

Appendix C: Recapitulation of Numerical schemes

(August 31, 2009) **SUMMARY:** Certain numerical schemes of general use are regrouped here in order to facilitate implementations of simple models.

C.1 The tridiagonal system solver

As a special case of the general **LU** decomposition (*e.g.*, Riley *et al.* 1997), an efficient tridiagonal system solver, based on the so-called *Thomas algorithm*, can be constructed. We begin by assuming that there exists a decomposition for which the lower (**L**) and upper (**U**) matrices possess the same bandwidth of 2:

$$\begin{pmatrix} a_1 & c_1 & 0 & 0 & \cdots & 0 \\ b_2 & a_2 & c_2 & 0 & \cdots & 0 \\ 0 & b_3 & a_3 & c_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & b_{m-1} & a_{m-1} & c_{m-1} \\ 0 & 0 & \cdots & 0 & b_m & a_m \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ \beta_2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & \beta_3 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \beta_{m-1} & 1 & 0 \\ 0 & 0 & \cdots & 0 & \beta_m & 1 \end{pmatrix} \begin{pmatrix} \alpha_1 & \gamma_1 & 0 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \gamma_2 & 0 & \cdots & 0 \\ 0 & 0 & \alpha_3 & \gamma_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \alpha_{m-1} & \gamma_{m-1} \\ 0 & 0 & \cdots & 0 & 0 & \alpha_m \end{pmatrix} \quad (\text{C.1})$$

where the first matrix is the original tridiagonal matrix to be decomposed (*i.e.*, elements a_1 *etc.* are known), the second matrix is **L** with one line of non-zero elements below the diagonal, and the last matrix is **U** with one line of non-zero elements above the diagonal.

Performing the product of matrices, we identify elements $(k, k - 1)$, (k, k) and $(k, k + 1)$ of the product as

$$b_k = \beta_k \alpha_{k-1} \quad (\text{C.2})$$

$$a_k = \beta_k \gamma_{k-1} + \alpha_k \quad (\text{C.3})$$

$$c_k = \gamma_k. \quad (\text{C.4})$$

These relations can be solved for the components of \mathbf{L} and \mathbf{U} by observing that $\gamma_k = c_k$ for all k . Since the first row demands $a_1 = \alpha_1$, subsequent rows provide α_k and β_k recursively from

$$\beta_k = \frac{b_k}{\alpha_{k-1}}, \quad \alpha_k = a_k - \beta_k c_{k-1}, \quad k = 2, \dots, m \quad (\text{C.5})$$

provided that no α_k is zero (otherwise the matrix is singular and cannot be decomposed). Note that there is no β_1 .

The tridiagonal matrix \mathbf{A} has now been decomposed as the product of lower and upper triangular matrices. The solution of $\mathbf{Ax} = \mathbf{LUx} = \mathbf{f}$ is then obtained by first solving $\mathbf{Ly} = \mathbf{f}$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ \beta_2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & \beta_3 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \beta_{m-1} & 1 & 0 \\ 0 & 0 & \cdots & 0 & \beta_m & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{m-1} \\ y_m \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{m-1} \\ f_m \end{pmatrix} \quad (\text{C.6})$$

by proceeding from the first row downward and then solving $\mathbf{Ux} = \mathbf{y}$

$$\begin{pmatrix} \alpha_1 & \gamma_1 & 0 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \gamma_2 & 0 & \cdots & 0 \\ 0 & 0 & \alpha_3 & \gamma_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \alpha_{m-1} & \gamma_{m-1} \\ 0 & 0 & \cdots & 0 & 0 & \alpha_m \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{m-1} \\ x_m \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{m-1} \\ y_m \end{pmatrix}, \quad (\text{C.7})$$

by proceeding from the bottom row upward. The solution is

$$y_1 = f_1, \quad y_k = f_k - \beta_k y_{k-1}, \quad k = 2, \dots, m \quad (\text{C.8})$$

$$x_m = \frac{y_m}{\alpha_m}, \quad x_k = \frac{y_k - \gamma_k x_{k+1}}{\alpha_k}, \quad k = m-1, \dots, 1. \quad (\text{C.9})$$

