Solution Methods for Distributed Parameter Systems

Fredrik Mikal Hjelmeland Rygg
NTNU

July 2005

Chapter 1

Introduction

Due to efficient solution methods for system of partial differential equations, PDEs, *distributed parameter systems* have occupied an important place in control and system theory. The states of a control problem that is described by PDEs, depends not only on time, but also space. The classical theory of PDE is well developed, but still new mathematical techniqes emerge. One such techniqe, of value for a control-engineer, is the theory of model reduction. Model reduction is not presented here due to time restrictions [1] . If the reader is interested in model reduction, we recommend the course material from a course on model reduction, thought by Dr Weiland at the university of Eidhoven.
It was the intention to provide a simulation example of a distributed control system, thid turned out to be difficult. First it was wery hard in the beginning to understand the examples that i found in the literature, second when all the practical schemes required, for the author, non-familiar solution methods. From the authors opinion, and experience, it is essential to understand numerical linear algebra, especially solution methods for sparse sytems, in order to solve a distributed control system. For example the discretization of poissons problem on a 81×81 grid with a five-point formula results in 6400 equations, which is a modest problem. in three diemnsions this rises to 80^3. The Poisson problem is simple, for a combustion problem depending on, say, 14 parameters the number of equations are $14 \times 80^3 = 7168000$ equations. The cost of finding the solution with the familiar *Gaussian elimination* is $\mathcal{O}(d^3)$ for a $d \times d$ system, and this renders it useless of size such as the above. This is the reason why we choose to focus on efficient methods for solving

[1] I admit that this document is somewhat incomplete. As usual i ended up in a hurry as the time limit dangerously approached. I have not read through this document, and im not suprised if it contains a lot of errors. I hope it is not to painful to read. Also i remark that the notation is not always consistent

large, and sparse systems. We mention for the inerested reader [26] contains several examples of distributed control problems, and algorithms to solve them. Also of interest are the siam books in the series *Advances in Design and Control*.
[2]

[2]I borrowed this book ones, and started to simulate a Stefan-like (moving boundary) control problem. Unfourtunately i had to deliver the book at the library before i could finish. I tried to borrow it again, but the book is not available in Norway, it had to be imported from USA. Im still waiting for the book to arive.

Chapter 2

Finite-Difference Approximations

2.1 Finite Differences

The main idea in the classical theory of *finite differences* is to replace derivatives with linear combination of discrete function values, thus reducing a differential equation to an algerbraic system.
We wish to find the numerical solution by finite differences of a general linear PDE

$$\begin{aligned} \frac{\partial u}{\partial t} &= \mathcal{L}u + f \qquad \mathbf{x} \in \Omega \subset \mathbb{R}^n, \quad t \geq 0, && (2.1)\\ \mathcal{B}u &= \phi \qquad \mathbf{x} \in \partial\Omega \subset \mathbb{R}^{n-1} && (2.2)\\ u(\mathbf{x}, 0) &= g(\mathbf{x}) \qquad \mathbf{x} \in \Omega && (2.3) \end{aligned}$$

where $\phi = \phi(\mathbf{x}, t)$ is the *boundary data*, $g(\mathbf{x})$ is the initial data, and $f = f(\mathbf{x}, t)$ is a known source term. $\mathcal{L}$ is a *linear differential operator*.

$$\mathcal{L} = \sum_{i_1+i_2+\ldots+i_n \leq r} a_{i_1,i_2,\ldots,i_n} \frac{\partial^{i_1+i_2+\ldots i_s}}{\partial x_1^{i_1} \partial x_2^{i_2} \ldots \partial x_n^{i_n}}.$$

Similarly $\mathcal{B}$ is a linear operator on the boundary.

$$\mathcal{B}u = \alpha + \beta \frac{\partial u}{\partial \mathbf{n}}$$

where $\mathbf{n}$ is the outward normal. Let Ω_h be a computational *grid* over Ω, and let $\mathcal{L}_h$ be an approximation to the derivative $\mathcal{L}u$ at the *grid points*.

