by the one-dimensional diffusion equation

$$\frac{\partial u(x,t)}{\partial t} = D \frac{\partial^2 u(x,t)}{\partial x^2}. \quad (8.1)$$

Here $D > 0$ is a constant coefficient that determines the magnitude of the “diffusion” in the process under investigation (being given by the thermal conductivity and dye diffusion coefficient respectively in the above mentioned examples).

In what follows, some numerical methods will be presented to solve a simple diffusive problem in 1+1 dimensions distinguishing explicit methods from implicit methods. A semi-analytic solution of the model parabolic equation (8.1) is presented in Appendix A.

69
8.3 Explicit updating schemes

8.3.1 The FTCS method

The most straightforward way to finite difference equation (8.1) is by the FTCS method, i.e.,

\[
\frac{u_{j}^{n+1} - u_{j}^{n}}{\Delta t} = D \frac{u_{j+1}^{n} - 2u_{j}^{n} + u_{j-1}^{n}}{\Delta x^2} + O(\Delta t, \Delta x^2),
\]

(8.2)

Unlike for a hyperbolic equation, where the FTCS method leads to an unconditionally unstable method, the presence of a second space derivative in the model parabolic equation (8.1) allows the FTCS method to be conditionally stable [10]. A von Neumann stability analysis leads in fact to the stability criterion

\[
\gamma := 2 \frac{\Delta t}{\Delta x^2} \leq 1,
\]

(8.3)

that lends itself to a physical interpretation: the maximum time step is, up to a numerical factor, the diffusion time across a cell of width \( \Delta x \). This stability condition poses a serious limit in the use of the above scheme since the typical timescales of interest will require a number of timesteps which could be prohibitive in multidimensional calculations. The additional fact that the overall scheme is first-order accurate in time only strengthens the need for a different method.

8.3.2 The Du Fort-Frankel method and the \( \theta \)-method

With this objective in mind, it is not difficult to think of a way to avoid the reduced accuracy due to the forward-time finite differencing approach used in FTCS. A simple time-centered finite differencing

\[
\frac{u_{j}^{n+1} - u_{j}^{n-1}}{2\Delta t} = D \frac{u_{j+1}^{n} - u_{j-1}^{n}}{\Delta x^2}
\]

(8.4)

should grant second-order accuracy. Unfortunately, this method is unconditionally unstable. To overcome the stability problem, Du Fort and Frankel [12] suggested the following scheme

\[
\frac{u_{j}^{n+1} - u_{j}^{n-1}}{2\Delta t} = D \frac{u_{j+1}^{n} - u_{j-1}^{n}}{\Delta x^2},
\]

(8.5)

which is obtained from (8.4) with the substitution of \( u_{j}^{n} \) with \( \frac{1}{2} (u_{j}^{n+1} + u_{j}^{n-1}) \), that is, by taking the average of \( u_{j}^{n+1} \) and \( u_{j}^{n-1} \), i.e.,

\[
u_{j}^{n+1} = \left( 1 - \frac{\gamma}{1 + \gamma} \right) u_{j}^{n-1} + \left( \frac{\gamma}{1 + \gamma} \right) \left( u_{j+1}^{n} + u_{j-1}^{n} \right) + O(\Delta x^2).
\]

(8.6)

With this substitution, the method is still explicit and becomes unconditionally stable, but not without a price. A consistency analysis shows, in fact, that the Du Fort-Frankel
8.3. EXPLICIT UPDATING SCHEMES

method could be inconsistent. The local truncation error is \[8\]

\[
\epsilon = \frac{\Delta t^2}{6} \frac{\partial^3 u}{\partial t^3} - \frac{\Delta x^2}{12} \frac{\partial^4 u}{\partial x^4} j,n + \left( \frac{\Delta t}{\Delta x} \right)^2 \frac{\partial^2 u}{\partial t^2} j,n + \ldots \quad (8.7)
\]

which shows that if \(\Delta t\) and \(\Delta x\) tend to zero at the same rate, \(i.e., \Delta t = k\Delta x\) with \(k\) being a constant, then the truncation error does not vanish for \(\Delta t \to 0\) and \(\Delta x \to 0\). Indeed, the solution obtained with this method will effectively be the solution to equation

\[
\frac{\partial u(x, t)}{\partial t} + k^2 \frac{\partial^2 u(x, t)}{\partial x^2} = D \frac{\partial^2 u(x, t)}{\partial x^2}, \quad (8.9)
\]

and not the solution of (8.1), which is also known as the “telegraph equation” (see [9] for a discussion).

On the other hand, it is also clear from (8.7) that having a timestep \(\Delta t = k\Delta x^{1+\varepsilon}\) with \(\varepsilon > 0\) will assure the consistency of the method. Of course, the closer is \(\varepsilon\) to 1, the smaller will have to be \(\Delta x\) in order to achieve consistency. Moreover, accuracy requirements pose an additional constraint on \(\varepsilon\). For a first-order method it is necessary to have \(\varepsilon = 1/2\), while to achieve second-order accuracy the requirement is \(\varepsilon = 1\). It would be pointless and computationally inefficient to set \(\varepsilon > 1\) since in this case the dominant contribution to the truncation error would be determined by the term \(O(\Delta x^2)\) which acts as an upper limit to the overall accuracy order. This means that \(\varepsilon\) is constrained to be in the interval \(1/2 \leq \varepsilon \leq 1\).

The advantages of the Du Fort-Frankel method over the FTCS scheme should now be easily seen. To achieve first-order accuracy, a timestep \(\Delta t = (\Delta x)^{3/2}\) is needed with the Du Fort-Frankel method, while the FTCS scheme requires \(\Delta t \approx (\Delta x)^2\), hence smaller timesteps. On the other hand, if a timestep \(\Delta t = (\Delta x)^2\) is used, then the Du Fort-Frankel method gains second-order accuracy. Finally, any desired accuracy between first and second order could be achieved with a timestep that is independent of the diffusion coefficient \(D\). The only minor drawback of the Du Fort-Frankel scheme lies in the requirement of keeping track of an additional time level.

A generalisation of the Du Fort-Frankel scheme is also straightforward. In particular, when averaging \(u^{n+1}_j\) and \(u^{-1}_j\), instead of weighting them equally, it is possible to average them with different weights. The resulting update scheme is therefore

\[
\frac{u^{n+1}_j - u^{n-1}_j}{2\Delta t} = D u^{n+1}_j + 2 (\theta u^{n+1}_j - (1 - \theta) u^{n-1}_j) + u^{n}_j, \quad (8.10)
\]

where \(\theta\) is a variable parameter. In [8] it is shown that the local truncation error for this scheme is

\[
\frac{\Delta t^2}{6} \frac{\partial^3 u}{\partial t^3} j,n - D \frac{\Delta x^2}{12} \frac{\partial^4 u}{\partial x^4} j,n + (2\theta - 1) \frac{2\Delta t}{\Delta x^2} \frac{\partial u}{\partial t} j,n + \frac{\Delta t^2}{\Delta x^2} \frac{\partial^2 u}{\partial t^2} j,n + O \left( \frac{\Delta t^2}{\Delta x^2}, \Delta t^4, \Delta x^4 \right), \quad (8.11)
\]

\[
\frac{\Delta t^2}{\Delta x^2} \frac{\partial^2 u}{\partial t^2} j,n + O \left( \frac{\Delta t^2}{\Delta x^2}, \Delta t^4, \Delta x^4 \right), \quad (8.12)
\]
which clearly shows that consistency could be achieved for any value of $\theta$ but only if
$\Delta t = k \Delta x^2 + \varepsilon$ with $\varepsilon$ and $k$ being positive real numbers. If $\theta = 1/2$, on the other
hand, the scheme is actually the Du Fort-Frankel scheme [cf. expression (8.7)] with the
consistency constraints already outlined above. It is therefore clear that, when
solving equation (8.1), timestep considerations show that the only viable $\theta$-scheme is
the $\theta = 1/2$ scheme, i.e., the Du Fort-Frankel scheme.

### 8.3.3 ICN as a $\theta$-method

We next extend the stability analysis of the $\theta$-ICN discussed in Sect. 3.6.1 to the a
parabolic partial differential equation and use as model equation the one-dimensional
diffusion equation (8.1). Parabolic equations are commonly solved using implicit
methods such as the Crank-Nicolson, which is unconditionally stable and thus removes
the constraints on the timestep [i.e., $\Delta t \approx O(\Delta x^2)$] imposed by explicit schemes [10].
In multidimensional calculations, however, or when the set of equations is of mixed
hyperbolic-parabolic type, implicit schemes can be cumbersome to implement since
the resulting system of algebraic equations does no longer have simple and tridiagonal
matrices of coefficients. In this case, the most convenient choice may be to use an
explicit method such as the ICN.

Also in this case, the first step in our analysis is the derivation of a finite-difference
representation of the right-hand-side of eq. (8.1) which, at second-order, has the form

$$\mathcal{L}_\Delta (u^n_{j,j+1}) = \frac{u^n_{j+1} - 2u^n_j + u^n_{j-1}}{\Delta x^2} + O(\Delta x^2) \quad .$$

(8.13)

**Constant Arithmetic Averages**

Next, we consider first the case with constant arithmetic averages (i.e., $\theta = 1/2$) and
the expression for the amplification factor after $M$-iterations is then purely real and
given by

$$(^M)\xi = 1 + 2 \sum_{n=1}^{M} (-\gamma)^n \quad ,$$

(8.14)

where $\gamma := (2D\Delta t/\Delta x^2) \sin^2(k\Delta x/2)$. Requiring now for stability that $\sqrt{\xi^2} \leq 1$
and bearing in mind that

$$-1 \leq \sum_{n=0}^{M} (-\gamma)^{n+1} \leq 0 \quad , \quad \text{for } \gamma \leq 1 \quad ,$$

(8.15)

we find that the scheme is stable for any number of iterations provided that $\gamma \leq 1$.
Furthermore, because the scheme is second-order accurate from the first iteration on,
our suggestion when using the ICN method for parabolic equations is that one iteration
should be used and no more. In this case, in particular, the ICN method coincides with
a FTCS scheme [10].

Note that the stability condition $\gamma \leq 1$ introduces again a constraint on the timestep
that must be $\Delta t \leq \Delta x^2/(2D)$ and thus $O(\Delta x^2)$. As a result and at least in this respect,
8.3. EXPLICIT UPDATING SCHEMES

Figure 8.1: Left panel: stability region in the $(\theta, \gamma)$ plane for the two-iterations $\theta$-ICN for the diffusion equation (8.1). Thick solid lines mark the limit at which $\xi^2 = 1$, while the dotted contours indicate the values of the amplification factor in the stable region. Right panel: same as in the left panel but with swapping the averages between two corrections.

the ICN method does not seem to offer any advantage over other explicit methods for the solution of a parabolic equation\(^1\).