In practice, the α values can be stored in vector \mathbf{a} initially holding the values a_k , since once α_k is known, a_k is not needed anymore. Similarly β values can be stored in vector \mathbf{b} initially holding b_k and γ values in a vector \mathbf{c} . Also the y and f values can share the same vector \mathbf{f} as the f_k value is no longer needed once the y_k value has been computed. With the additional vector \mathbf{x} for the solution, only 5 vectors are required, and the solution is obtained with only three loops over m points. This demands approximately $5m$ floating-point operations instead of the m^3 operations that a brutal matrix inversion would have required. The algorithm is implemented in MATLAB™ file `thomas.m`.

C.2 1D finite-difference schemes of various orders

Table C.1 STANDARD FINITE-DIFFERENCE OPERATORS FOR UNIFORM GRIDS

FORWARD DIFFERENCE $\mathcal{O}(\Delta t^2)$						
	u^n	u^{n+1}	u^{n+2}	u^{n+3}	u^{n+4}	u^{n+5}
$2 \Delta t \frac{\partial u}{\partial t}$	-3	4	-1			
$\Delta t^2 \frac{\partial^2 u}{\partial t^2}$	2	-5	4	-1		
$2 \Delta t^3 \frac{\partial^3 u}{\partial t^3}$	-5	18	-24	14	-3	
$\Delta t^4 \frac{\partial^4 u}{\partial t^4}$	3	-14	26	-24	11	-2

FORWARD DIFFERENCE $\mathcal{O}(\Delta t)$						
	u^n	u^{n+1}	u^{n+2}	u^{n+3}	u^{n+4}	
$\Delta t \frac{\partial u}{\partial t}$	-1	1				
$\Delta t^2 \frac{\partial^2 u}{\partial t^2}$	1	-2	1			
$\Delta t^3 \frac{\partial^3 u}{\partial t^3}$	-1	3	-3	1		
$\Delta t^4 \frac{\partial^4 u}{\partial t^4}$	1	-4	6	-4	1	

BACKWARD DIFFERENCE $\mathcal{O}(\Delta t)$						
	u^{n-4}	u^{n-3}	u^{n-2}	u^{n-1}	u^n	
$\Delta t \frac{\partial u}{\partial t}$				-1	1	
$\Delta t^2 \frac{\partial^2 u}{\partial t^2}$			1	-2	1	
$\Delta t^3 \frac{\partial^3 u}{\partial t^3}$		-1	3	-3	1	
$\Delta t^4 \frac{\partial^4 u}{\partial t^4}$	1	-4	6	-4	1	

BACKWARD DIFFERENCE $\mathcal{O}(\Delta t^2)$						
	u^{n-5}	u^{n-4}	u^{n-3}	u^{n-2}	u^{n-1}	u^n
$2 \Delta t \frac{\partial u}{\partial t}$				1	-4	3
$\Delta t^2 \frac{\partial^2 u}{\partial t^2}$			-1	4	-5	2
$2 \Delta t^3 \frac{\partial^3 u}{\partial t^3}$		3	-14	24	-18	5
$\Delta t^4 \frac{\partial^4 u}{\partial t^4}$	-2	11	-24	26	-14	3

CENTRAL DIFFERENCE $\mathcal{O}(\Delta t^2)$					
	u^{n-2}	u^{n-1}	u^n	u^{n+1}	u^{n+2}
$2 \Delta t \frac{\partial u}{\partial t}$		-1	0	1	
$\Delta t^2 \frac{\partial^2 u}{\partial t^2}$		1	-2	1	
$2 \Delta t^3 \frac{\partial^3 u}{\partial t^3}$	-1	2	0	2	1
$\Delta t^4 \frac{\partial^4 u}{\partial t^4}$	1	-4	6	-4	1