A approximation at the boundary nodes is also necessary if the solution is unknown there. This result in a linear system of difference equations

$$\begin{aligned} \mathcal{L}_h \mathbf{u}_h &= \mathbf{f}_h \qquad \mathbf{x}_h \in \Omega_h \\ \mathcal{B}_h \mathbf{u}_h &= \phi \qquad \mathbf{x}_h \in \partial\Omega_h \end{aligned}$$

To be useful the FD solution must resemble the true solution as the node diststance $h \to 0$. We say that the method is *consistent* if

$$\begin{aligned} \mathcal{L}_h u_{ij} &\xrightarrow{h\to 0} \mathcal{L}u_{ij} \qquad \text{for all} \quad \mathbf{x}_h \in \Omega \\ \mathcal{B}_h u_{ij} &\xrightarrow{h\to 0} \mathcal{B}u_{ij} \qquad \text{for all} \quad \mathbf{x}_h \in \partial\Omega \end{aligned}$$

The method is *pointwise* convergent if

$$U_{ij} \xrightarrow{h\to 0} u_{ij}, \qquad \text{for all} \quad \mathbf{x}_h \in \overline{\Omega}.$$

2.1.1 Taylor Expansion

We commence by finding consistent finite differece formulas for the dervia-tives. It is common practice to obtain a difference approximation by means a Taylor expansion. The *forward difference* approximation

$$u'(x) \approx \frac{u(x+h) - u(x)}{h}$$

is derived from the Taylor expansion of $u(x+h)$:

$$u(x+h) = u(x) + hu'(x) + \frac{1}{2}h^2u''(x) + \frac{1}{6}h^3u'''(\xi_1)$$

In a similar manner the *backward difference* approximation

$$u'(x) \approx \frac{u(x) - u(x-h)}{h}$$

is derived from the Taylor expansion of $u(x-h)$:

$$u(x-h) = u(x) - hu'(x) + \frac{1}{2} - \frac{1}{6}u'''(\xi_2).$$

If we subtract $u(x+h)$ from $u(x-h)$ and solve for $u'(x)$, we obtain the *centered difference* approximation

$$u'(x) \approx \frac{u(x+h) - u(x-h)}{2h}. \tag{2.4}$$

Supposing that u has a second derivative, the forward and backward differences are first order accurate. If the third derivative of f exist, the centered difference is second order accurate.
The centered difference approximation of the second derivative is obtained by adding $u(x+h)$ to $u(x-h)$, and then solve for $u''(x)$:

$$u''(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + \frac{1}{12}h^2 u(\eta)$$

where $x-h \leq \eta \leq x+h$. When u has a fourth derivative, the approximation

$$u''(x) \approx \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} \tag{2.5}$$

is second order accurate.

2.1.2 Taylor Tables

A simple and convenient way of forming a difference scheme of any order is to constuct a *Taylor table*. The Taylor table is best explained by an example. Consider the Taylor table for a 3-point backward differencing operator reprecenting a first derivative with $h = \Delta x$.

$$\left(\frac{\partial u}{\partial x}\right)_j - \frac{1}{\Delta x}(a_2 u_{j-2} + a_1 u_{j-1} + a_0 u_j) = ?$$

	u_j	$\Delta x\left(\frac{\partial u}{\partial x}\right)_j$	$\Delta x^2\left(\frac{\partial^2 u}{\partial x^2}\right)_j$	$\Delta x^3\left(\frac{\partial^3 u}{\partial x^3}\right)_j$	$\Delta x^4\left(\frac{\partial^4 u}{\partial x^4}\right)_j$
$\Delta x\left(\frac{\partial u}{\partial x}\right)_j$		1			
$-a_2 \cdot u_{j-2}$	$-a_2$	$a_2 \cdot 2$	$-a_2 \cdot 2$	$a_2 \cdot \frac{4}{3}$	$-a_2 \cdot \frac{2}{3}$
$-a_1 \cdot u_{j-1}$	$-a_1$	a_1	$-a_1 \cdot \frac{1}{2}$	$a_1 \cdot \frac{1}{6}$	$-a_1 \cdot \frac{1}{24}$
$-a_0 \cdot u_0$	$-a_0$				