Constant Weighted Averages

We next consider the stability of the $\theta$-ICN method but focus our attention on a two-iterations scheme since this is the number of iterations needed in the solution of the parabolic part in a mixed hyperbolic-parabolic equation when, for instance, operator-splitting techniques are adopted [10]. In this case, the amplification factor is again purely real and given by

$$\xi = 1 - 2\gamma + 4\gamma^2 \theta - 8\gamma^3 \theta^2,$$

(8.16)

so that stability is achieved if

$$0 \leq \gamma \left( 1 - 2\theta \gamma + 4\theta^2 \gamma^2 \right) \leq 1.$$  

(8.17)

Since $\gamma > 0$ by definition, the left inequality is always satisfied, while the right one is true provided that, for $\gamma < 4/3$,

$$\frac{\gamma - \sqrt{\gamma(4 - 3\gamma)}}{4\gamma^2} \leq \theta \leq \frac{\gamma + \sqrt{\gamma(4 - 3\gamma)}}{4\gamma^2}.$$  

(8.18)

\(^1\)Note that also the Dufort-Frankel method [12], usually described as unconditionally stable, does not escape the timestep constraint $\Delta t \approx O(\Delta x^2)$ when a consistent second-order accurate solution is needed [8].
The stability region described by the condition (8.18) is shown in the left panel of Fig. 8.1 for \( \sin k\Delta x = 1 \) and illustrates that the scheme is stable for any value \( 0 \leq \theta \leq 1 \), and also that slightly larger timesteps can be taken when \( \theta \approx 0.2 \).

**Swapped Weighted Averages**

After some lengthy algebra the calculation of the amplification factor for the \( \theta \)-ICN method with swapped weighted averages yields

\[
\xi = 1 - 2\gamma + 4\gamma^2\theta - 8\gamma^3\theta(1 - \theta),
\]

and stability is then given by

\[
-1 \leq 1 - 2\gamma + 4\gamma^2\theta - 8\gamma^3\theta(1 - \theta) \leq 1.
\]

Note that none of the two inequalities is always true and in order to obtain analytical expressions for the stable region we solve the condition (8.20) with respect to \( \theta \) and obtain

\[
\theta \leq \frac{2\gamma - 1 + \sqrt{4\gamma^2 - 4\gamma + 5}}{4\gamma},
\]

\[
\theta \leq \frac{\gamma(2\gamma - 1) - \sqrt{\gamma(4\gamma^3 - 4\gamma^2 + 5\gamma - 4)}}{4\gamma^2},
\]

\[
\theta \geq \frac{\gamma(2\gamma - 1) + \sqrt{\gamma(4\gamma^3 - 4\gamma^2 + 5\gamma - 4)}}{4\gamma^2}.
\]

The resulting stable region for \( \sin k\Delta x = 1 \) is plotted in the right panel of Fig. 8.1 and seems to suggest that arbitrarily large values of \( \gamma \) could be considered when \( \theta \gtrsim 0.6 \). It should be noted, however, that the amplification factor is also severely reduced as larger values of \( \gamma \) are used and indeed it is essentially zero in the limit \( \theta \to 1 \).

### 8.4 Implicit updating schemes

#### 8.4.1 The BTCS method

It is common for explicit schemes to be only conditionally stable and in this respect the Du Fort-Frankel method is somewhat unusual. Implicit methods, on the other hand, do not share this property being typically unconditionally stable. This suggests to apply an implicit finite differencing to equation (8.1) in the form of a “backward-time centered-space” (BTCS) scheme and obtain

\[
\frac{u_j^{n+1} - u_j^n}{\Delta t} = D \frac{u_j^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{\Delta x^2} + O(\Delta t, \Delta x^2).
\]

As a von-Neumann stability analysis shows [10], the amplification factor is given by

\[
\xi = 1 / \left[ 1 + 2\gamma \sin^2 \left( \frac{k\Delta x}{2} \right) \right],
\]

(8.23)
so that the finite-differencing (8.22) is unconditionally stable. This method is also called \textit{backward time}. Rearranging the terms it is easy to obtain

\[
-\gamma u_{j}^{n+1} + 2(1 + \gamma)u_{j}^{n+1} - \gamma u_{j+1}^{n+1} = 2u_{j}^{n},
\]

which shows that to obtain \( u \) at time level \( n + 1 \) is necessary to solve a system of linear equations. Luckily, the system is \textit{tridiagonal}, i.e., only the nearest neighbours of the diagonal term are non zero, which allows the use of \textit{sparse matrix} techniques (a matrix is called sparse if the number of non zero elements is small compared to the number of all the elements). The main disadvantage of this scheme, besides that of requiring the simultaneous solution of \( N \) algebraic equations, is that it is only first-order accurate in time.

\subsection*{8.4.2 The Crank-Nicolson method}

Combining the stability of an implicit method with the accuracy of a method that is second-order both in space and in time is possible and is achieved by averaging explicit FTCS and implicit BTCS schemes:

\[
\frac{u_{j}^{n+1} - u_{j}^{n}}{\Delta t} = \frac{D}{2} \left[ \frac{(u_{j+1}^{n+1} - 2u_{j}^{n+1} + u_{j-1}^{n+1}) + (u_{j+1}^{n} - 2u_{j}^{n} + u_{j-1}^{n})}{\Delta x^2} \right] + O(\Delta t^2, \Delta x^2).
\]

This scheme is called \textit{Crank-Nicolson} and is second-order in time since both the left hand side and the right hand side are centered in \( n + 1/2 \). A more compact and computer-ready representation of the algorithm is then given by

\[
-\frac{\gamma}{4} u_{j-1}^{n+1} + (1 + 2\gamma)u_{j}^{n+1} - \frac{\gamma}{4} u_{j+1}^{n+1} = \frac{\gamma}{4} u_{j-1}^{n} + (1 - 2\gamma)u_{j}^{n} + \frac{\gamma}{4} u_{j+1}^{n} + O(\Delta t^2, \Delta x^2).
\]

The amplification factor in this case is given by

\[
\xi = \left[ 1 - \gamma \sin^2 \left( \frac{k \Delta x}{2} \right) \right] / \left[ 1 + \gamma \sin^2 \left( \frac{k \Delta x}{2} \right) \right],
\]

so that, as with the fully implicit BTCS scheme, the CN scheme is unconditionally stable. For this reason and for being higher order than BTCS, is the best choice for the solution of simple one-dimensional diffusive problems.

The disadvantage of this scheme with respect to an explicit scheme like the Du Fort-Frankel scheme lies in the fact that in more than one dimension the system of linear equation will no longer be tridiagonal, although it will still be sparse. The extension of the Du Fort-Frankel scheme, on the other hand, is straightforward and with the same constraints as in the one-dimensional case. Because of this and other problems which emerge in multidimensional applications, more powerful methods, like the \textit{Alternating Direction Implicit} (ADI) have been developed. ADI embodies the powerful concept of \textit{operator splitting} or \textit{time splitting}, which requires a more detailed explanation and will not be given in these notes.
Appendix A

Semi-analytical solution of the model parabolic equation

In this appendix we present details on the derivation of the semi-analytic solution to

\[ \frac{\partial u(x,t)}{\partial t} = D \frac{\partial^2 u(x,t)}{\partial x^2}, \]  

(A.1)

where \( D \) is a constant coefficient. We will first consider homogeneous Dirichlet and then homogeneous Neumann boundary conditions. Because the initial value \( u(x,0) = h(x) \) is also needed, we will consider two different initial profiles for the two cases. The solutions we will obtain are to be considered semi-analytical in the sense that it is usually necessary to evaluate them numerically. This is so because infinite series and integrals that could not always be evaluated analytically are involved.

A.1 Homogeneous Dirichlet boundary conditions

Consider a generic problem for which equation (8.1) holds over a domain \([0, L]\). Suppose also that the boundary conditions could be written as homogeneous DBC, i.e., \( u(0,t) = u(L,t) = 0 \), and that at time \( t_0 = 0 \) the distribution of \( u(x,t) \) is that shown in Figure A.1, which could be written as

\[
h(x) := u(x,0) = \begin{cases} 
2x/L & \text{if } 0 \leq x \leq L/2 \\
-2x/L + 2 & \text{if } L/2 < x \leq L 
\end{cases}
\]  

(A.2)

while the boundary conditions are \( u(0,t) = u(L,t) = 0 \).

The equation could be solved by means of the separation of variables technique, i.e., by searching for a solution of the form

\[ u(x,t) = f(x)g(t), \]  

(A.3)
which allows to write equation (A.1) as
\[ f \frac{\partial g}{\partial t} = D g \frac{\partial^2 f}{\partial x^2}. \] (A.4)

Multiplying both sides by \( \frac{1}{fg} \) the result is
\[ \frac{1}{g} \frac{\partial g}{\partial t} = D \frac{1}{f} \frac{\partial^2 f}{\partial x^2}. \] (A.5)

The left hand side of (A.5) is a function of \( t \) only while the right hand side depends only on \( x \). Because of that, their common value can only be a constant, with this constant being a negative number because otherwise \( g \to \infty \) (and therefore \( u \to \infty \)) as \( t \to \infty \). Thus the common value could be denoted as \(-\lambda\) with \( \lambda > 0 \) and so (A.5) becomes
\[ \frac{1}{g} \frac{\partial g}{\partial t} = -\lambda = D \frac{1}{f} \frac{\partial^2 f}{\partial x^2}. \] (A.6)

Recalling that the initial condition has been written as \( h(x) \) it is possible to write the solution as
\[ u(x, t) = h(x)e^{-\lambda t}, \] (A.7)
with the requirement that
\[ -D \frac{\partial^2 f}{\partial x^2} = \lambda f. \] (A.8)

The problem (A.8) is an eigenvalue problem for the differential operator \(-D \partial^2/\partial x^2\) with eigenvalue \( \lambda \) and eigenfunction \( f(x) \). The eigenfunctions and eigenvalues will be determined imposing the boundary conditions.