CENTRAL DIFFERENCE $\mathcal{O}(\Delta t^4)$							
	u^{n-3}	u^{n-2}	u^{n-1}	u^n	u^{n+1}	u^{n+2}	u^{n+3}
$12 \Delta t \frac{\partial u}{\partial t}$		1	-8	0	8	-1	
$12 \Delta t^2 \frac{\partial^2 u}{\partial t^2}$		-1	16	-30	16	-1	
$8 \Delta t^3 \frac{\partial^3 u}{\partial t^3}$	1	-8	13	0	-13	8	-1
$6 \Delta t^4 \frac{\partial^4 u}{\partial t^4}$	-1	12	-39	56	-39	12	-1

C.3 Time-stepping algorithms

Table C.2 STANDARD TIME-STEPPING METHODS FOR $du/dt = Q(t, u)$

EULER METHODS		
	Scheme	Order
Explicit	$\bar{u}^{n+1} = \bar{u}^n + \Delta t Q^n$	Δt
Implicit	$\bar{u}^{n+1} = \bar{u}^n + \Delta t Q^{n+1}$	Δt
Trapezoidal	$\bar{u}^{n+1} = \bar{u}^n + \frac{\Delta t}{2} (Q^n + Q^{n+1})$	Δt^2
General	$\bar{u}^{n+1} = \bar{u}^n + \Delta t ((1 - \alpha)Q^n + \alpha Q^{n+1})$	Δt

MULTI-STAGE METHODS		
	Scheme	Order
Runge-Kutta	$\bar{u}^{n+1/2} = \bar{u}^n + \frac{\Delta t}{2} Q(t^n, \bar{u}^n)$ $\bar{u}^{n+1} = \bar{u}^n + \Delta t Q(t^{n+1/2}, \bar{u}^{n+1/2})$	Δt^2
Runge-Kutta	$\bar{u}_a^{n+1/2} = \bar{u}^n + \frac{\Delta t}{2} Q(t^n, \bar{u}^n)$ $\bar{u}_b^{n+1/2} = \bar{u}^n + \frac{\Delta t}{2} Q(t^{n+1/2}, \bar{u}_a^{n+1/2})$ $\bar{u}^* = \bar{u}^n + \Delta t Q(t^{n+1/2}, \bar{u}_b^{n+1/2})$ $\bar{u}^{n+1} = \bar{u}^n + \Delta t \left(\frac{1}{6} Q(t^n, \bar{u}^n) + \frac{2}{6} Q(t^{n+1/2}, \bar{u}_a^{n+1/2}) + \frac{2}{6} Q(t^{n+1/2}, \bar{u}_b^{n+1/2}) + \frac{1}{6} Q(t^{n+1}, \bar{u}^*) \right)$	Δt^4

MULTI-STEP METHODS		
	Scheme	Truncation order
Leapfrog	$\bar{u}^{n+1} = \bar{u}^{n-1} + 2\Delta t Q^n$	Δt^2
Adams-Bashforth	$\bar{u}^{n+1} = \bar{u}^n + \frac{\Delta t}{2} (-Q^{n-1} + 3Q^n)$	Δt^2
Adams-Moulton	$\bar{u}^{n+1} = \bar{u}^n + \frac{\Delta t}{12} (-Q^{n-1} + 8Q^n + 5Q^{n+1})$	Δt^3
Adams-Bashforth	$\bar{u}^{n+1} = \bar{u}^n + \frac{\Delta t}{12} (5Q^{n-2} - 16Q^{n-1} + 23Q^{n+1})$	Δt^3

PREDICTOR-CORRECTOR METHODS		
	Scheme	Order
Heun	$\bar{u}^* = \bar{u}^n + \Delta t Q(t^n, \bar{u}^n)$ $\bar{u}^{n+1} = \bar{u}^n + \frac{\Delta t}{2} (Q(t^n, \bar{u}^n) + Q(t^{n+1}, \bar{u}^*))$	Δt^2
Leapfrog-Trapezoidal	$\bar{u}^* = \bar{u}^{n-1} + 2\Delta t Q^n$ $\bar{u}^{n+1} = \bar{u}^n + \frac{\Delta t}{2} (Q^n + 5Q(t^{n+1}, \bar{u}^*))$	Δt^2
ABM	$\bar{u}^* = \bar{u}^n + \frac{\Delta t}{2} (-Q^{n-1} + 3Q^n)$ $\bar{u}^{n+1} = \bar{u}^n + \frac{\Delta t}{12} (-Q^{n-1} + 8Q^n + 5Q(t^{n+1}, \bar{u}^*))$	Δt^3