At the top of the table the unknown error of an approximation is displayed. Our goal is to find the constants in the error to maximize the order of accuracy. In each row you find the coefficients of the Taylor expansion of the leftmost term. For example, the second row in the table corresponds to the Taylor series expansion of $-a_2 \cdot u_{j-2}$

$$\begin{aligned} -a_2 u_{j-21} &= -a_2 u_j - a_2 \cdot (-2) \cdot \Delta x\left(\frac{\partial u}{\partial x}\right)_j - a_2 \cdot (-2)^2 \cdot \frac{1}{2}\Delta x^2\left(\frac{\partial^2 u}{\partial x^2}\right)_j \\ &\quad -a_2 \cdot (-2)^3 \cdot \frac{1}{6} \cdot \left(\frac{\partial^3 u}{\partial x^4}\right)_j - a_2 \cdot (-2)^4 \cdot \frac{1}{24}\Delta x^4\left(\frac{\partial^4 u}{\partial x^4}\right) - \ldots \ .. \end{aligned}$$

There are thre unknow a_2, a_1 and a_0. To get a method with second order accuracy, the first three columns must sum to zero

$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 1 & 0 \\ -4 & -1 & 0 \end{bmatrix} \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$$

which gives $[a_2, a_1, a_0] = \frac{1}{2}[1, -4, 3]$. Thus a second-order backward difference approximation to the first derivative is

$$\Big(\frac{\partial u}{\partial x}\Big)_j = \frac{1}{2\Delta x}(u_{j-2} - 4u_{j-1} + 3u_j) + \mathcal{O}(\Delta x^2) \tag{2.6}$$

The truncation error er_t is found from the first non-vanishing column sum. In this case the fourth column provides the leading truncation error term

$$er_t = \frac{1}{2\Delta x}\Big[\frac{8a_2}{6} + \frac{a_1}{6}\Big]\Delta x^3 \Big(\frac{\partial^3 u}{\partial x^3}\Big)_j = \frac{\Delta x^2}{3}\Big(\frac{\partial^3 u}{\partial x^3}\Big)_j$$

2.1.3 Matrix Difference Equations

The difference relation

$$(\delta_x)_j = \frac{1}{2\Delta x}(u_{j+1} - u_{j-1}) \tag{2.7}$$

$$(\delta_{xx})_j = \frac{1}{2\Delta x}(u_{j+1} - 2u_j + u_{j-1}) \tag{2.8}$$

defines *point difference* operators since they give an approximation to the derivative at grid points in terms of surrounding points. We recognize (2.7) and (2.8) as the three-point centered difference approximations for the first and second derivatives. They correspond to (2.4) and (2.5) derived previously. However, point operators dont tell us how other points in the mesh are differenced, or how boundary conditions are enforced. Now the *matrix* operator comes to rescue. It is best explained by an example. Let us derive the the matrix representation of (2.8) on a four-point mesh with boundary points a and b, and Dirichlet boundary conditions, $u(0) = u_a$, $u(1) = u_b$. If we write down the point difference formula for every *interior* point we arive at the system of equations

$$\begin{aligned}
(\delta_{xx})_1 &= \frac{1}{\Delta x^2}(u_a - 2u_1 + u_2) \\
(\delta_{xx})_2 &= \frac{1}{\Delta x^2}(u_1 - 2u_2 + u_3) \\
(\delta_{xx})_3 &= \frac{1}{\Delta x^2}(u_2 - 2u_3 + u_4) \\
(\delta_{xx})_4 &= \frac{1}{\Delta x^2}(u_3 - 2u_4 + u_b)
\end{aligned}$$

In matrix notation this becomes

$$\delta_{xx}\mathbf{u} = A\mathbf{u} + \mathbf{bc}$$

where

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}, \quad \mathbf{bc} = \frac{1}{\Delta x^2}\begin{bmatrix} u_a \\ 0 \\ 0 \\ u_b \end{bmatrix} \quad \text{and} \quad A = \begin{bmatrix} -2 & 1 & & \\ 1 & -2 & 1 & \\ & 1 & -2 & 1 \\ & & 1 & -2 \end{bmatrix}$$