The general solution to (A.8) can be written as
\[ f(x) = Ae^{-ikx} + Be^{ikx}, \] (A.9)
with \( k := \sqrt{\lambda / D} \), \( A \) and \( B \) are constants to be determined through the boundary conditions. Requiring that \( f(0) = 0 \) it is easily found that \( B = -A \) and thus

\[
    f(x) = A \left( e^{-ikx} - e^{ikx} \right) = -2iA \sin kx .
\]

(A.10)

The second boundary condition \( f(L) = 0 \) allows to find the eigenvalues and the eigenfunctions (and the trivial solution \( f(x) = 0 \) as well). In fact \( \sin (kL) = 0 \) as soon as

\[
    kL = \sqrt{\frac{\lambda}{D}} = m\pi , \quad m = 0, \pm 1, \pm 2, \pm 3, \ldots
\]

(A.11)

so that the eigenvalues and the eigenfunctions are

\[
    \lambda_m = D \left( \frac{m\pi}{L} \right)^2 , \quad f_m(x) = \sin \left( \frac{m\pi}{L} x \right) .
\]

(A.12)

The solution to (A.8) will therefore be a linear superposition of the eigenfunctions \( f_m(x) \),

\[
    u(x,t) = \sum_{m=1}^{\infty} a_m \sin \left( \frac{m\pi}{L} x \right) \exp \left[ -D \left( \frac{m\pi}{L} \right)^2 t \right] .
\]

(A.13)

One last condition is still not satisfied, the initial value condition. And is exactly this condition that allows to find the coefficients \( a_m \) such that

\[
    u(x,0) = \sum_{m=1}^{\infty} a_m \sin \left( \frac{m\pi}{L} x \right) = h(x) .
\]

(A.14)

This is a Fourier series on the interval \([0, L]\) of the initial value \( h(x) \) and its coefficients may easily be evaluated keeping in mind the orthogonality property of the eigenfunctions. It is not difficult to show that

\[
    \int_0^L \sin \left( \frac{m\pi}{L} x \right) \sin \left( \frac{k\pi}{L} x \right) \, dx = \begin{cases} 
        0 & \text{if } k \neq m, k = m = 0 , \\
        L/2 & \text{if } k = m ,
    \end{cases}
\]

(A.15)

which allows to compute the coefficients \( a_m \) as

\[
    a_m = \frac{2}{L} \int_0^L h(x) \sin \left( \frac{m\pi}{L} x \right) \, dx .
\]

(A.16)

With \( h(x) \) as defined in (A.2), the above computation leads to the final solution which therefore is

\[
    u(x,t) = \sum_{m=1}^{\infty} a_m \sin \left( \frac{m\pi}{L} x \right) \exp \left[ -D \left( \frac{m\pi}{L} \right)^2 t \right] , \quad a_m = \frac{8 \sin \left( \frac{m\pi}{2} \right)}{m^2\pi^2} .
\]

(A.17)
A.2 Homogeneous Neumann boundary conditions

Once equation (A.1) has been solved for homogeneous Dirichlet boundary conditions it is straightforward to solve it with homogeneous Neumann boundary conditions. In fact, the same procedure could be carried over to yield the correct solution.

Once again, let the mathematical domain be $x \in [0, L]$ for $t > 0$ and if $q(x, t) := \partial u/\partial x$ the homogeneous Neumann boundary conditions are written as $q(0, t) = q(L, t) = 0$. Since the boundary conditions require the derivative to vanish, the initial condition is chosen so that this condition is satisfied at $t = 0$ as well. The initial condition will then be

$$h(x) := u(x, 0) = 1 + 2 \left(\frac{x}{L}\right)^3 - 3 \left(\frac{x}{L}\right)^2 \ . \ \ \ \ (A.18)$$

Everything that has been said in the previous case up to (A.9) still holds. The boundary conditions now require that

$$f'(x) := \frac{df}{dx} = ik \left(Ae^{ikx} - Be^{-ikx}\right), \ \ \ \ (A.19)$$

vanishes at the boundaries of the domain. From $f'(0) = 0$ follows that $A = B$ while $f'(L) = 0$ leads to the same eigenvalue $\lambda_m = D \left(m\pi/L\right)^2$ as in the previous case. The eigenfunction on the other hand changes since the general solution could be now written as

$$f(x) = A \left(e^{ikx} + e^{-ikx}\right) = 2A \cos(kx) \ \ \ \ (A.20)$$

so that the eigenvalue and the eigenfunction in this case are

$$\lambda_m = D \left(m\pi/L\right)^2, \ \ f_m(x) = \cos\left(\frac{m\pi}{L}x\right). \ \ \ \ (A.21)$$

To satisfy the initial condition it is necessary that

$$u(x, 0) = \sum_{m=0}^{\infty} a_m \cos\left(\frac{m\pi}{L}x\right) = h(x) \ \ \ \ (A.22)$$

where the sum now extends from 0 to $\infty$. This is because the orthogonality property of the eigenfunctions, which still holds and could once again be used to compute the coefficients $a_m$, now reads

$$\int_0^L \cos\left(\frac{m\pi}{L}x\right) \cos\left(\frac{k\pi}{L}x\right) \, dx = \ \ \ \ (A.23)$$

Because of this, the initial condition could be written as

$$h(x) = 1+2 \left(\frac{x}{L}\right)^3 - 3 \left(\frac{x}{L}\right)^2 = \frac{1}{2} \sum_{m=1}^{\infty} a_m \cos\left(\frac{m\pi}{L}x\right), \ \ a_m = 24 \frac{1-\cos\left(m\pi\right)}{m^4 \pi^4}, \ \ \ \ (A.24)$$

so that the complete solution is

$$u(x, t) = \frac{1}{2} + \sum_{m=1}^{\infty} a_m \cos\left(\frac{m\pi}{L}x\right) \exp\left[-D \left(\frac{m\pi}{L}\right)^2 t\right], \ \ a_m = 24 \frac{1-\cos\left(m\pi\right)}{m^4 \pi^4}. \ \ \ \ (A.25)$$
Bibliography


We will concentrate now on elliptic equations of the type
\[ \nabla^2 u = f \]

First of all we need to introduce some important and basic concepts which emerge in the discretization of a continuous problem.

The most obvious discretization of a computational domain is in terms of an orthogonal Cartesian grid.
In this way the function is discretized as
\[ f(x, y) \rightarrow f_{ij} = f(x_{ij}, y_{ij}) \quad i, j \in [1, N] \]

Similarly for derivatives. The mathematical justification comes from the Taylor expansion
\[ u(x + \Delta x, y + \Delta y) = u(x, y) + \Delta x \frac{\partial u}{\partial x} + \frac{(\Delta x)^2}{2!} \frac{\partial^2 u}{\partial x^2} \]
\[ + \Delta y \frac{\partial u}{\partial y} + \frac{(\Delta y)^2}{2!} \frac{\partial^2 u}{\partial y^2} + O(\Delta x^3) \]

Let now \( \Delta y = 0 \) and neglect terms \( O(\Delta x^3) \)
\[ u(x + \Delta x, y) = u(x, y) + \Delta x \frac{\partial u}{\partial x} + O(\Delta x^2) \]
\[ \Rightarrow \quad \frac{u(x + \Delta x, y) - u(x, y)}{\Delta x} = \frac{\partial u}{\partial x} + O(\Delta x) \]
This is referred as "forward" differencing and is first order in $\Delta x$.

\[
\frac{\partial u}{\partial x} = \frac{u(x+\Delta x, y) - u(x) + O(\Delta x)}{\Delta x}
\]

Note this is at $i, j$.

\[
\frac{u_x}{i, j} = \frac{u_{i+1,j} - u_{i,j}}{\Delta x} + O(\Delta x)
\]

Note this is not centred.

We can do the same trick for the "backward differencing".

\[
u(x-\Delta x, y) = u(x, y) - \Delta x \frac{\partial u}{\partial x} + \frac{(\Delta x)^2}{2!} \frac{\partial^2 u}{\partial x^2} + O(\Delta x^3)
\]

Using forward and backward differencing we then obtain (subtracting):.

\[
\frac{\partial u}{\partial x} = \frac{u(x+\Delta x, y) - u(x-\Delta x, y) + O(\Delta x^3)}{2\Delta x}
\]

and this is 2nd-order and is centred.

\[
\frac{2u}{\partial x (i, j)} = \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} + O(\Delta x^2)
\]
Backward and forward differencing could be used at boundaries but penalize the order of the scheme.

Q: What about second-derives?

If we add the forward and backward differencing:

\[
\frac{\partial^2 u}{\partial x^2} \bigg|_{i,j} = \frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{\Delta x^2} + O(\Delta x^2)
\]

Q: What about mixed derivative?

Let \( \Delta x = \Delta y = h \)

\[
\frac{\partial^2 u}{\partial x \partial y} \bigg|_{i,j} = \frac{1}{4h^2} \left[ \frac{U_{i-1,j-1} - U_{i-1,j+1} + U_{i+1,j+1} - U_{i+1,j-1}}{4h^2} \right] + O(h^2)
\]

\[2y(2xU_{i,j}) = \frac{U_{i+1,j+1} - U_{i-1,j+1} - U_{i+1,j-1} + U_{i-1,j-1}}{2h} \]
I can always perform a change of coods $(x,y) \rightarrow (x',y')$ that removes the mixed derivatives.

We can write the Laplacian in Cartesian coods

$$\nabla^2 u = U_{xx} + U_{yy}$$

then

$$U_{xx} = u_{i+1,j} - 2u_{i,j} + u_{i-1,j} ; \quad U_{yy} = u_{i,j+1} - 2u_{i,j} + u_{i,j-1}$$

$$\nabla^2 u \bigg|_{i,j} = \frac{1}{h^2} \left( u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} \right) \quad \text{(4)}$$

Five points stencil

If there is nonlinearity on the RHS one needs to resort necessarily to numerical methods using iterations

$$\nabla^2 u = f(u) \quad : \quad \text{Poisson equation}$$

Here is the nonlinearity

$$f(u) = qx^2 + y^2$$

$$\nabla^2 u = 2a + 2b = 2(a+b)$$
Consider now

$$\nabla^2 u = f$$

which can be written as

$$u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} = h^2 f_{i,j}$$

or, in matrix form as

$$A \cdot u = b$$

Note that $A$ and $b$ are $N \times N$ matrices.

Where

$$A = \begin{pmatrix} -4 & 1 & 0 & 0 & \cdots \\ 1 & -4 & 1 & 0 & \cdots \\ 0 & 1 & -4 & 1 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \ddots \\ \end{pmatrix}$$

is a tridiagonal matrix.

In other words, Poisson equation can be described as a problem involving matrices.
Suppose we have \( N_x \) and \( N_y \) gridpoints in each direction.

Then,

\[
A \text{ is a matrix } (N_x \times N_y) \times (N_x \cdot N_y)
\]

This because I write \( U \) as a 1-D vector having components

\[
U = (U_{11}, U_{12}, U_{13}, \ldots U_{1N_y}, U_{21}, \ldots U_{N_xN_y})
\]

\[
N_x \times N_y
\]

If the problem is linear (e.g., Laplace), then this problem has a simple solution and can be solved as a \( N \times N \) problem with an "exact" solution.
ITERATIVE OR RELAXATION METHODS

The idea is that of splitting the matrix of coefficients and iterate until a solution is found.

We can think of a relaxation method in terms of what we have learnt with parabolic eqs.