C.4 Partial-derivatives finite differences

On a regular grid $x = i\Delta x + x_0$, $y = j\Delta y + y_0$, the following expressions are of second order

- Jacobian $J(a, b) = \frac{\partial a}{\partial x} \frac{\partial b}{\partial y} - \frac{\partial b}{\partial x} \frac{\partial a}{\partial y}$

$$J_{i,j}^{++} = \frac{(a_{i+1,j} - a_{i-1,j})(b_{i,j+1} - b_{i,j-1}) - (b_{i+1,j} - b_{i-1,j})(a_{i,j+1} - a_{i,j-1})}{4\Delta x \Delta y}$$

$$J_{i,j}^{+\times} = \frac{[a_{i+1,j}(b_{i+1,j+1} - b_{i+1,j-1}) - a_{i-1,j}(b_{i-1,j+1} - b_{i-1,j-1})]}{4\Delta x \Delta y} - \frac{[a_{i,j+1}(b_{i+1,j+1} - b_{i-1,j+1}) - a_{i,j-1}(b_{i+1,j-1} - b_{i-1,j-1})]}{4\Delta x \Delta y}$$

$$J_{i,j}^{\times+} = \frac{[b_{i,j+1}(a_{i+1,j+1} - a_{i-1,j+1}) - b_{i,j-1}(a_{i+1,j-1} - a_{i-1,j-1})]}{4\Delta x \Delta y} - \frac{[b_{i+1,j}(a_{i+1,j+1} - a_{i+1,j-1}) - b_{i-1,j}(a_{i-1,j+1} - a_{i-1,j-1})]}{4\Delta x \Delta y}$$

- Cross derivatives

$$\left. \frac{\partial^2 u}{\partial x \partial y} \right|_{i+1/2, j+1/2} \sim \frac{u_{i+1, j+1} - u_{i+1, j} + u_{i, j} - u_{i, j+1}}{\Delta x \Delta y}$$

$$\left. \frac{\partial^2 u}{\partial x \partial y} \right|_{i, j} \sim \frac{u_{i+1, j+1} - u_{i+1, j-1} + u_{i-1, j-1} - u_{i-1, j+1}}{4\Delta x \Delta y}$$

- Laplacian

$$\left. \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right|_{i, j} \sim \frac{u_{i+1, j} + u_{i-1, j} - 2u_{i, j}}{\Delta x^2} + \frac{u_{i, j+1} + u_{i, j-1} - 2u_{i, j}}{\Delta y^2}$$

C.5 DFT and FFT

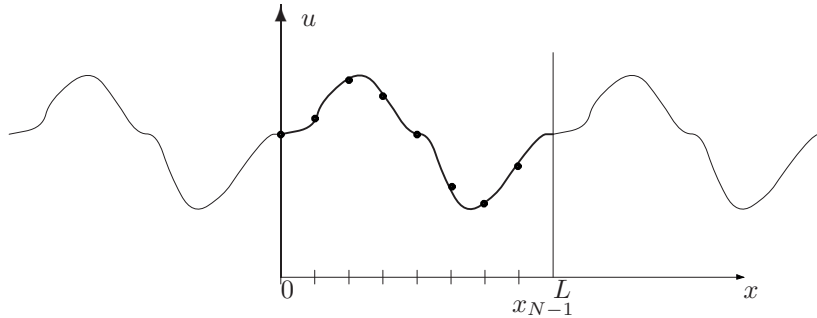


Figure C-1 Periodic signal sampled with N evenly-spaced points x_0 to x_{N-1} . Note that no x_N is needed since by periodicity the value of the function in x_N is the same as in x_0 .

