Numerical Analysis - Part II

Anders C. Hansen

Lecture 4

Solving PDEs with finite difference methods

Special structure of 5-point equations

Observation 1 (Special structure of 5-point equations)

We wish to motivate and introduce a family of efficient solution methods for the 5-point equations: the *fast Poisson solvers.* Thus, suppose that we are solving $\nabla^2 u = f$ in a square $m \times m$ grid with the 5-point formula (all this can be generalized a great deal, e.g. to the nine-point formula). Let the grid be enumerated in *natural ordering,* i.e. by columns. Thus, the linear system $A\boldsymbol{u} = \boldsymbol{b}$ can be written explicitly in the block form

$$\underbrace{\begin{bmatrix} B & I & & \\ I & B & \ddots & \\ & \ddots & \ddots & I \\ & & I & B \end{bmatrix}}_{A} \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{u}_m \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \vdots \\ \boldsymbol{b}_m \end{bmatrix}, \qquad B = \begin{bmatrix} -4 & 1 & & \\ 1 & -4 & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -4 \end{bmatrix}_{m\times m},$$

where $\boldsymbol{u}_k, \boldsymbol{b}_k \in \mathbb{R}^m$ are portions of $\boldsymbol{u}$ and $\boldsymbol{b}$, respectively, and B is a TST-matrix which means *tridiagonal, symmetric* and *Toeplitz* (i.e., constant along diagonals).

Special structure of 5-point equations

Observation 2 (Special structure of 5-point equations)

By Exercise 4, its eigenvalues and orthonormal eigenvectors are given as

$$B\boldsymbol{q}_\ell = \lambda_\ell \boldsymbol{q}_\ell, \qquad \lambda_\ell = -4 + 2\cos\tfrac{\ell\pi}{m+1},$$

$$\boldsymbol{q}_\ell = \gamma_m \big(\sin\tfrac{j\ell\pi}{m+1}\big)_{j=1}^{m}, \qquad \ell = 1..m,$$

where $\gamma_m = \sqrt{\frac{2}{m+1}}$ is the normalization factor. Hence $B = QDQ^{-1} = QDQ$, where $D = \operatorname{diag}(\lambda_\ell)$ and $Q = Q^T = (q_{j\ell})$. Note that all $m \times m$ TST matrices share the same full set of eigenvectors, hence they all commute!

The Hockney method

Set $\boldsymbol{v}_k = Q\boldsymbol{u}_k$, $\boldsymbol{c}_k = Q\boldsymbol{b}_k$, therefore our system becomes

$$\begin{bmatrix} D & I & & \\ I & D & \ddots & \\ & \ddots & \ddots & I \\ & & I & D \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \\ \vdots \\ \boldsymbol{v}_m \end{bmatrix} = \begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \\ \vdots \\ \boldsymbol{c}_m \end{bmatrix}.$$

Let us by this stage reorder the grid *by rows, instead of by columns.*. In other words, we permute $\boldsymbol{v} \mapsto \widehat{\boldsymbol{v}} = P\boldsymbol{v}$, $\boldsymbol{c} \mapsto \widehat{\boldsymbol{c}} = P\boldsymbol{c}$, so that the portion $\widehat{\boldsymbol{c}}_1$ is made out of the first components of the portions $\boldsymbol{c}_1, \dots, \boldsymbol{c}_m$, the portion $\widehat{\boldsymbol{c}}_2$ out of the second components and so on.

The Hockney method

This results in new system

$$\begin{bmatrix} \Lambda_1 & & & \\ & \Lambda_2 & & \\ & & \ddots & \\ & & & \Lambda_m \end{bmatrix} \begin{bmatrix} \widehat{\boldsymbol{v}}_1 \\ \widehat{\boldsymbol{v}}_2 \\ \vdots \\ \widehat{\boldsymbol{v}}_m \end{bmatrix} = \begin{bmatrix} \widehat{\boldsymbol{c}}_1 \\ \widehat{\boldsymbol{c}}_2 \\ \vdots \\ \widehat{\boldsymbol{c}}_m \end{bmatrix}, \quad \Lambda_k = \begin{bmatrix} \lambda_k & 1 & & \\ 1 & \lambda_k & 1 & \\ & \ddots & \ddots & \ddots \\ & & 1 & \lambda_k \end{bmatrix}_{m\times m},$$

where $k = 1...m$.

The Hockney method

These are m *uncoupled* systems, $\Lambda_k \widehat{\boldsymbol{v}}_k = \widehat{\boldsymbol{c}}_k$ for $k = 1...m$. Being *tridiagonal*, each such system can be solved fast, at the cost of $\mathcal{O}(m)$. Thus, the steps of the algorithm and their computational cost are as follows.