In particular, suppose we want to solve the elliptic equation

\[ Lu = f \]  \hspace{1cm} (1)

\[ \text{elliptic operator (eg } \nabla^2 \text{)} \]

We can think that the solution \( u \) of (1) is the asymptotic solution of a diffusion problem

\[ \frac{\partial u}{\partial t} = Lu - f \]  \hspace{1cm} (2)

In other words, starting from an initial \( u^0 \) which is not solution of (1), the system will evolve (i.e. \( \frac{\partial u}{\partial t} \neq 0 \)) up until the asymptotic state is reached (i.e. \( \frac{\partial u}{\partial t} = 0 \)) and of course this state will be described by \( u^* \) which is solution of (1).
In practice this will require that $t \to 0$.

Let's consider a more specific example:

(3) \[ \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f \]

Using an FTCS differencing ($\Delta x = \Delta y = \Delta$)

\[
\frac{u_{i,j}^{n+1} - u_{i,j}^n}{\Delta t} = \frac{u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n - 4u_{i,j}^n}{\Delta^2}
\]

\[
- \Delta t \cdot f_{i,j}
\]

We have seen that the FTCS scheme is stable if $\Delta t / \Delta^2 \leq \frac{1}{2} \text{ in 1D}$

$\Delta t / \Delta^2 \leq \frac{1}{4} \text{ in 2D}$

\[ \Rightarrow \Delta t = \frac{1}{4} \Delta^2 \]

\[
\frac{u_{i,j}^{n+1} - u_{i,j}^n}{\Delta t} = \frac{1}{4} \left( u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n - 4u_{i,j}^n \right) - \frac{\Delta^2}{4} f_{i,j}
\]

\[
= \frac{1}{4} \left[ u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n - \Delta^2 f_{i,j} \right]
\]

This is called the JACOBI METHOD and is based on the idea that

\[ u_{i,j}^{n+1} = \text{(average of } u_{i,j} \text{ among neighbouring points, i.e. no contribution from } u_{i,j}^{n} \text{)} \]
\( h \approx \frac{p_{\text{max}}}{\text{ups}} \); \( n \to 0 \) for \( p_s \to 0 \)
\( n \to \infty \) for \( p_s \to \infty \)

In general,

\[ p_s \to 1 - \frac{\pi}{J^n} \quad \text{so that} \]

\[ p_s \to 1 \quad (\text{i.e. } J \to \infty) \quad \text{for } J \to \infty \]

In other words, the larger the grid, the larger the no of iterations.

\[ h \approx \frac{p \sqrt{2}}{\sqrt{2}} \implies p \approx \frac{2h}{J^2} \]

\[ p \propto N \quad \text{the larger the no of iterations the smaller the truncation error} \]

\[ p \propto J^{-2} \quad \text{the larger the grid, the larger the truncation error} \]

\[ J \to \infty \text{ for } p \to 0 \]
This certainly works, but how rapidly?

One can write Jacobi's method in a matrix form after splitting $A$

$$A = L + D + U$$

$A \cdot x = b$  $\iff$  $\begin{pmatrix} L + D + U \end{pmatrix} \cdot x = b$

$$D \cdot x = b - (L+U) \cdot x$$

Jacobi's method then just says

$$D^{(k+1)} \cdot x = b - (L+U) \cdot x^{(k)}$$

where $k$ is the no of the iteration and thus

$$x^{(k+1)} = D^{-1} \left[ b - (L+U) \cdot x^{(k)} \right]$$

In the example before, $D$ is the identity matrix
\[ n_0 = \frac{p(n)}{\ln p_5} \implies p_s = \left(\frac{1}{10}\right)^{p/n} \]

At each iteration the error is suppressed of an amount \(10^{-p}\).

If \(n\) operations are performed, the total error is suppressed of a factor

\[ p_s \approx 10^{-p/n} \]

Clearly, the smaller \(p\), the faster the error is suppressed.
To establish how rapidly the solution converges, it is useful to define

\[ 0 \leq \rho_s \leq 1 \quad : \quad \text{SPECTRAL RADIUS of the RELAXATION OPERATOR} \]

\[ (\text{the smaller the better!}) \]

\[ \Rightarrow \]

\[ n \approx \frac{p \ln 10}{-\ln \rho_s} \quad : \quad \begin{cases} \text{no of iterations} \\ \text{necessary to reduce} \\ \text{the error by a} \\ \text{quantity } 10^{-p} \end{cases} \]

\[ e = 10^{-p} \]

In general, \( \rho_s \to 1 \) as the grid size is increased so that the larger the grid, the larger the no. of iterations!

For a problem like (3) with Dirichlet bcs and a \( 5 \times 5 \) grid

\[ \rho_s \approx 1 - \frac{\pi^2}{2J^2} \Rightarrow \]

\[ n \approx \frac{p \ln 10}{\ln \left(1 - \frac{\pi^2}{2J^2}\right)} = \frac{2J^2 p \ln 10}{\pi^2} \]

\[ \sim \frac{\pi^2}{2J^2} - \frac{\pi^4}{4J^4} \sim \frac{\pi^2}{2J^2} \]

\[ \sim \frac{1}{2} p J^2 \quad (0) \quad (p=1) \]

i.e., \( n \approx J^2 \) for 5% error; if \( 3 \times 10^{-3} \rightarrow 0 \) \((J=6)\) order.
An alternative approach to the Jacobi method is based on the possibility of doing the Gauss-Seidel method with all the new information that is available.

\[ \begin{align*}
U_{i,j}^{n+1} &= \text{average} \left( U_{i\pm 1,j \pm 1}^{n} \right) \quad \text{JACOBI} \\
U_{i,j}^{n+1} &= \text{average} \left( U_{i-1,j-1}^{n}, U_{i+1,j+1}^{n} \right) \\
(L + D)^{(k+1)} + U^{(k)} &= b \\
X^{(k+1)} &= (L + D)^{-1} \left[ b - U^{(k)} \right] \\
\text{Using the component notation} \\
U_{i,j}^{n+1} &= \frac{1}{4} \left( U_{i+1,j}^{n} + U_{i-1,j}^{n} + U_{i,j+1}^{n} + U_{i,j-1}^{n} \right) \\
-\frac{\Delta^2}{4} f_{i,j} \\
\text{GAUSS-SEIDEL METHOD} \\
\end{align*} \]

Although at a new time level, these quantities have been already calculated in a loop over \( i,j \), this trick would not work if using \((L+D)^{-1}\).
$\text{By}$

\[
\text{UNPO}(1, 1) = \ldots; \text{UNPO}(N, 1) = \ldots;
\]

\[
\text{UNPO}(1, 1) = \ldots; \text{UNPO}(1, N) = \ldots
\]

\[
\text{DO } I = 2, N-1
\]

\[
\text{DO } J = 2, N-1
\]

\[
\text{UNPO}(I, J) = 0.25 \times (\text{UN}(I+1, J) + \text{UNPO}(I-1, J)
\]

\[
+ \text{UN}(I, J+1) + \text{UNPO}(I, J-1)
\]

\[
- (\text{DELTA}^2 / 4.0) \times F(I, J)
\]

\[
\text{END DO}
\]

\[
\text{END DO}
\]

Although **Gauss-Seidel** maximizes the efficiency in averaging, it is not much better than the **Jaddi** method.

\[
(p^*_{c})_{\text{GS}} = (p^*_{c})_{J} \approx 1 - \frac{\pi^2}{J^2} + \frac{\pi^4}{4J^4} \approx 1 - \frac{\pi^2}{J^2}
\]

\[
\eta_{\text{GS}} \approx \frac{p^2 J^2 \ln 10}{\pi^2} \approx \frac{1}{4} p J^2
\]

\[
\Rightarrow \left( \frac{\eta_s}{\eta_{\text{GS}}} \right) \approx 2
\]

A gain of a factor 2 only.
A dramatic improvement in the GAUSS-SEIDEL iteration scheme can be introduced if we "anticipate" the future corrections at the \((k+1)\)-th stage of the iteration. How do we do this?

Gauss-Seidel:

\[
X^{(k+1)} = -(L+D)^{-1} \left[ U X^{(k)} - b \right]
\]

Add and subtract \(X^{(k)}\):

\[
X^{(k+1)} = X^{(k)} - (L+D)^{-1} \left[ (L+D+U) X^{(k)} - b \right]
\]

However,

\[
(L+D+U) X^{(k)} - b = \varepsilon_{(k)} \quad \text{k-th residual:}
\]

\(\varepsilon_{(k)}\) is the error made in the \(k\)-th iteration (we need this or something of this sort).

Then,

\[
X^{(k+1)} = X^{(k)} - (L+D)^{-1} \varepsilon_{(k)}
\]

We can now "anticipate" or "overcorrect":

\[
X^{(k+1)} = X^{(k)} - \omega (L+D)^{-1} \varepsilon_{(k)}
\]

overrelaxation parameter

\textbf{SIMULTANEOUS OVERRELAXATION METHOD (SOR)}
\[
\left( 1 - \frac{\pi^2}{J^2} \right)^2 \approx \left( 1 - \frac{2\pi^2}{J^2} + \frac{\pi^4}{J^4} \right) \left( 1 - \frac{2\pi}{J^2} \right) \\
= \left( 1 - \frac{2\pi^2}{J^2} \right) \left( 1 - \frac{\pi}{J^2} \right) \\
= 1 - \frac{\pi}{J^2} - 2\frac{\pi^2}{J^2} + 2\frac{\pi^3}{J^3} - 1 - \frac{\pi}{J^2} \\
= 1 - \frac{\pi}{J^2} - 2\frac{\pi^2}{J^2} + 2\frac{\pi^3}{J^3} - 1 - \frac{\pi}{J^2}
\]
The conditions over which the SOR works are

I) SOR is convergent only for 0 < ω < 2
   (if 0 < ω < 1 the method is said of under-relaxation)

II) Under rather generic conditions, SOR has a faster convergence than Gauss-Seidel

III) If \( \rho_j \) is the spectral radius for the Jacobi method

\[
\rho_{\text{SOR}} = \left( \frac{\rho_j}{1 + \sqrt{1 - \rho_j^2}} \right)^2 \approx \rho_j^2 \left( 1 - 2 \sqrt{1 - \rho_j^2} \right)
\]

IV) There exists an optimal overrelaxation parameter

\[
\omega = \left( \frac{2}{1 + \sqrt{1 - \rho_j^2}} \right) \quad 1 - \rho_j^2 = \left( 1 - \left( 1 - \frac{\pi^2}{L^2} \right)^2 \right)
\]

\[
\approx 1 - \frac{\pi^2}{4L^2} \approx \frac{\pi^2}{4L^2}
\]

\[
\rho_j \approx 1 - \frac{\pi^2}{2L^2} \implies \rho_{\text{SOR}} \approx \left( \frac{1 - \frac{\pi^2}{J^2}}{1 + \frac{\pi}{J}} \right)^2 \approx 1 - \frac{\pi}{J}
\]

\[
\implies \omega \approx \frac{2}{1 + \frac{\pi}{J}} \approx 1.9391 \quad J=100
\]

\[
\approx 1.9937 \quad J=1000
\]
\( (*) \)
\[
\frac{n_0 - \text{plu}10}{\text{lu psor.}} = \frac{\text{plu}10}{\pi/2} = p_5 \left( \frac{\text{lu}10}{\pi} \right) ^{\frac{3}{4}} p_5
\]
\[
\text{lu psor.} = \ln \left( 1 - \frac{\pi}{2} \right) - \frac{\pi}{2}
\]
\[
= 0.73 \sim \frac{3}{4}
\]

10) This is just the Gauss-Seidel step.
the no of iterations to reduce the error by a factor $10^{-p/n}$ is

$$n_{SOR} \sim \frac{pJ \ln 10}{\pi} \sim \frac{3}{4} \frac{pJ}{\pi}$$

$$\Rightarrow \left\{ \begin{array}{l}
\frac{n_{SOR}}{n_{GS}} \sim \frac{1}{5} \\
10^{-2} - 10^{-3} \text{ typically!}
\end{array} \right.$$  

How do we implement SOR in practice?