The matrix representation of (2.7) can be obtained in a similar manner. Thus the matrix operators representing the three-point central difference approximations (2.7)and (2.8) on a four point mesh with Dirichlet boundary conditions are

$$\delta_x = \frac{1}{2\Delta x}\begin{bmatrix} 0 & 1 & & \\ -1 & 0 & 1 & \\ & -1 & 0 & 1 \\ & & -1 & 0 \end{bmatrix}, \quad \delta_{xx} = \frac{1}{\Delta x}^2\begin{bmatrix} -2 & 1 & & \\ 1 & -2 & 1 & \\ & 1 & -2 & 1 \\ & & 1 & -2 \end{bmatrix} \tag{2.9}$$

The structure of these matrix operators are important. They are *sparse* and *banded*. A matrix A is sparse if each variable is coupled to just a few other variables. It is banded with *bandwidth* s if $a_{k,l} = 0$ for every $k, l \in \{1, 2, ..., N\}$ such that $|k - l| > s$. Here N denotes the number of *interior* points. A practical notation for a banded matrix is

$$B(N : a_{-1}, a_0, a_1) = \begin{bmatrix} a_0 & a_1 & & & \\ a_{-1} & a_0 & a_1 & & \\ & & \ddots & & \\ & & a_{-1} & a_0 & a_1 \\ & & & a_{-1} & a_0 \end{bmatrix} \tag{2.10}$$

Use of the matrix dimension N is optional. The illustration is given for a *tridiagonal* matrix although any number of bands is a posibility. A tridiagonal matrix without constants along the band can be expressed as $B(\mathbf{a}_{-1}, \mathbf{a}_0, \mathbf{a}_1$. The arguments for a banded matrix are always odd in number, and the central one *always* refers to the central diagonal.

2.2 Difference Operators at Boundaries

Each line of a matrix difference equation is derived from a point operator, but the point operators can differe from point to point. For instance, a point

operator near the boundary might differe from one in the strict interior due to imposed boundary conditions. To illustrate how different point operators affect the matrix operator, we replace the right Dirichlet condition $u(1) = u_b$ with a Neumann boundary condition

$$\left(\frac{\partial u}{\partial x}\right)_{x=1} = \left(\frac{\partial u}{\partial x}\right)_b \tag{2.11}$$

specified at $j = N + 1$. The interior operator is again

$$(\delta_{xx}u)_j = \frac{1}{\Delta x^2}(u_{j+1} - 2u_j + u_{j-1}) \tag{2.12}$$

At node N we cant simply use (2.2) as it stand, because u_{N+1} is undefined. However the Neumann condition can be put to use. We seek an operator at node N in the following form:

$$(\delta_{xx}u)_j = \frac{1}{\Delta x^2}(au_{N-1} + bu_N) + \frac{c}{\Delta x}\left(\frac{\partial u}{\partial x}\right)_{N+1} \tag{2.13}$$

where a, b and c are constants to be found. From the Taylor table

$$\left(\tfrac{\partial^2 u}{\partial x^2}\right)_j - \left[\tfrac{1}{\Delta x^2}(au_{j-1} + bu_j) + \tfrac{c}{\Delta x}\left(\tfrac{\partial u}{\partial x}\right)_{j+1}\right] = ?$$

	u_j	$\Delta x\left(\frac{\partial u}{\partial x}\right)_j$	$\Delta x^2\left(\frac{\partial^2 u}{\partial x^2}\right)_j$	$\Delta x^3\left(\frac{\partial^3 u}{\partial x^3}\right)_j$	$\Delta x^4\left(\frac{\partial^4 u}{\partial x^4}\right)_j$
$\Delta x^2\left(\frac{\partial^2 u}{\partial x^2}\right)_j$			1		
$-a \cdot u_{j-1}$	$-a$	a	$-a \cdot \frac{1}{2}$	$a \cdot \frac{1}{6}$	$-a \cdot \frac{1}{24}$
$-b \cdot u_j$	$-b$				
$-\Delta x \cdot c \cdot \left(\frac{\partial u}{\partial x}\right)_{j+1}$		$-c$	$-c$	$-c \cdot \frac{1}{2}$	$-c \cdot \frac{1}{6}$

we see that the method is second order accurate if

$$\begin{bmatrix} -1 & -1 & 0 \\ 1 & 0 & -1 \\ -1 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