1. Form the products $\boldsymbol{c}_k = Q\boldsymbol{b}_k$, $k = 1...m$ .... $\mathcal{O}(m^3)$
2. Solve $m \times m$ tridiagonal systems $\Lambda_k \widehat{\boldsymbol{v}}_k = \widehat{\boldsymbol{c}}_k$, $k = 1...m$ ......$\mathcal{O}(m^2)$
3. Form the products $\boldsymbol{u}_k = Q\boldsymbol{v}_k$, $k = 1...m$ .... $\mathcal{O}(m^3)$

The improved Hockney method

We observe that the computational bottleneck is to be found in the $2m$ *matrix-vector products by the matrix* Q. Recall further that the elements of Q are $q_{j\ell} = \gamma_m \sin \frac{\pi j \ell}{m+1}$. This special form lends itself to a considerable speedup in matrix multiplication. Before making the problem simpler, however, let us make it more complicated! We write a typical product in the form

$$(Q\boldsymbol{y})_\ell = \sum_{j=1}^{m} \sin \frac{\pi j \ell}{m+1} y_j = \mathrm{Im} \sum_{j=0}^{m} \exp \frac{\mathrm{i}\pi j \ell}{m+1} y_j = \mathrm{Im} \sum_{j=0}^{2m+1} \exp \frac{2\mathrm{i}\pi j \ell}{2m+2} y_j, \tag{1}$$

where $y_{m+1} = \cdots = y_{2m+1} = 0$.

The discrete Fourier transform (DFT)

Definition 3 (The discrete Fourier transform (DFT))

Let Π_n be the space of all *bi-infinite complex n-periodic sequences* $\boldsymbol{x} = \{x_\ell\}_{\ell\in\mathbb{Z}}$ (such that $x_{\ell+n} = x_\ell$). Set $\omega_n = \exp\frac{2\pi \mathrm{i}}{n}$, the primitive root of unity of degree n. The *discrete Fourier transform (DFT)* of $\boldsymbol{x}$ is

$$\mathcal{F}_n : \Pi_n \to \Pi_n \quad \text{such that} \quad \boldsymbol{y} = \mathcal{F}_n \boldsymbol{x}, \quad \text{where} \quad y_j = \frac{1}{n}\sum_{\ell=0}^{n-1} \omega_n^{-j\ell} x_\ell,$$

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

Trivial exercise: You can easily prove that $\mathcal{F}_n$ is an isomorphism of Π_n onto itself and that

$$\boldsymbol{x} = \mathcal{F}_n^{-1}\boldsymbol{y}, \qquad \text{where} \qquad x_\ell = \sum_{j=0}^{n-1} \omega_n^{j\ell} y_j, \quad \ell = 0\ldots n-1.$$

The discrete Fourier transform (DFT)

An important observation: Thus, multiplication by Q in (1) can be reduced to calculating an inverse of DFT.
Since we need to evaluate DFT (or its inverse) only in a single period, we can do so by multiplying a vector by a matrix, at the cost of $\mathcal{O}(n^2)$ operations. This, however, is suboptimal and the cost of calculation can be lowered a great deal!

The fast Fourier transform (FFT)

We assume that n is a power of 2, i.e. $n = 2m = 2^p$, and for $\boldsymbol{y} \in \Pi_{2m}$, denote by

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

the even and odd portions of $\boldsymbol{y}$, respectively. Note that $\boldsymbol{y}^{(\mathrm{E})}, \boldsymbol{y}^{(\mathrm{O})} \in \Pi_m$.
Suppose that we already know the inverse DFT of both ‘short’ sequences,

$$\boldsymbol{x}^{(\mathrm{E})} = \mathcal{F}_m^{-1}\boldsymbol{y}^{(\mathrm{E})}, \qquad \boldsymbol{x}^{(\mathrm{O})} = \mathcal{F}_m^{-1}\boldsymbol{y}^{(\mathrm{O})}.$$

Computing the fast Fourier transform (FFT)

It is then possible to assemble $\boldsymbol{x} = \mathcal{F}_{2m}^{-1}\boldsymbol{y}$ in a small number of operations. Note that we have $\omega_{2m}^2 = \omega_m$, thus

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

where $\ell = 0, \ldots, m-1$.