Let's go back to (1)

$$Lu = f$$

$$a_{ij} u_{i+1,j} + b_{ij} u_{i-1,j} + c_{ij} u_{i,j+1} + d_{ij} u_{i,j-1} + e_{ij} u_{i,j} = f_{ij}$$

If $L = D$, then the new solution is obtained first calculating

$$u_{ij}^* = \frac{1}{e_{ij}} \left[ f_{ij} - a_{ij} u_{i+1,j} - b_{ij} u_{i-1,j} - c_{ij} u_{i,j+1} - d_{ij} u_{i,j-1} \right]$$

Gauss-Seidel

so that one can write the new $k+1$ estimate as

$$u_{ij}^{(k+1)} = \omega u_{ij}^* + (1-\omega) u_{ij}^{(k)}$$

$$= u_{ij}^{(k)} - \omega (u_{ij}^{(k)} - u_{ij}^*)$$
(4)

Comparing \( U^{*ij} \) and \( E^{ij} \), it is clear that
\( E^{ij} \) measure the difference between \( U^{*ij} \) and \( U^{(k)ij} \), i.e.

\[
\frac{E^{ij}}{e^{ij}} = \frac{U^{(k)ij} - U^{*ij}}{e^{ij}}
\]

(4x)

Note that this is not the mathematically correct definition of the residual, which cannot mix values of \( u \) at different iterations i.e.

\[
\bar{E} = (du - f)^{(k)}
\]
The k-th "residual" can be calculated as \( \eta_{ij}^{(k)} \):

\[
\eta_{ij}^{(k)} = a_{ij} U_{i,j}^{(k)} + b_{ij} U_{i,j}^{(k-1)} + c_{ij} U_{i,j}^{(k+1)} + d_{ij} U_{i,j-1}^{(k)} + e_{ij} U_{i,j+1}^{(k)} - f_{ij}
\]

and given that by def \( \frac{\eta_{ij}^{(k)}}{e_{ij}} = U_{ij}^{(k)} - U_{ij}^{*} \) — the residual must be evaluated using GS.

Note the simplicity of the algorithm where the norm of \( \eta_{ij} \) can be used as criterion for terminating the iteration.

ie one can calculate

\[
\| \eta^{(k)} \|_2 = \left( \sum_{ij} (\eta_{ij}^{(k)})^2 \right)^{1/2}
\]

check that

\[
\| \eta^{(k+1)} \|_2 < \| \eta^{(k)} \|_2
\]

and stop the iteration when

\[
\| \eta^{(k)} \|_2 = \max \{ \eta_{ij}^{(k)} \} < \varepsilon
\]

TOLERANCE
1) SOR does not reach its asymptotic rate of convergence before about $O(\sqrt{N})$ iterations.

This can be avoided by specifying the proper relaxation parameter.

In other words: the optimal relaxation parameter is not necessarily the correct initial one.

To implement the so-called Chebyshev acceleration in SOR it's first necessary to realize that effectively odd and even mesh points are decoupled.

Consider that

$$U_{i,j} = \frac{1}{e_{i,j}} \left( f_{i,j} - a_{i,j} U_{i+1,j} - b_{i,j} U_{i-1,j} - c_{i,j} U_{i,j+1} - d_{i,j} U_{i,j-1} \right)$$

means that

$$U_{i,j} = f(U_{i+1,j+1}) \quad \text{i.e.}$$

the solution on the white mesh points depends only on the black ones and vice versa.
As a result, the algorithm can update only the odd mesh points in half-sweep and then the even ones in the second half sweep.

\[ u_{ij}^{n+1} = u_{ij}^n - \frac{\varepsilon_{ij}}{\eta_{ij}} w^{n+1/2} \quad i,j: \text{odd} \]

\[ u_{ij}^n = u - \frac{\varepsilon_{ij}}{\eta_{ij}} w^n \quad \text{for } i,j: \text{even} \]

In this case, the Chebyshev acceleration consists of specifying the relaxation parameter as follows:

\[ \omega^{(0)} = 1 \]

\[ \omega^{(1/2)} = 1 / (1 - \eta^2 / 2) \]

\[ \vdots \]

\[ \omega^{(n+1/2)} = 1 / (1 - \eta^2 w^n / 4) \quad n = 1/2, 1, \ldots \infty \]

so that \( \omega^{(\infty)} \rightarrow \omega_{\text{optimal}} \)

When this is done, the norm of the error decreases at each iteration.
2) All of this applies also when the matrices are complex

\[ C \bar{z} = w, \text{ where} \]

\[ C = A + iB; \quad z = x + iy; \quad w = u + iv \]

then

\[ C \bar{z} = (A + iB)(x + iy) \]

\[ = Ax - By + i(Bx + Ay) = u + iv \]

\[ \iff \]

\[ \begin{cases} Ax - By = u \\ Bx + Ay = w \end{cases} \]
Another relaxation method with performances that are comparable or better than the SOR is the ADI (Alternating Direction Implicit) method.

Before discussing the ADI in detail it is useful to introduce the more generic class of Operator Splitting methods to which ADI belongs.

Consider the IVP

$$\frac{\partial u}{\partial t} = L u$$  \hspace{1cm} (1)

where $L$ is some operator, not necessarily linear, but that can be written as a linear decomposition of $m$ distinct operators, i.e.

$$L u = L_1 u + L_2 u + \cdots + L_m u$$

(e.g. $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$)

Let $U_1, U_2, \ldots, U_m$ be the updating algorithms for each of the different operators $L_1, L_2, \ldots, L_m$. In this case one can regard each term as independent and apply the $U_i$ as if the corresponding $L_i$ is the ONLY operator on the RHS of (1)
\[ U^{n+1} = U_1(u^n, \Delta x, \Delta t) \]
\[ U^{n+2} = U_2(u^{n+1}, \Delta x, \Delta t) \]
\[ \vdots \]
\[ U^{n+m} = U_m(u^{n+m-1}, \Delta x, \Delta t) \]

In this case, the operator splitting consists of continuously updating \( U \) from \( n \) to \( n+1 \) through fractional steps, each making use of the different updating schemes \( U_1, U_2, \ldots U_m \).

An important aspect to note in this operator splitting method is that each update is over a time step \( \Delta t \) and not a fraction of it!
Alternatively, it is possible to update $u$ from $n$ to $n+1$ using the full operator $L$ but with an update algorithm which is stable at least for the $L_1$ term. In this case

$$u^{n+1/m} = L^{(1)}(u^n, \Delta x, \Delta t/m)$$

$$u^{n+2/m} = L^{(2)}(u^{n+1/m}, \Delta x, \Delta t/m)$$

: 

\[ ... \quad \text{full operator with the update scheme is stable} \]

$$u^{n+1} = L(u, \Delta x, \Delta t/m)$$

Example:

Let's consider the diffusion equation

$$\partial_t u = \nabla^2 u - u$$

so that $\nabla^2 = L = L_x + L_y$

where

$L_x u = u_{i+1,j} - 2u_{i,j} + u_{i-1,j}$

$L_y u = u_{i,j+1} - 2u_{i,j} + u_{i,j-1}$

The time update can then be made as
\[
\begin{align*}
(u_{n+1}) &= \left( u^n - \frac{\Delta t}{2h^2} \left( L_x u_{n+1/2} + L_y u^n \right) \right) - \epsilon \\
(u_{n+1}^{n+1/2}) &= \left( u_{n+1/2} - \frac{\Delta t}{2h^2} \left( L_x u_{n+1/2} + L_y u_{n+1} \right) \right) - \epsilon
\end{align*}
\]

where \( h = \Delta x = \Delta y \)

Written in matrix form this is

\[
\begin{align*}
(L_x + r \mathbf{I}) \cdot u_{n+1/2} &= (r \mathbf{I} - L_y) \cdot u^n - \epsilon h^2 \\
(L_y + r \mathbf{I}) \cdot u_{n+1} &= (r \mathbf{I} - L_x) \cdot u_{n+1/2} - \epsilon h^2
\end{align*}
\]

where \( r = \frac{2h^2}{\Delta t} \)

In practice, the ADI method can be implemented as follows

1) Initialize a vector \( \mathbf{u} \) as

\[
\mathbf{y} = (r \mathbf{I} - L_y) \cdot \mathbf{u}^0
\]
\[(\star)\]

\[(1x + rI)u^{n+1/2} = 4 - h^2e\]

\[1xu^{n+1/2} = 4 - h^2e - ru^{n+1/2} \implies\]

\[(rI - 1x)u^{n+1/2} = ru^{n+1/2} - (4 - h^2e - ru^{n+1/2})\]

\[= -4 + 2ru^{n+1/2} + h^2e\]

\[\implies\]

\[(rI - 1x)u^{n+1/2} - h^2e = -4 + 2ru^{n+1/2}\]
2) Evolve $u^n$ to the following fractional time step

$$(L_x + r I) u^{n+1/2} = y - h^2 e$$

3) Compute

$$\phi = -y + 2r u^{n+1/2} \quad (\phi \text{ can overwrite } y)$$

4) Evolve to the following integer timestep

$$(Ly + r I) u^{n+1} = (r I - Lx) u^{n+1/2} - h^2 e$$

$$\phi$$

5) Check convergence. If not reached, set

$$y = (r I - 2y) u^{n+1}$$

$$= -\phi + 2r u^{n+1}$$

and restart from 2)
ADDENDUM

Let's consider

\[ \lim_{n \to \infty} \{ \phi - \phi_{A_n} \}^N = \]

\[ = \max \{ \phi - \phi_{A_n} \}^N \]

\[ \ell_0 \quad \text{low res} \quad \text{high res} \quad \text{asymptotic level} \]

\[ \log \text{ (no. of iters) } \]