Solving for a, b and c, we obtain the following point operator

$$(\delta_{xx}u)_N = \frac{1}{3\Delta x^2}(2u_{N-1} - 2u_N) + \frac{2}{3\Delta x}\left(\frac{\partial u}{\partial x}\right)_{N+1} \tag{2.14}$$

What about the truncation error? From the fourth column sum we obtain

$$\Delta x^2\Big(\frac{\partial^2 u}{\partial x^2}\Big)_j - \frac{1}{\Delta x^2}(2u_{N-1} - 2u_N) - \frac{2}{3\Delta x}\Big(\frac{\partial u}{\partial x}\Big)_j = -\frac{2}{9}\Delta x^3\Big(\frac{\partial^3 u}{\partial x^3}\Big)_j + \mathcal{O}(\Delta x^4)$$

Thus the operator is first-order accurate, however, this dont destroy the overall secod-order accuracy. With $(\delta_{xx})_N$ at hand the matrix operator reads

$$\delta_{xx}\mathbf{u} = \frac{1}{\Delta x^2}B(\mathbf{a}_{-1}, \mathbf{a}_0, 1)\mathbf{u} + \mathbf{bc}, \tag{2.15}$$

where

$$\begin{aligned} \mathbf{a}_{-1} &= [1, 1, ..., 2/3]^T \\ \mathbf{a}_0 &= [-2, -2, -2, ..., -2/3]^T \\ \mathbf{bc} &= \frac{1}{\Delta x^2}[u_a, 0, 0, ..., \frac{2\Delta x}{3}(\frac{\partial u}{\partial x})_b]^T. \end{aligned}$$

We can also obtain the point operator (2.14) using space extrapolation. If we approximate the Neumann condition with the backward-difference (2.6), then

$$\Big(\frac{\partial u}{\partial x}\Big)_{N+1} = \frac{1}{2\Delta x}(u_{N-1} - 4u_N + 3u_{N+1}) + \mathcal{O}(\Delta x^2).$$

Solving for u_{M+1} gives

$$u_{N+1} = \frac{1}{3}\Big[4u_N - u_{N-1} + 2\Delta x\Big(\frac{\partial u}{\partial x}\Big)_b\Big] + \mathcal{O}(\Delta x^3)$$

Sustituting this into $(u_{xx})_N$ yields

$$\begin{aligned} (\delta_{xx}u)_N &= \frac{1}{\Delta x^2}(u_{N+1} - 2u_N + u_{N-1}) \\ &= \frac{1}{3\Delta x^2}\Big[3u_{N-1} - 6u_N + 4u_N - u_{N-1} + 2\Delta x\Big(\frac{\partial u}{\partial x}\Big)_b\Big] \\ &= \frac{1}{3\Delta x^2}(2u_{N-1} - 2u_N) + \frac{2}{3\Delta x}\Big(\frac{\partial u}{\partial x}\Big)_b, \end{aligned}$$

which is identical to (2.15).

2.3 Poisson's Equation in One Dimension

We can now approximate a differential equation using finite differences. Consider the one-dimensional Poisson equation.

$$-\frac{d^2u(x)}{d^2x} = f(x) \qquad 0 < x < 1 \tag{2.16}$$

To converte the differential equation (2.16) to a difference equation, we *discretize* the problem by computing approximate solutions at $N+2$ evenly spaced points $x_i = ih$ between 0 and 1, where $h = \frac{1}{N+1}$, and $0 \leq i \leq N+1$. We then apply the centered difference scheme for the second derivative (2.5). This yield the difference equation

$$-u_{i-1} + 2u_i - u_{i+1} = h^2 f_i + h^2 \tau_i$$

where τ_i is the truncation error. If we impose the Dirichlet conditions, $u_0 = u_{N+1} = 0$, we obtain N equations in N unknowns $u_1, u_2, ..., u_N$:

$$T_N \cdot \begin{bmatrix} u_1 \\ \vdots \\ \vdots \\ u_N \end{bmatrix} = \begin{bmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ \vdots \\ \vdots \\ u_N \end{bmatrix} = h^2 \begin{bmatrix} f_1 \\ \vdots \\ \vdots \\ f_N \end{bmatrix} + h^2 \begin{bmatrix} \tau_1 \\ \vdots \\ \vdots \\ \tau_N \end{bmatrix} \tag{2.17}$$

