

## Mathematical Tripos Part II: Michaelmas Term 2015

### Numerical Analysis – Lecture 4

**Algorithm 1.19 (The fast Fourier transform (FFT))** We assume that  $n$  is a power of 2, i.e.  $n = 2m = 2^p$ , and for  $\mathbf{y} \in \Pi_{2m}$ , denote by

$$\mathbf{y}^{(E)} = \{y_{2j}\}_{j \in \mathbb{Z}} \quad \text{and} \quad \mathbf{y}^{(O)} = \{y_{2j+1}\}_{j \in \mathbb{Z}}$$

the even and odd portions of  $\mathbf{y}$ , respectively. Note that  $\mathbf{y}^{(E)}, \mathbf{y}^{(O)} \in \Pi_m$ .

Suppose that we already know the inverse DFT of both ‘short’ sequences,

$$\mathbf{x}^{(E)} = \mathcal{F}_m^{-1} \mathbf{y}^{(E)}, \quad \mathbf{x}^{(O)} = \mathcal{F}_m^{-1} \mathbf{y}^{(O)}.$$

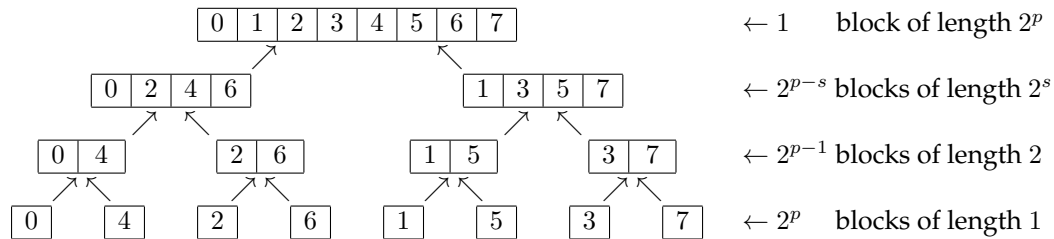
It is then possible to assemble  $\mathbf{x} = \mathcal{F}_{2m}^{-1} \mathbf{y}$  in a small number of operations. Since  $\omega_{2m}^{2m} = 1$ , we obtain  $\omega_{2m}^2 = \omega_m$ , and

$$\begin{aligned} x_\ell &= \sum_{j=0}^{2m-1} \omega_{2m}^{j\ell} y_j = \sum_{j=0}^{m-1} \omega_{2m}^{2j\ell} y_{2j} + \sum_{j=0}^{m-1} \omega_{2m}^{(2j+1)\ell} y_{2j+1} \\ &= \sum_{j=0}^{m-1} \omega_m^{j\ell} y_j^{(E)} + \omega_{2m}^\ell \sum_{j=0}^{m-1} \omega_m^{j\ell} y_j^{(O)} = x_\ell^{(E)} + \omega_{2m}^\ell x_\ell^{(O)}, \quad \ell = 0, \dots, m-1. \end{aligned}$$

Therefore, it costs just  $m$  products to evaluate the first half of  $\mathbf{x}$ , provided that  $\mathbf{x}^{(E)}$  and  $\mathbf{x}^{(O)}$  are known. It actually costs nothing to evaluate the second half, since

$$\omega_m^{j(m+\ell)} = \omega_m^{j\ell}, \quad \omega_{2m}^{m+\ell} = -\omega_{2m}^\ell \quad \Rightarrow \quad x_{m+\ell} = x_\ell^{(E)} - \omega_{2m}^\ell x_\ell^{(O)}, \quad \ell = 0, \dots, m-1.$$

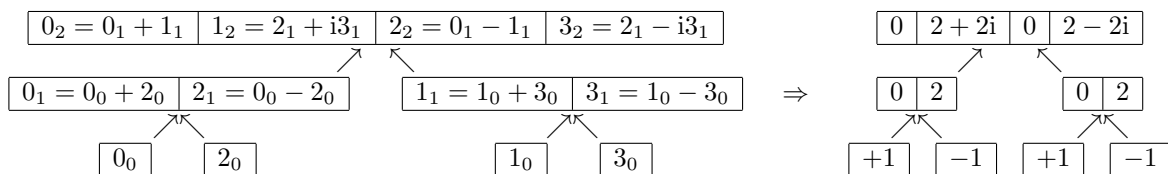
To execute FFT, we start from vectors of unit length and in each  $s$ -th stage,  $s = 1 \dots p$ , assemble  $2^{p-s}$  vectors of length  $2^s$  from vectors of length  $2^{s-1}$ : this costs  $2^{p-s} 2^{s-1} = 2^{p-1}$  products. Altogether, the cost of FFT is  $p2^{p-1} = \frac{1}{2} n \log_2 n$  products.



For  $n = 1024 = 2^{10}$ , say, the cost is  $\approx 5 \times 10^3$  products, compared to  $\approx 10^6$  for naive matrix multiplication! For  $n = 2^{20}$  the respective numbers are  $\approx 1.05 \times 10^7$  and  $\approx 1.1 \times 10^{12}$ , which represents a saving by a factor of more than  $10^5$ .

**Matlab demo:** Check out the online animation for computing the FFT at [http://www.maths.cam.ac.uk/undergrad/course/na/ii/fft\\_gui/fft\\_gui.php](http://www.maths.cam.ac.uk/undergrad/course/na/ii/fft_gui/fft_gui.php) and download the Matlab GUI from there to follow the computation of each single FFT term.

**Example 1.20** Computation of FFT for  $n = 4$  in general, and for the vector  $\mathbf{y} = (1, 1, -1, -1)$  in particular.