Computing the fast Fourier transform (FFT)

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

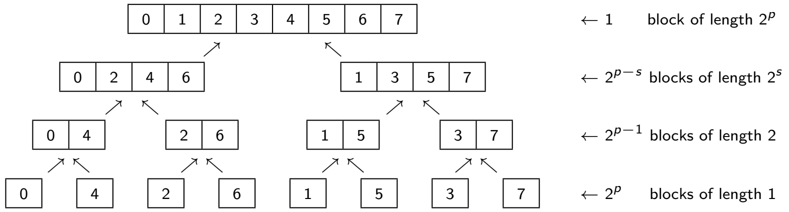
$$\omega_m^{j(m+\ell)} = \omega_m^{j\ell}, \quad \omega_{2m}^{m+\ell} = -\omega_{2m}^{\ell} \quad \Rightarrow \quad x_{m+\ell} = x_\ell^{(\mathrm{E})} - \omega_{2m}^{\ell} x_\ell^{(\mathrm{O})},$$

where $\ell = 0, \ldots, m-1$.

Note: To execute FFT, we start from vectors of unit length and in each s-th stage, $s = 1 \ldots p$, 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.

Computing the fast Fourier transform (FFT)

Altogether, the cost of FFT is $p2^{p-1} = \frac{1}{2} n \log_2 n$ products.


0 1 2 3 4 5 6 7
0 2 4 6
1 3 5 7
0 4
2 6
1 5
3 7
0
4
2
6
1
5
3
7
$\leftarrow 1$ block of length 2^p
$\leftarrow 2^{p-s}$ blocks of length 2^s
$\leftarrow 2^{p-1}$ blocks of length 2
$\leftarrow 2^p$ blocks of length 1

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.damtp.cam.ac.uk/user/hf323/M21-II-NA/demos/fft_gui/fft_gui.html` and download the Matlab GUI from there to follow the computation of each single FFT term.

Numerical analysis vs Foundations of computational matematics

Numerical analysis is mostly concerned with providing upper bounds.

Foundations of computational mathematics is concerned with determining the boundaries of what computers can achieve. That means also lower bounds.

Is $n \log(n)$ the best one can do when computing the discrete Fourier transform? Does there exist an algorithm that is faster?

Partial differential equations of evolution

Recall the Poisson equation

Recall the *Poisson equation*

$$\nabla^2 u = f \qquad (x, y) \in \Omega, \tag{2}$$

where $\nabla^2 = \Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ is the Laplace operator and Ω is an open connected domain of $\mathbb{R}^2$ with a Jordan boundary, specified together with the *Dirichlet boundary condition*

$$u(x, y) = \phi(x, y) \qquad (x, y) \in \partial\Omega. \tag{3}$$

(You may assume that $f \in C(\Omega)$, $\phi \in C^2(\partial\Omega)$, but this can be relaxed by an approach outside the scope of this course.)

Solving the diffusion equation

We consider the solution of the *diffusion equation*

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \qquad 0 \le x \le 1, \quad t \ge 0,$$

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

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

so that, for the true solution, we obtain

$$u(x,t+k) = u(x,t) + \tfrac{k}{h^2}\big[u(x-h,t) - 2u(x,t) + u(x+h,t)\big] + \mathcal{O}(k^2 + kh^2)\,. \tag{4}$$

Numerical scheme for the diffusion equation

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

$$u_m^{n+1} = u_m^n + \mu \left(u_{m-1}^n - 2u_m^n + u_{m+1}^n \right), \qquad m = 1...M\,. \qquad (5)$$

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 μ 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 (5) from $\boldsymbol{u}^n := [u_1^n, \ldots, u_M^n]$ to $\boldsymbol{u}^{n+1} := [u_1^{n+1}, \ldots, u_M^{n+1}]$.

Convergence

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

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

In other words, if $e_m^n := u_m^n - u(mh, nk)$ is the error of approximation, and $\boldsymbol{e}^n = [e_1^n, \dots, e_M^n]$ with $\|\boldsymbol{e}^n\| := \max_m |e_m^n|$, then convergence is equivalent to

$$\lim_{h\to 0} \max_{1\le n\le T/k} \|\boldsymbol{e}^n\| = 0.$$

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

Proving convergence

Theorem 4

If $\mu \leq \frac{1}{2}$, then method (5) converges.

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

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

for every constant $T > 0$. Subtracting (4) from (5), 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

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

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

$$\|\boldsymbol{e}^n\| \leq cnh^4 \leq \frac{cT}{k}\,h^4 = \frac{cT}{\mu}\,h^2 \to 0 \qquad (h\to 0) \qquad \square$$

Practical issues

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 (5) is of limited practical interest. The situation is similar to stiff ODEs: like the Euler method, the scheme (5) is simple, plausible, explicit, easy to execute and analyse – but of very limited utility. . . .

Matlab demo

Matlab demo: Download the Matlab GUI for *Stability of 1D PDEs* from `http://www.damtp.cam.ac.uk/user/hf323/M21-II-NA/demos/pde_stability/pde_stability.html` and solve the diffusion equation in the interval $[0, 1]$ with method (5) 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!