In a periodic domain in which x varies between 0 and L , a complex function $u(x)$ can be expanded in Fourier modes according to

$$u(x) = \sum_{n=-\infty}^{+\infty} a_n e^{i n \frac{2\pi x}{L}} . \quad (\text{C.10})$$

Using orthogonality properties of the Fourier modes⁹

$$\frac{1}{L} \int_0^L e^{i (n-m) \frac{2\pi x}{L}} dx = \delta_{nm} , \quad (\text{C.11})$$

the complex coefficients a_n are readily obtained by multiplying (C.10) by $\exp(-i m 2\pi x/L)$ and integrating

$$a_m = \frac{1}{L} \int_0^L u(x) e^{-i m \frac{2\pi x}{L}} dx . \quad (\text{C.12})$$

Note that we could have used $a_n = b_n/L$ where b_n would then be calculated using the integral without the normalization $1/L$.

Discrete Fourier Transform (DFT) simply truncates¹⁰ the infinite series to a finite series

$$\tilde{u}(x) = \sum_{n=0}^{N-1} a_n e^{i n \frac{2\pi x}{L}} . \quad (\text{C.13})$$

It uses $u(x)$ at a series of sampling points (the discrete values on a numerical grid for example) to determine the coefficients a_n . The points are regularly spaced $x_j = j\Delta x$, $j = 0, \dots, N-1$. Because of periodicity, using x_N would amount to use x_0 and the point is therefore not retained in the sampling. Complex coefficients can be evaluated as

$$\begin{aligned} a_n &= \frac{1}{N} \sum_{j=0}^{N-1} u(x_j) e^{-i n \frac{2\pi x_j}{L}} \\ &= \frac{1}{N} \sum_{j=0}^{N-1} u_j e^{-i n j \frac{2\pi}{N}} \end{aligned} \quad (\text{C.14})$$

where we write $u(x_j) = u_j$ to highlight the fact that we only use information at the discrete points. Obviously this is a discrete version of the integral (C.12). The interesting point is that coefficients a_m are exact in the sense that if we evaluate $\tilde{u}(x_j)$ with those coefficients, we obtain the values at the grid points $u(x_j)$.

The proof of this result is not trivial and we must verify that if we use coefficients (C.14)

⁹ δ_{ij} stands for the Kronecker symbol (0 if $i \neq j$, 1 otherwise)

¹⁰The truncation from 0 to $N-1$ is somehow arbitrary and sometimes the presentation with a series from $-N/2$ to $N/2$ is preferred.

in (C.13) we obtain $\tilde{u}(x_j) = u(x_j)$ at the grid points: We obtain successively

$$\tilde{u}(x_j) = \sum_{n=0}^{N-1} a_n e^{i n \frac{2\pi x_j}{L}} \quad (\text{C.15})$$

$$\begin{aligned} &= \sum_{n=0}^{N-1} \frac{1}{N} \sum_{m=0}^{N-1} u_m e^{-i n m \frac{2\pi}{N}} e^{i n j \frac{2\pi}{N}} \\ &= \frac{1}{N} \sum_{m=0}^{N-1} u_m \left[\sum_{n=0}^{N-1} e^{i (j-m)n \frac{2\pi}{N}} \right] \\ &= \frac{1}{N} \sum_{m=0}^{N-1} u_m \left[\sum_{n=0}^{N-1} \rho^n \right] \end{aligned} \quad (\text{C.16})$$

where

$$\rho = e^{i (j-m) \frac{2\pi}{N}} \quad (\text{C.17})$$

For $j \neq m$, $\rho \neq 1$ and the geometric sum takes the value

$$\sum_{n=0}^{N-1} \rho^n = \frac{1 - \rho^N}{1 - \rho} = 0 \quad (\text{C.18})$$

because $j - m$ is an integer so that $\rho^N = 1$.

When $j = m$ the sum between brackets is simply N and we arrive at $\tilde{u}(x_j) = u(x_j)$, the desired result. Transformation (C.14) is called the direct transform (DFT) and (C.15) the inverse transform (IDFT). As for the Fourier series, scaling can vary and the factor $1/N$ is commonly applied in the inverse transform rather than the direct. Note that the inverse and direct transform are almost identical transformations, except for this scaling and more importantly the sign in the exponential.