It is convenient to use the notation $B(a_{-1}, a_0, a_1)$ for a tridiagonal matrix with constants along the bands. In the new notation the *banded* matrix equation (2.17) becomes

$$B(-1, 2, -1)\mathbf{u} = \mathbf{f} + h^2 \boldsymbol{\tau}$$

2.3.1 TST Matrices

The matrix operator T_N plays a central role in the analysis of discretized PDEs, and in the computation of fast solutions to the Poisson equation. Our friend T_N is symmetric and and tridiagonal, with constants on the diagonals. A $N \times N$ matrix $A = (a_{i,j})_{k,l=1}^N$ is said to be *Toeplitz* if it is constant along its diagonals, in other words, if there exist numbers $\tau_{-N+1}, \tau_{-N+2}, ..., \tau_0, ..., \tau_{N-1}$ such that

$$a_{k,l} = \tau_{k-l}, \qquad k, l = 1, 2..., n.$$

The matrix T_N is a Toeplitz matrix with

$$\tau_{-N+1} = ... = \tau_{-2} = 0, \quad \tau_{-1} = 1, \quad \tau_0 = -2, \quad \tau_1 = 1, \quad \tau_2 = ... = \tau_{N-1} = 0.$$

We say tha a matrix is TST if it is Toeplitz, symmetric and tridiagonal. Therefore, A is TST if

$$\tau_j = 0, \quad |j| = 2, 3, ..., N-1, \qquad \tau_{-1} = \tau_1.$$

We are interested in the eigenvalues and eigenvectors of a TST matrix.

Lemma 2.1 *Let A be a $N \times N$ TST matrix and $\alpha := a_0,\ \beta := a_{-1} = a_1$. Then the eigenvalues of A are*

$$\lambda_j = \alpha + 2\beta \cos\Big(\frac{\pi j}{N+1}\Big), \qquad j = 1, 2, ..., N, \tag{2.18}$$

each with corresponding orthnormal eigenvector, $\mathbf{q_j}$, where

$$q_{j,l} = \sqrt{\frac{2}{N+1}} \sin\Big(\frac{\pi j l}{N+1}\Big), \qquad j, l = 1, 2, ..., N. \tag{2.19}$$

Proof Suppose that λ is an eigenvalue of A with corresponding eigenvector $\mathbf{q}$, in other words, $A\mathbf{q} = \lambda\mathbf{q}$. Letting $q_0 = q_{N+1} = 0$, we arrive at the equations

$$\beta q_{l+1} + (a - \lambda)q_l + \beta q_{l-1} = 0, \qquad l = 1, 2, ..., N$$

This is a difference equation with the general sloution

$$q_l = a\eta_1^l + b\eta_2^l \qquad l = 0, 1, ..., d+1,$$

where $\eta_i,\ i = 1, 2$ are zeros of the characteristic polynomial

$$\beta\eta^2 + (\alpha - \lambda)\eta + \beta = 0.$$

The constraint $q_0 = q_{N+1} = 0$ determine the constants a and b. The first condition yields $a + b = 0$, hence $q_l = a(\eta_1^l - \eta_2^l)$, where $a \neq 0$ is arbitrary. The second condition is fulfilled when

$$\eta_1^{N+1} = \eta_2^{N+1}$$

hence

$$\eta_1 = \eta_2 \exp\Big(\frac{2\pi \mathrm{i} j}{N+1}\Big), \qquad j = 0, 1, ..., N \tag{2.20}$$

The case $j = 0$ corresponds to the trivial eigenvector $q_l = 0$, and can be discarded. We insert the roots η_1, η_2 into (5) and multiply by $\exp[-\pi \mathrm{i} j/(N+1)]$,

$$\Big(\lambda - \alpha\sqrt{(\lambda-\alpha)^2 - 4\beta^2}\Big) \exp\Big(\frac{-\pi \mathrm{i} j}{N+1}\Big) = \Big(\lambda - \alpha - \sqrt{(\lambda-\alpha)^2 - 4\beta^2}\Big) \exp\Big(\frac{\pi \mathrm{i} j}{N+1}\Big)$$