## 2 Partial differential equations of evolution

**Method 2.1** We consider the solution of the *diffusion equation*

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \quad 0 \leq x \leq 1, \quad t \geq 0,$$

with *initial conditions*  $u(x, 0) = u_0(x)$  for  $t = 0$  and *Dirichlet boundary conditions*  $u(0, t) = \phi_0(t)$  at  $x = 0$  and  $u(1, t) = \phi_1(t)$  at  $x = 1$ . By Taylor's expansion

$$\begin{aligned} \frac{\partial u(x, t)}{\partial t} &= \frac{1}{k} [u(x, t+k) - u(x, t)] + \mathcal{O}(k), & k = \Delta t, \\ \frac{\partial^2 u(x, t)}{\partial x^2} &= \frac{1}{h^2} [u(x-h, t) - 2u(x, t) + u(x+h, t)] + \mathcal{O}(h^2), & h = \Delta x, \end{aligned}$$

so that, for the true solution, we obtain

$$u(x, t+k) = u(x, t) + \frac{k}{h^2} [u(x-h, t) - 2u(x, t) + u(x+h, t)] + \mathcal{O}(k^2 + kh^2). \quad (2.1)$$

That motivates the numerical scheme for approximation  $u_m^n \approx u(x_m, t_n)$  on the rectangular mesh  $(x_m, t_n) = (mh, nk)$ :

$$u_m^{n+1} = u_m^n + \mu (u_{m-1}^n - 2u_m^n + u_{m+1}^n), \quad m = 1 \dots M. \quad (2.2)$$

Here  $h = \frac{1}{M+1}$  and  $\mu = \frac{k}{h^2} = \frac{\Delta t}{(\Delta x)^2}$  is the so-called *Courant number*. With  $\mu$  being fixed, we have  $k = \mu h^2$ , so that the local truncation error of the scheme is  $\mathcal{O}(h^4)$ . Substituting whenever necessary initial conditions  $u_m^0$  and boundary conditions  $u_0^n$  and  $u_{M+1}^n$ , we possess enough information to advance in (2.2) from  $\mathbf{u}^n := [u_1^n, \dots, u_M^n]$  to  $\mathbf{u}^{n+1} := [u_1^{n+1}, \dots, u_M^{n+1}]$ .

Similarly to ODEs or Poisson equation, we say that the method is *convergent* if, for a fixed  $\mu$ , and for every  $T > 0$ , we have

$$\lim_{h \rightarrow 0} \max_m |u_m^n - u(x_m, t_n)| = 0 \quad \text{uniformly for } (x_m, t_n) \in [0, 1] \times [0, T].$$

In the present case, however, a method has an extra parameter  $\mu$ , and it is entirely possible for a method to converge for some choice of  $\mu$  and diverge otherwise.

**Theorem 2.2** *If  $\mu \leq \frac{1}{2}$ , then method (2.2) converges.*

**Proof.** Let  $e_m^n := u_m^n - u(mh, nk)$  be the error of approximation, and let  $\mathbf{e}^n = [e_1^n, \dots, e_M^n]$  with  $\|\mathbf{e}^n\| := \max_m |e_m^n|$ . Convergence is equivalent to

$$\lim_{h \rightarrow 0} \max_{1 \leq n \leq T/k} \|\mathbf{e}^n\| = 0$$

for every constant  $T > 0$ . Subtracting (2.1) from (2.2), we obtain

$$\begin{aligned} e_m^{n+1} &= e_m^n + \mu (e_{m-1}^n - 2e_m^n + e_{m+1}^n) + \mathcal{O}(h^4) \\ &= \mu e_{m-1}^n + (1 - 2\mu)e_m^n + \mu e_{m+1}^n + \mathcal{O}(h^4). \end{aligned}$$

Then

$$\|\mathbf{e}^{n+1}\| = \max_m |e_m^{n+1}| \leq (2\mu + |1 - 2\mu|) \|\mathbf{e}^n\| + ch^4 = \|\mathbf{e}^n\| + ch^4,$$

by virtue of  $\mu \leq \frac{1}{2}$ . Since  $\|\mathbf{e}^0\| = 0$ , induction yields

$$\|\mathbf{e}^n\| \leq cnh^4 \leq \frac{cT}{k} h^4 = \frac{cT}{\mu} h^2 \rightarrow 0 \quad (h \rightarrow 0) \quad \square$$

**Discussion 2.3** In practice we wish to choose  $h$  and  $k$  of comparable size, therefore  $\mu = k/h^2$  is likely to be large. Consequently, the restriction of the last theorem is disappointing: unless we are willing to advance with tiny time step  $k$ , the method (2.2) is of limited practical interest. The situation is similar to stiff ODEs: like the Euler method, the scheme (2.2) is simple, plausible, explicit, easy to execute and analyse – but of very limited utility...

**Matlab demo:** Download the Matlab GUI for *Stability of 1D PDEs* from [http://www.maths.cam.ac.uk/undergrad/course/na/ii/pde\\_stability/pde\\_stability.php](http://www.maths.cam.ac.uk/undergrad/course/na/ii/pde_stability/pde_stability.php) and solve the diffusion equation in the interval  $[0, 1]$  with method (2.2) and  $\mu = 0.51 > \frac{1}{2}$ . Using (as preset) 100 grid points to discretise  $[0, 1]$  will then require the time steps to be  $5.1 \cdot 10^{-5}$ . The solution will evolve very slowly, but wait long enough to see what happens!