Once the coefficients known, (C.13) can be evaluated at any position x in the domain and because the function $\tilde{u}(x)$ takes the values $u(x_j)$ on the discrete points, we now have an interpolation function. It can be used to calculate the value of the function at any desired location using N discrete values. Also (C.13) can be used to evaluate derivatives:

$$\frac{d\tilde{u}}{dx} = \sum_{n=0}^{N-1} i k_n a_n e^{i k_n x}, \quad k_n = \frac{2\pi n}{L}. \quad (\text{C.19})$$

All we have to do to calculate derivatives is to create Fourier coefficients $b_n = i k_n a_n$ from the original coefficient. These are the Fourier coefficient for the derivative, which, by way of an inverse transform, lead to the values of derivatives

$$u(x) \xrightarrow{\text{sampling}} u(x_j) \xrightarrow{\text{DFT}(u_j)} a_n \xrightarrow{\text{derivation}} b_n = i k_n a_n \xrightarrow{\text{IDFT}(b_n)} \left. \frac{d\tilde{u}}{dx} \right|_{x_j}$$

We note that the wavenumber associated with $N - m$ in (C.13) is equivalent to the one associated with m . In reality, the shortest signal resolved by the series corresponds to $n = N/2$ with wavenumber $k = N\pi/L$ or wavelength $2\Delta x$. Hence coefficient a_0 contains the information on the average value, a_1 on the wavelength L and so on up to $a_{N/2}$ which contains the

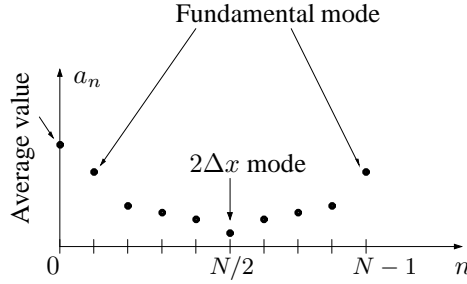


Figure C-2 Coefficients a_n contain the amplitudes of the different modes. With the summation chosen from 0 to $N - 1$, $a_{N/2}$ contains the amplitude of the shortest signal.

complex amplitude of the shortest signal. We see why some presentations of the DFT define the series between $-N/2$ and $N/2$. The information content is however the same.

Up to now, the functions treated were considered complex functions and for a real signal u there is a redundancy in the coefficients a_n . Indeed, with $u_j = u_j^*$, where u^* stands for the complex conjugate, we have

$$\begin{aligned}
 a_{N-m} &= \frac{1}{N} \sum_{j=0}^{N-1} u_j e^{-i(N-m)j\frac{2\pi}{N}} \\
 &= \frac{1}{N} \sum_{j=0}^{N-1} u_j e^{+i mj\frac{2\pi}{N}} \\
 &= a_m^*.
 \end{aligned} \tag{C.20}$$

For real functions u , the DFT contains thus some redundant information and some algorithms exploit this property. Cosine transforms CFT and sine transforms SFT perform in a similar way as DFT using only cosine or sine functions. They are particularly useful if the solution of a problem is known to satisfy homogeneous boundary conditions. For Neuman conditions in $x = 0$ and $x = L$, CFT is the method of choice while for homogeneous Dirichlet conditions, the development of the solution in terms of sine functions allows to satisfy these boundary conditions easily.

The annoying aspect with the transformation presented up to now is that a straightforward implementation as a sum of factors needs a data array of length N , and for each coefficient a_n the calculation of N terms. Therefore, in total, N^2 operations are necessary for each transform. This is prohibitively expensive when used in large-size problems. Fortunately Cooley and Tukey introduced in 1965 a method which is probably among the most celebrated numerical algorithms, reducing the cost of the DFT to $N \log_2 N$ operations. Interestingly enough, the original idea of the method, now called Fast Fourier Transform, goes back to Gauss in 1805 (see collected reprints Gauss, 1866), long before the advent of calculators able to exploit it.