Using the Euler identity, we obtain

$$\sqrt{(\lambda-\alpha)^2 - \beta^2} \cos\Big(\frac{\pi j}{N+1}\Big) = (\lambda - \alpha)\mathrm{i} \sin\Big(\frac{\pi j}{N+1}\Big).$$

We square this equation on both sides, and solve for the eigenvalues

$$\lambda = \alpha \pm 2\beta \cos\Big(\frac{\pi j}{N+1}\Big).$$

Taking the plus sign we recover (2.18), while the minus repats $\lambda = \lambda_{N+1-j}$. The eigenvectors are found by inserting the eigenvalues (2.18) into the roots of the characteristic polynomial

$$\eta = \cos\Big(\frac{\pi j}{N+1}\Big) \pm \mathrm{i}\sin\Big(\frac{\pi j}{N+1}\Big) = \exp\Big(\frac{\pm\pi \mathrm{i} j}{N+1}\Big),$$

therefore

$$q_{j,l} = a(\eta_1^l - \eta_2^l) = 2a\mathrm{i}\sin\Big(\frac{\pi j l}{N+1}\Big), \qquad j,l = 1,2,...,N.$$

The vectors $\mathbf{q}_j,\ j = 1,2,...,N$ are orrhogonal, they are also normal if

$$\sum_{l=1}^{N} q_{j,l}^2 = 1$$

It can be shown that

$$\sum_{l=1}^{N} \sin^2\Big(\frac{\pi j l}{N+1}\Big) = \frac{1}{2}(N+1).$$

Using this to eliminate a we obtain the eigenvectors in (2.19) □

We return to the matrix T_N. From (2.18)and (2.19) we conclude that the eigenvalues and the corresponding eigenvectors of T_N are given by

$$\begin{aligned} \lambda_j &= 2 - 2\cos\Big(\frac{\pi j}{N+1}\Big) \\ q_{j,l} &= \sqrt{\frac{jk\pi}{N+1}}\sin\Big(\frac{jk\pi}{N+1}\Big) \end{aligned}$$

Figure (2.3.1) is a plot of the eigenvalues for $N = 21$. The largest eigenvalue is $\lambda_N = 2(1 - \cos\frac{\pi N}{N+1}) \approx 4$. The smallest eigenvalue is $\lambda_1 = 2(1-\cos\frac{\pi}{N+1}) \approx (\frac{\pi}{N+1})^2$. Thus T_N is positive definite with condition number $\lambda_N/\lambda_1 \approx 4(N+1)^2/\pi^2$ for large N. The eigenvectors are sinusoids with lowest frequency at $j = 1$ and highest at $j = N$. Some of the eigenvectors are shown in figure (2.3.1).

Figure 2.1: Eigenvectors of T_{21}

Figure 2.2: Eigenvalues of T_{21}

To prove that the difference approximation is consistent we provide a bound on the error $\|u - u_h\|$

$$\|u - u_h\| \leq h^2 \|T_N^- 1\|_2 \|\boldsymbol{\tau}\|_2 \approx h^2 \frac{(N+1)^2}{\pi^2} \|\tau\|_2 = \mathcal{O}(\|\boldsymbol{\tau}\|_2) = \mathcal{O}\Big(h^2 \|\frac{d^4 u}{dx^4}\|_{\inf}\Big).$$

This shows that the error goes to zero proportionally to h^2, provided the solution is smooth enough.
It turns out that the eigenvalues and eigenvectors of $h^{-2}T_N$ also approximate the eigenvalues and *eigenfunctions* of the differential equation, in other words, the error in Fourier space goes to zero. The eigenvalues and *eigenfunctions* of the continuous problem must satisfy the eigenvalue equation

$$-\frac{d^2 q}{dx^2} = \lambda q \qquad u(0) = u(1) = 0$$

The eigenvalues are $i\pi$ and the eigenvectors are $sin(i\pi x)$. Thus the eigenvectors are *precisely* equal to the eigenfunction q evaluated at the sample points $x_j = jh$ when scaled by $\sqrt{2h}$. For the low modes $\lambda_i = (ih\pi)^2 + \mathcal{O}(h^2)$. The high modes however, are not resolved properly.