To explain why there is an asymptotic level for \( \ell_0 \) which is not the machine accuracy it is useful to consider the Fourier series expansion of the solution.
Fourier series

\[ f(x) = \lim_{m \to \infty} \sum_{j=-m}^{m} c_j e^{i \left( \frac{\pi j x}{L} \right)} \]

where \( f \) is continuous in \( x \in [-L, L] \)

\( j \to j = j/2 \)

\( j = M \Rightarrow j = m/2 \)

\[ = \lim_{m \to \infty} \sum_{j=-m/2}^{m/2} c_j e^{i \left( \frac{2\pi j x}{L} \right)} \]

Let

\[ \lambda j = \frac{L}{j} \Rightarrow k_j = \frac{2\pi}{\lambda j} = \frac{2\pi}{j} \]

\[ = \lim_{m \to \infty} \sum_{j=-m/2}^{m/2} i k_j x \]

\( k_{m/2} \) is minimum wave number corresponds to a wavelength

\[ \lambda_{m/2} = \frac{2L}{m} \to 0 \text{ when } m \to \infty \]
In practice, when calculating a numerical solution on a grid with resolution $\Delta x = h$ one is building a superposition of solutions with wave numbers between the maximum

$$k_{\text{max}} = \frac{2\pi}{L}, \quad \frac{m}{2} = \frac{\pi m}{L} \quad (\lambda_{\text{min}} = \frac{2L}{m})$$

and the minimum

$$k_{\text{min}} = \frac{2\pi}{L}, \quad \lambda_{\text{min}} = \frac{h}{2L}$$

In other words, selecting $h$ one selects

$$m = \frac{2L}{h}$$

the smaller $h$, the longer $m$ and thus the elements in the series

This explains why machine accuracy is not necessarily reached

Also, this explains why for a given number of iterations, the error will be longer for smaller $h$; there are more wavelets to resolve.
A dramatic improvement in the Gauss-Seidel iteration scheme can be introduced if we "anticipate" the future corrections at the \((k+1)\)-th stage of the iteration. How do we do this?

Gauss-Seidel: (exchanged \(n\) with \(k\))

\[
x^{(k+1)} = -(L+D)^{-1} \left[ Ux^{(k)} - b \right]
\]

Add and subtract \(x^{(k)}\)

\[
x^{(k+1)} = x^{(k)} - (L+D)^{-1} \left[ (L+D+U)x^{(k)} - b \right]
\]

However

\[
(L+D+U)x^{(k)} - b = \xi^{(k)} \quad \text{\(k\)-th residual, i.e. the error made in the \(k\)-th iteration (we need this or something of this sort)}
\]

Then

\[
x^{(k+1)} = x^{(k)} - (L+D)^{-1} \xi^{(k)}
\]

We can now "anticipate" or "overcorrect"

\[
x^{(k+1)} = x^{(k)} - \omega (L+D)^{-1} \xi^{(k)}
\]

\uparrow \text{overrelaxation parameter}

**SIMULTANEOUS OVERRELAXATION METHOD (SOR)**
\[
\left( \frac{1 - \frac{\pi^2}{J^2}}{1 + \frac{\pi}{2J}} \right)^2 = \left( 1 - 2\frac{\pi^2}{J^2} + \frac{\pi^4}{J^4} \right) \left( 1 - \frac{\pi}{J^2} \right) = \left( 1 - 2\frac{\pi^2}{J^2} \right) \left( 1 - \frac{\pi}{J^2} \right) = 1 - \frac{\pi}{J} - 2\frac{\pi^2}{J^2} + 2\frac{\pi^3}{J^3} \sim 1 - \frac{\pi}{J}
\]
The conditions over which the SOR works are:

I) SOR is convergent only for $0 < w < 2$ (if $0 < w < 1$ the method is said of under-relaxation).

II) Under rather generic conditions, SOR has a faster convergence than Gauss-Seidel.

III) If $\rho_j$ is the spectral radius for the Jacobi method, then:

$$
\rho_{\text{SOR}} = \left( \frac{\rho_j}{1 + \sqrt{1 - \rho_j^2}} \right)^2 \approx \rho_j^2 \left( 1 - 2 \sqrt{1 - \rho_j^2} \right)
$$

IV) There exists an optimal overrelaxation parameter $\omega$ such that:

$$
\omega = \left( \frac{2}{1 + \sqrt{1 - \rho_j^2}} \right)^2 = \frac{1 - \rho_j^2}{1 - \rho_j^2 - \frac{\pi^2}{24}}
$$

i.e.

$$
\rho_j \approx 1 - \frac{\pi^2}{2 \delta^2} \Rightarrow \rho_{\text{SOR}} \approx \left( \frac{1 - \pi^2 / 2 \delta^2}{1 + \frac{\pi}{2}} \right)^2 1 - \frac{\pi}{2}
$$

Substituting $\delta = 1, 100, 1000$:

$$
\omega \approx \frac{2}{1 + \pi / 5} \approx \begin{cases} 
1.9391 & J=100 \\
1.9937 & J=1000
\end{cases}
$$
\[ n^* - \frac{\mu n_{10}}{\ln \phi_{sr}} = \frac{\mu n_{10}}{\frac{\pi}{2}} = P \phi \left( \frac{\ln 10}{\pi} \right) \sim \frac{3}{4} P \phi \]

\[ \ln \phi_{sr} = \ln \left( 1 - \frac{\Pi}{J} \right) \sim -\frac{\Pi}{J} = 0.73 \sim \frac{3}{4} \]

(10)

This is just the Gauss-Seidel step where I indicate the LHS with \( \mathbf{u}^* \) rather than \( \mathbf{u}_{i,j}^{**} \).
the no of iterations to reduce the error by a factor $10^{-p/n}$ is

$$n_{opt} \sim \frac{p J}{\ln 10} \sim \frac{3}{4} p J$$

$$\Rightarrow \quad \frac{n_{opt}}{n_{gs}} \sim \frac{1}{5} \sim 10^{-2} - 10^{-3} \text{ typically!}$$

How do we implement SOR in practice?

Let's go back to (1)

$$L u = f \iff a_{ij} u_{i+j} + b_{ij} u_{i-j} + c_{ij} u_{i,j+1} + d_{ij} u_{i,j-1} + e_{ij} u_{i,j} = f_{ij}$$

(If $L = \nabla^2$, $a_{ij} = b_{ij} = c_{ij} = d_{ij} = 1$; $e_{ij} = -4$)

then the new solution is obtained first calculating

$$u_{i,j}^* = \frac{1}{e_{ij}} \left[ f_{ij} - a_{ij} u_{i+j}^k - b_{ij} u_{i-j}^k - c_{ij} u_{i,j+1}^k - d_{ij} u_{i,j-1}^k \right]$$

so that one can write the new $k+1$ estimate as

$$u_{i,j}^{(k+1)} = \omega u_{i,j}^* + (1-\omega) u_{i,j}^{(k)}$$

$$= u_{i,j}^{(k)} - \omega (u_{i,j}^{(k)} - u_{i,j}^*)$$
Comparing $U^{*\,ij}$ and $U^{(k)\,ij}$, it is clear that $E^{(k)\,ij}$ measures the difference between $U^{*\,ij}$ and $U^{(k)\,ij}$, i.e.

$$E^{(k)\,ij} = U^{(k)\,ij} - U^{*\,ij}$$

Note that this is not the mathematically correct definition of the residual, which cannot mix values of $x$ at different locations, i.e.

$$E^{(k)} = (P, f)^{(k)}$$
The k-th "residual" can be calculated as
\( (k) \)
\[ q_{ij}^{(k)} = q_{ij}^{(k)} - b_{ij} U_{i,j+1} + c_{ij} U_{i,j-1} + d_{ij} U_{ij} - c_{ij} \]
and given that by def.
\[ \frac{q_{ij}^{(k)}}{e_{ij}} = \frac{U_{ij}^{*} - U_{ij}^{(k)}}{e_{ij}} \]

\[ U_{i,j}^{(k+1)} = U_{i,j}^{(k)} - w \frac{q_{ij}^{(k)}}{e_{ij}} \]

Note the simplicity of the algorithm where the norm of \( q_{ij} \) can be used as criterion for terminating the iteration.

ie one can calculate
\[ \| q^{(k)} \|_2 = \left( \sum_{ij} \left( q_{ij}^{(k)} \right)^2 \right)^{\frac{1}{2}} \]

check that
\[ \| q^{(k+1)} \|_2 < \| q^{(k)} \|_2 \]

and stop the iteration when
\[ \| q^{(k)} \|_2 = \max_{\delta} \left\{ \frac{\delta^{(k)}}{e_{ij}} \right\} < \varepsilon \]

---

TOLERANCE
1) SOR does not reach its asymptotic rate of convergence before about 0.5 iterations. This can be avoided by specifying the proper relaxation parameter.

In other words: the optimal relaxation parameter is not necessarily the correct initial one.

To implement the so-called Chebyshev acceleration in SOR it's first necessary to realize that effectively odd and even mesh points are decoupled.

Consider that

$$u_{i,j} = \frac{1}{e_{i,j}} \left[ f_{i,j} - a_{i,j} u_{i+1,j} - b_{i,j} u_{i,j} - c_{i,j} u_{i-1,j} + d_{i,j} u_{i,j} \right]$$

means that

$$u_{i,j} = f(u_{i+1,j+1}) \quad \text{i.e.}$$

the solution on the white mesh points depends only on the black ones and vice versa.
As a result, the algorithm can update only the odd mesh points in half-sweep and then the even ones in the second half sweep.

\[ u_{i,j}^{n+1} = u_{i,j}^{n} - \frac{e_{i,j}^{n}}{\xi_{i,j}} w^{n+\frac{1}{2}} \quad i,j: \text{odd} \]

\[ u_{i,j}^{n} = u_{i,j}^{n} - \frac{e_{i,j}^{n}}{\xi_{i,j}} w^{n+1} \quad i,j: \text{even} \]

In this case, the Chebyshev acceleration consists of specifying the relaxation parameter as follows:

\[ w^{(0)} = 1 \]

\[ w^{(\frac{1}{2})} = \frac{1}{1 - \xi_{i,j}^{2}/2} \]

\[ w^{(n+\frac{1}{2})} = \frac{1}{1 - \xi_{i,j}^{2} w^{(n)}}/4 \quad n=\frac{1}{2}, 1, \ldots \infty \]

so that

\[ w^{(\infty)} \to w_{\text{optimal}} \]

When this is done, the norm of the error decreases at each iteration.
2) All of this applies also when the matrices are complex

\[ C \frac{\mathbf{x}}{\mathbf{z}} = w \]

where

\[ C = A + iB; \quad \mathbf{x} = \mathbf{x} + iy; \quad w = u + iv \]

then

\[ \frac{\mathbf{x}}{\mathbf{z}} = (A + iB) \left( x + iy \right) \]

\[ = Ax - By + i (Bx + Ay) = u + iv \]

\[ \iff \]

\[ \begin{cases} Ax - By = u \\ Bx + Ay = w \end{cases} \]
Another relaxation method with performance that are comparable or better than the SOR is the ADI (Alternating Direction Implicit) method.