The Fast Fourier Transform is a practical calculation of the discrete Fourier transform and starts with the observation that we can write the transform by separating even and odd terms in j :

$$\begin{aligned}
Na_n &= \sum_{\substack{j=0 \\ j \text{ even}}}^{N-1} u_j e^{-i nj \frac{2\pi}{N}} + \sum_{\substack{j=0 \\ j \text{ odd}}}^{N-1} u_j e^{-i nj \frac{2\pi}{N}} \\
&= \sum_{m=0}^{N/2-1} u_{2m} e^{-i nm \frac{2\pi}{(N/2)}} + e^{-i n \frac{2\pi}{N}} \sum_{m=0}^{N/2-1} u_{2m+1} e^{-i nm \frac{2\pi}{(N/2)}} \quad (\text{C.21})
\end{aligned}$$

and for simplicity we assume N to be a power of two, which is also the case in which the method performs best. Each of the two sums involved is nothing else than a discrete transform working on $N/2$ data points. But those in turn can be broken up into two smaller pieces and so on, a technique called *divide and conquer*. When N is a power of two, this recursive division is possible until all series contain only a single term. Such a single data point can be transformed in a single operation and the recurrence can be rolled up back to retrieve the transform of the original series. To estimate the cost $C(N)$ for a transformation with N data, we see that it is the cost of two transformations of size $N/2$ and the multiplication by $e^{-i n \frac{2\pi}{N}}$ for each of the N coefficients a_n . Hence

$$C(N) = 2C(N/2) + N \quad (\text{C.22})$$

For a single point, one operation is needed so that we can deduce

$$C(N) \sim N \log_2 N. \quad (\text{C.23})$$

There is thus a substantial gain compared to a brute-force approach, which is particularly interesting in spectral methods (Section 18.4) where direct and inverse transforms are intensively used.

Interpolation of a function knowing expansion coefficients could be obtained by a brute-force summation of (C.13) at the desired locations. But we can do better. Suppose we want to interpolate into a regular (finer) grid: $x_k = k/ML$, $k = 0, \dots, M - 1$ having at our disposal measurements from original data on another regular grid $x_j = j/NL$, $j = 0, \dots, N - 1$. The interpolation can be performed very efficiently by simply assuming that shorter signal than those initially capture by the N points have zero amplitude. In other words, from a perspective of the new grid of M points, we take the Fourier coefficients provided from the N points (up to the shortest wavelength resolved by this grid) and set all coefficients corresponding to shorter wavelength to zero. This amounts to add zeros to the array of Fourier coefficients, a procedure called *padding*. With the summing convention taken here (from 0 to $N - 1$), the padding must add zeros in the the middle of the array a_n . Depending on the scaling used, the coefficients also can need a rescaling before transforming back to real space. The interpolation procedure is therefore schematically described by:

$$u(x_j) \xrightarrow{\text{FFT}(u_j)} a_n \xrightarrow{\text{Padding}} b_n = a_n \text{ plus zeros; scaling} \xrightarrow{\text{IFFT}(b_n)} \tilde{u}(x_k)$$

Analytical Problems

- C-1.** Find the truncation errors of the two Adams-Bashforth schemes by a Taylor expansion.
- C-2.** Which of the two second-order approximations of the cross-derivative has a lower truncation error?
- C-3.** Try to establish a finite-difference approximation of a Jacobian at a corner point $i + 1/2, j + 1/2$, using only values from the nearest grid points.

Numerical Exercises

- C-1.** Compare the behavior of the second-order Adams-Bashforths method with the leapfrog-trapezoidal method on the calculation of an inertial oscillation.
- C-2.** Discretize a function $u(x, y) = \sin(2\pi x/L) \cos(2\pi y/L)$ on a regular grid in (x, y) . Calculate the numerical Jacobian between
- \tilde{u} and \tilde{u} ,
 - \tilde{u} and \tilde{u}^2 ,
 - \tilde{u} and \tilde{u}^3 ,
- and interpret your result.
- C-3.** Perform a FFT on $f(x) = \sin(2\pi x/L)$ between $x = 0$ and $x = L$, by sampling with 10, 20 or 40 points. Using the spectral coefficients from the FFT, plot the series expansion using a very fine resolution (200 points) and verify that you recover the initial function. Then repeat with the function $f(x) = x$. What do you observe?
- C-4.** Redo Exercise C-3. using the padding technique instead of a brute-force series evaluation to plot the expansion.