Before discussing the ADI in detail, it is useful to introduce the more generic class of Operator Splitting methods to which ADI belongs.

Consider the IVP

\[ \frac{\partial u}{\partial t} = L u \]  \hspace{1cm} (1)

where \( L \) is some operator, not necessarily linear, but that can be written as a linear decomposition of \( m \) distinct operators, i.e.

\[ L u = L_1 u + L_2 u + \ldots + L_m u \]

(\( e.g. \), \( \frac{\partial^2 u}{\partial x_1 \partial x_2} = \frac{\partial u}{\partial x_1} + \frac{\partial^2 u}{\partial x_2} \))

Let \( U_1, U_2, \ldots, U_m \) be the updating algorithms for each of the different operators \( L_1, \ldots, L_m \). In this case one can regard each term as independent and apply the \( U_i \) as if the corresponding \( L_i \) is the ONLY operator on the RHS of (1).
\[ U^{n+1} = \mathcal{U}_1 (u^n, \Delta x, \Delta t) \]
\[ U^{n+1} = \mathcal{U}_2 (u^n, \Delta x, \Delta t) \]
\[ \vdots \]
\[ U^{n+1} = \mathcal{U}_m (u^n, \Delta x, \Delta t) \]

In this case, the operator splitting consists of continuously updating \( U \) from \( n \) to \( n+1 \) through fractional steps, each making use of the different updating schemes \( \mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_m \)

\[ U^{n+1/2} = \mathcal{U}_1 (u^n, \Delta x, \Delta t) \]
\[ U^{n+1} = \mathcal{U}_2 (u^{n+1/2}, \Delta x, \Delta t) \]
\[ \vdots \]
\[ U^{n+1} = \mathcal{U}_m (u^{n+(m-1)/2}, \Delta x, \Delta t) \]

An important aspect to note in this operator splitting method is that each update is over a time step \( \Delta t \) and not a fraction of it!
Alternatively, it is possible to update $u$ from $n$ to $n+1$ using the full operator $\mathbf{L}$ but with an update algorithm which is stable at least for the $L^1$ term. In this case:

\[ u^{n+1/m} = \mathbf{L}^{(1)} (u^n, \Delta x, \Delta t/m) \]

\[ u^{n+2/m} = \mathbf{L}^{(2)} (u^{n+1/m}, \Delta x, \Delta t/m) \]

...full operator with the update scheme $u$ stable...

\[ u^{n+1} = \mathbf{L}^{(m)} (u^{n+(m-1)/m}, \Delta x, \Delta t/m) \]

Example:

Let's consider the diffusion equation

\[ \partial_t u = \nabla^2 u - f \]

so that \[ \nabla^2 = \mathbf{L} = \mathbf{L}_x + \mathbf{L}_y \]

where

\[ \mathbf{L}_x u = u_{i+1,j} - 2u_{i,j} + u_{i-1,j} \]

\[ \mathbf{L}_y u = u_{i,j+1} - 2u_{i,j} + u_{i,j-1} \]

The time update can then be made as...
\( u^{n+1/2} = u^n - \frac{\Delta t}{2h^2} \left( \frac{\partial_x u^{n+1/2}}{2} + \frac{\partial_y u^n}{2} \right) - \phi \)

\( u^{n+1} = u^{n+1/2} - \frac{\Delta t}{2h^2} \left( \frac{\partial_x u^{n+1/2}}{2} + \frac{\partial_y u^{n+1}}{2} \right) - \phi \)

Implict part

Written in matrix form this is

\[
(L_x + rI)\cdot u^{n+1/2} = (rI - Ly)\cdot u^n - \phi h^2
\]

\[
(L_y + rI)\cdot u^{n+1} = (rI - Lx)\cdot u^{n+1/2} - \phi h^2
\]

where \( r = \frac{2h^2}{\Delta t} \)

In practice, the ADI method can be implemented as follows

1) Initialize a vector \( \bar{u} \) as

\[
\bar{u} = (rI - Ly)\cdot u^0
\]
\[(l_x + r \Pi) u^{n+1/2} = \nabla \cdot \mathbf{v} - h^2\epsilon\]

\[l_x u^{n+1/2} = \nabla \cdot \mathbf{v} - h^2\epsilon - r u^{n+1/2} \Rightarrow\]

\[(r \Pi - l_x) u^{n+1/2} = r u^{n+1/2} - (\nabla \cdot \mathbf{v} - h^2\epsilon - r u^{n+1/2})\]

\[= -\nabla \cdot \mathbf{v} + 2r u^{n+1/2} + h^2\epsilon\]

\[\Rightarrow\]

\[(r \Pi - l_x) u^{n+1/2} - h^2\epsilon = -\nabla \cdot \mathbf{v} + 2r u^{n+1/2}\]
2) Evolve $u^0$ to the following fractional timestep

$$(Lx + rI)u^{n+1/2} = y - h^2e$$

3) Compute

$$\phi = -y + 2r \cdot u^{n+1/2} \quad (+ \text{ can overwrite } y)$$

4) Evolve to the following integer timestep

$$(Ly + rI)u^{n+1} = (rI - Lx)u^{n+1/2} - h^2e$$

$$= \phi$$

5) Check convergence. If not reached, set

$$y = (rI - Ly)u^{n+1}$$

$$= -\phi + 2r \cdot u^{n+1}$$

and restart from 2)
Let's consider \[ L_0 \| \phi - \phi_A \|_V = \max \{ \phi - \phi_A \}^N \]

To explain why there is an asymptotic level for \( L_0 \) which is not the machine accuracy, it is useful to consider the Fourier series expansion of the solution.
Fourier series

\[ f(x) = \lim_{m \to \infty} \sum_{j=-m}^{m} c_j e^{i \frac{\pi j}{L} x} \]

where \( f \) is continuous in \( x \in [-L, L] \)

\[ j \to \hat{j} = j/2 \]

\[ j = m \to \hat{j} = m/2 \]

\[ = \lim_{m \to \infty} \sum_{j=-m/2}^{m/2} c_j e^{i \frac{2\pi j}{L} x} \]

Let \( \lambda_j = \frac{L}{j} \Rightarrow k_j = \frac{2\pi}{\lambda_j} = \frac{2\pi}{L} j \)

\[ = \lim_{m \to \infty} \sum_{j=-m/2}^{m/2} c_j e^{i k_j x} \]

\( k_{m/2} : \text{minimum wave number corresponds to a wavelength} \]

\[ \lambda_{m/2} = \frac{L}{m} \to 0 \text{ when } m \to \infty \]
In practice, when calculating a numerical solution on a grid with resolution $\Delta x = h$, one is building a superposition of solutions with wave numbers between the maximum

$$k_{\text{max}} = \frac{2\pi}{L} \cdot \frac{m}{2} = \frac{\pi m}{L} \quad (\lambda_{\text{min}} = \frac{2L}{m})$$

and the minimum

$$k_{\text{min}} = \frac{2\pi}{L} \cdot 0 = 0 \quad (\lambda_{\text{max}} = \infty)$$

In other words, selecting $h$ one selects

$$m = \frac{2L}{h}$$

the smaller $h$, the larger $m$ and thus the elements in the series.

This explains why machine accuracy is not necessarily reached.

Also, this explains why for a given number of iterations, the error will be longer for smaller $h$: there are more wavelets to resolve.
FOURIER TRANSFORM METHODS

A large class of computational problems fall under this general group. In what follows we will briefly review the mathematical framework and introduce the concept of discrete Fourier Transform.

Physical processes that have a time dependence can be studied in the "time domain" or else in the "frequency domain."

\[ h = h(t) \quad t \in (-\infty, \infty) \]
\[ H = H(f) \quad f \in (-\infty, \infty) \quad \text{with} \quad f = \frac{1}{t} \]

H and h can be considered as representations of the same function with the Fourier transform being the operation that allows to go from one to the other.

\[ H(f) = \int_{-\infty}^{\infty} h(t) e^{-2\pi if t} \, dt \quad \text{FOURIER TRANSFORM} \]
\[ h(t) = \int_{-\infty}^{\infty} H(f) e^{2\pi if t} \, df \quad \text{INVERSE FOURIER TRANSFORM} \]

Note that t needs not be a time coordinate.
but can be a length in which case $f$ will be a wave number.

**FTs are linear operations:**

$$f = g + h \implies F = G + H$$

$$f = \alpha g \implies F = \alpha G$$

Also, the properties of the original functions are reflected in their FTs.

- purely real: $h(t) \in \mathbb{R} \iff H(-f) = H^*(f)$
- purely imaginary: $h(t) \in \mathbb{C} \iff H(-f) = -H^*(f)$
- time symmetric: $h(t) = h(-t) \iff H(-f) = H(f)$
- time asym.: $h(t) = -h(-t) \iff H(-f) = -H(f)$

Other important properties of the FT are

A) "time scaling": $h(at) \iff H(f/a) / |a|$

B) "frequency scaling": $h(t/b) \iff H(bf) / |b|$

C) $\omega = 2\pi f \implies H(f) = H(\omega)$ with $f = \omega / 2\pi$.
So that:

\[ H(\omega) = \int_{-\infty}^{\infty} h(t) e^{-i\omega t} \, dt \]

\[ h(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} H(\omega) e^{-i\omega t} \, d\omega \]

D) **Time shifting**

\[ h(t - t_0) \leftrightarrow H(f) e^{2\pi i f t_0} \]

E) **Frequency shifting**

\[ h(t) e^{-2\pi i f_0 t} = H(f - f_0) \]
So far we have considered the application of FT to a simple function but an important result holds also for the FTs of two functions.

Let 
\[ g = g(t), \ h = h(t) \iff G(f), \ H(f) \]

and define

\[ g \ast h \equiv \int_{-\infty}^{\infty} g(t')h(t-t')dt' = \text{CONVOLUTION OF } g \text{ AND } h \]

\[ = h \ast g \]

then the FT transform associated to the convolution is just the product of the FTs of \( g \) and \( h \)

\[ g \ast h \iff G(f)H(f) : \text{CONVOLUTION THEOREM} \]

This property will be useful if we want to operate on the FT of a given function, for instance, applying a filter in freq (e.g. \( G_f(t) \)) to \( H(f) \).

Similarly, we can define the correlation between two functions as a function of \( t \)

\[ \text{Corr}(g,h) = \int_{-\infty}^{\infty} s.g(t+t')h(t')dt' \]

Then
\[ \text{corr}(g, h) \leftrightarrow G(f) H(-f) = G(f) H^*(f) \]
\[ \text{corr}(g, g) \leftrightarrow |G(f)|^2 \quad \text{AUTO CORRELATION} \]

An important quantity is the
\[ \text{TOTAL POWER} = \int_{-\infty}^{\infty} |h(t)|^2 dt = \int_{-\infty}^{\infty} |H(f)|^2 df \quad \text{PARSEVAL'S THEOREM} \]

\[ \Rightarrow \text{invariant under domain transf.} \quad \text{(same quantity in all domains)} \]

Similarly, one can define the PSD
\[ \text{POWER SPECTRAL DENSITY} = P_h(f) = |H(f)|^2 + |H(-f)|^2 \]
\[ \text{(ONE SIDED)} \]

\[ \Rightarrow \text{TOTAL POWER} = \int_{0}^{\infty} P_h(f) df = \int_{0}^{\infty} |H(f)|^2 df + \int_{0}^{\infty} |H(-f)|^2 df \]

If \( h \) is real, \( H(-f) = H^*(f) \Rightarrow |H(-f)|^2 = |H^*(f)|^2 = |H(f)|^2 \)
\[ P_h(f) = 2 |H(f)|^2 \]

\[ \uparrow \text{if this factor is missing one is really referring to the 2-sided power spectral densities} \]
The different normalizations can lead to two different PSDs according to whether 1- or 2-sided.

The two areas are the same

Sym. if $h(t)$ is real

If $h$ is nonzero everywhere $\Rightarrow$ PSD $\rightarrow \infty$

In these cases one is interested in the PSD per unit time:

$h(t) \rightarrow g(t) = \int_{t}^{t} h(t) \text{ in } t$

$0 \text{ elsewhere}$
PSD per unit time \( \frac{\text{PSD}}{\text{unit time}} = \frac{1}{2} \int_0^{2\pi} |h(t)|^2 dt = G(f) \)

Then, Parseval's theorem states that

\[
\int_{-\infty}^{\infty} G(f) df = \langle h(t)^2 \rangle
\]

mean square of the signal

\[
\langle h(t)^2 \rangle = \frac{1}{2} \int_{-\infty}^{\infty} |h(t)|^2 dt = \frac{1}{2} \int_{-\infty}^{\infty} h^2(t) dt = \langle h^2(t) \rangle
\]

Note that as \( T \) increases, we cover longer and longer portions of \( h(t) \) and the PSD per unit time converges to a finite value for all the frequencies with the exception of those at which \( h(t) \) has a discrete \( \sin-wave \) component (the average)

---

![Diagram of PSD](image)
by sampling at half a period I cannot distinguish between a component A and one B at twice the freq of A.

In other words, fc is the limit of my knowledge; I don't know anything with \( f > fc \).
So far we have considered continuous functions, however in computational physics one can only deal with discrete functions evaluated at specific points in time.

\[ h(t) \rightarrow h_n = h(n\Delta) \]

\( n = 0, 1, 2, \ldots, N-1 \); \( \Delta \): time interval between two samples

As soon as we define a sampling rate we also define a critical or Nyquist frequency

\[ f_c = \frac{1}{2\Delta} = \frac{1}{2} \text{ (sampling frequency)} \]

If \( h(t) = \sin (2\pi ft) = \sin \left( \frac{2\pi t}{\Delta} \right) \)

then the sampling points are every half period \( \Delta \)

\( (\ast) \)
The Nyquist frequency is important for two distinct reasons

1) If \( h(t) \) is continuous and is "bandwidth limited" to frequencies smaller than the critical one, i.e.

\[
H(f) = \begin{cases} 
0 & (f > f_c) \\
\neq 0 & \text{elsewhere}
\end{cases}
\]

then

\[
h(t) = \Delta \sum_{n=-\infty}^{\infty} \frac{\sin(2\pi f_c (t - n\Delta))}{\pi (t - n\Delta)} \delta_n
\]

In other words, all the information can be recovered if the sampling is at a frequency larger than the maximum of the PSD.

E.g. An amplifier has a known freq. response \( f_c \in [f_{\text{min}}, f_{\text{max}}] \)

Then there is no loss of signal if the sampling is made at a rate

\[
\Delta \leq \frac{1}{2f_{\text{max}}}
\]
2) If the function is not bandwidth limited below the Nyquist frequency, then the aliasing phenomenon is unavoidable.

\[ H(f) = \begin{cases} 
\approx 0 & f > f_c \\
\approx 0 & \text{elsewhere} 
\end{cases} \]

In this case all of the PSD that lies outside \(-f_c < f < f_c\) is moved into that range. The reason for this is that it's not possible to distinguish two harmonic components that differ by a multiple of \(\Delta\).

\[ h \quad \text{continuous function} \]

\[ H(f) \]

\[ \text{aliased PSD} \]

\[ \Delta > \frac{1}{2f_{\text{max}}} \]

\[ f_0 = \frac{1}{2\Delta} \]

\[ \text{spurious PSD} \]
to be compared with
Solutions to aliasing problem

a) know the "natural" bandwidth limits of the signal and sample at frequencies higher than those: if $f_{max} > \frac{1}{2 \Delta f_{max}}$

b) filter the PSD with filter having max freq below the critical one; some information is lost but what is left is correct (*)

Q: How do we know there is an aliasing problem?

A: Produce a PSD and check its value at the vicinity of the critical frequency. If the PSD is approaching zero the aliasing problem is not significant

DISCRETE FOURIER TRANSFORMS

Consider a function $h(t)$ sampled at $N$ discrete points

$$h_k = h(t_k) = h(k \Delta) \quad k = 0, 1, 2 \ldots N-1$$

where the $h_k$ can either contain all of the function values that are non-zero or just a portion of them. Hereafter we assume $N$ is even
\[ f_n = \frac{1}{C_n} = \frac{n}{C} = \frac{n}{\frac{n}{2}} \]

\[ f_N = \frac{N}{2} \frac{1}{N\Delta} = \frac{N}{2N\Delta} = \frac{1}{2\Delta} = f_c \]

This is essentially the extended closed trapezoidal rule where the edges do not have coefficients \( \frac{1}{2} \) but rather 1.

This explains why increasing the sampling rate does not help. The only way to improve the \( H_n \) is to increase the integration time, i.e., \( N \).
While it's possible to evaluate $H(f)$ at $f = \{f_0, f_1, f_2\}$, it is convenient to do so at the discrete freqs:

$$f_n = \frac{n}{N \Delta} = 2n \frac{f_c}{N}, \quad n = -\frac{N}{2}, \ldots, \frac{N}{2}$$

$$f_{-\frac{N}{2}} = -f_c \quad \text{and} \quad f_{\frac{N}{2}} = f_c \quad (\$)$$

Note that these are $N+1$ points in the freq. domain but this is fine because of the properties of the PSD: $H(f_c) = H(-f_c)$: symmetric at boundaries.

Then

$$H(f_n) = \int_{-\infty}^{\infty} h(t) e^{-j2\pi f_n t} \ dt \propto \sum_{k} h_k e^{j2\pi k \Delta t}$$

$$= \Delta \sum_{k} h_k e^{j2\pi k \frac{t}{N\Delta}} \Delta$$

$$= \Delta \sum_{k} h_k e^{j2\pi \frac{k}{N} \Delta t}$$

We can then define the DISCRETE FT (DFT)

$$H_n = \sum_{k} h_k e^{j2\pi \frac{k}{N} \Delta t}$$

and this quantity is independent of the sampling rate $(\$)$.
The DFT is such that given

\[ N \text{ complex (real) nos.: } (h_k) \rightarrow N \text{ complex (real) nos.: } H_n \]

Then \( H(fn) \approx \Delta H_n \)

Notes

1) \( n = -N/2 \ldots \frac{N}{2} \) but from \( H_n = \sum_{k=0}^{N-1} h_k e^{-2\pi i nk/N} \)

it's clear that the \( H_n \) are periodic with period \( N \) \( \Rightarrow \)

\( H_{-n} = H_{N-n} \) and therefore we can take \( n = 0, \ldots, \frac{N}{2}, \frac{N}{2} + 1, \ldots, N-1 \)

so that

\[
\begin{align*}
  n &= 0, & f &= 0 \\
  n &= 1, \ldots, \frac{N}{2} - 1, & 0 &< f < f_c \\
  n &= \frac{N}{2}, & f &= f_c, \quad f = -f_c \\
  n &= \frac{N}{2} + 1, \ldots, N-1, & -f_c &< f < 0 \\
  n &= N, & f &= -f_c
\end{align*}
\]
Notes continued

2) The DFT has the same properties of the continuous FT

3) The inverse DFT is readily obtained

\[ h_k(t) = \frac{1}{N} \sum_{n=0}^{N-1} H_n e^{-2\pi i nk/N} \]

4) The discrete form of Parseval’s theorem

\[ \sum_{k=0}^{N-1} |h_k(t)|^2 = \frac{1}{N} \sum_{n=0}^{N-1} |H_n|^2 \]

Q: How computationally expensive is to compute a DFT?

\[ H_n = \sum_{k=0}^{N-1} h_k e^{-2\pi i nk/N} \]

I have to repeat this for all frequencies

\[ = \sum_{k=0}^{N-1} W^{nk} h_k \]

where \( W = e^{-2\pi i/N} \) = const. coefficient

\( W^{nk} \) : power \( h \times k \) of \( W \)
We can consider (*) as a matrix operation in which the product of the matrix $N \times N$ with the vector of $h_k$ gives the $N$ vector components $H_n$. This calculation requires $O(N^2)$ operations.

However this is incorrect and a Fast Fourier Transform (FFT) can take as few as $N \log_2 N$ operations.

If $N = 10^3$

Dumb FT $\sim 10^6$ operations / Huge

FFT $\sim 10^3$ / / Difference!

There are many ways of doing an FFT but a logically simple one is the Danielson–Lanczos algorithm which splits the sampling points into even and odd:

$$H_n = \sum_{k=0}^{N/2-1} \frac{2\pi i k n}{N} h_k e^{2\pi i k (2n)/N} + \sum_{k=0}^{N/2-1} \frac{2\pi i k n}{N} h_k e^{2\pi i k (2n+1)/N}$$

$$= \sum_{k=0}^{N/2-1} h_{2k} e^{2\pi i k (2n)/N} + \sum_{k=0}^{N/2-1} h_{2k+1} e^{2\pi i k (2n+1)/N}$$

$$= H_n^e + \Delta_k H_n^o$$

even \quad \uparrow \quad \text{odd}
One can then iterate our split

$H^e_n$ into $H^e_n$ and $H^o_n$ and continue

until

$H^e_n \rightarrow H^{e_0 \ldots e_0}_n = h_k$

for some $k$.

Determining which $k$ corresponds to the sequence of $e$'s and $o$'s can be done easily if we reverse the pattern of $e$'s and $o$'s and then set $e = \varnothing$, $o = 1$. This pattern of zeros and ones is the value of $k$. 
It's important that \( N \) is an integer power of 2.