2.4 Poisson's Equation in Two Dimensions

The Poisson equation is perhaps the most important PDE, it is also one of the simplest. Solutions of stationary conduction, and diffusion problems satisfy the Poisson equation, so does the potential of a magnetic or electric field. In two dimension it reads

$$\Delta u = f, \qquad (x, y) \in \Omega \tag{2.21}$$

where

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

We consider the discrete Poisson equation on the unit square
$\Omega = \{(x, y) : \ 0 < x, y < 1\}$ with Dirichlet boundary conditions

$$\begin{aligned} -\Delta_h \mathbf{u_h} &= \mathbf{f}_h \qquad (x_i, y_j) \in \Omega_h \\ \mathbf{u}_h &= \boldsymbol{\phi_h} \qquad (x_j, y_j) \in \Gamma_h = \partial\Omega_h \end{aligned}$$

Here Δ_h is an approximation to the Laplace operator We assume an equispaced grid with spacing h in x and y. Then

$$x_i = ih \qquad y_j = jh \qquad h = \frac{1}{N+1}, \quad n \in \mathbb{N}$$

where N is the number of internal nodes in each direction. We abbreviate $u_{ij} = u(ih, jh)$ and $f_{ij} = f(ih, jh)$.
Now apply the centered scheme (2.5) to approximate Δu. We know that

$$\begin{aligned} -\frac{\partial^2 u(x,y)}{\partial x^2}\Big|_{x=x_i,y=y_i} &\approx \frac{2u_{i,j} - u_{i-1,j} - u_{i+1,j}}{h^2} \quad \text{and} \\ -\frac{\partial^2 u(x,y)}{\partial y^2}\Big|_{x=x_i,y=y_j} &\approx \frac{2u_{i,j} - u_{i,j-1} - u_{i,j+1}}{h^2} \end{aligned}$$

Adding these approximations gives us

$$-\Delta u(x,y)|_{x=x_i,y=y_i} = \frac{4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1}}{h^2} - \tau_{ij} \qquad (2.22)$$

where the truncation τ_{ij} is bounded by $\mathcal{O}(h^2)$.
A pictorial representation of the discretization is given by the *stencil*, also known as the computational molecule. A stencil represents a *grid operator*, and acts on *grid functions* rather than vectors. A stencil has the representation

$$[s_{\kappa_1\kappa_2}]_h = \begin{bmatrix} & \vdots & \vdots & \vdots & \\ \cdots & s_{-1,1} & s_{0,1} & s_{1,1} & \cdots \\ \cdots & s_{-1,0} & s_{0,0} & s_{1,0} & \cdots \\ \cdots & s_{-1,-1} & s_{0,-1} & s_{1,-1} & \cdots \\ & \vdots & \vdots & \vdots & \end{bmatrix}_h . \qquad s_{\kappa_1\kappa_2} \in \mathbb{R}$$

If $w_h : \Omega_h \longrightarrow \mathbb{R}$ is a grid function, the operator represented by the stencil is defined by

$$[s_{\kappa_1\kappa_2}]_h w_h(x,y) = \sum_{(\kappa_1,\kappa_2)} s_{\kappa_1\kappa_2} w_h(x + \kappa_1 h_x, y + \kappa_2 h_y)$$

To avoid confusion with the matrix notation, there is an alternative notation. For example, for a five-point and a nine-point operator the notation is

We mention before a point-operator, represented by a stencil may have to be modified for near boundary nodes. To solve a difference problem, it is not enough to specify a point operator. An implementation always commence by inscribing a grid into the domain of interest, then a point operator is chosen for each node. In our case we imose on $\overline{\Omega}$ a square grid $\Omega_{\Delta h}$, with an equal spacing of h in both spatial directions.
For an internal node the five point stencil yields

$$\frac{1}{h^2}\begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}_h u_{i,j} = f_{i,j}$$

For $(x_i, y_j) \in \Omega_h$ adjent to a boundary we have

$$\frac{1}{h^2}\begin{bmatrix} & -1 & \\ 0 & 4 & -1 \\ & -1 & \end{bmatrix}_h u_{i,j} = f_{i,j} + \frac{1}{h^2}\phi_{i-1,j}$$

For $(x_i, y_j) \in \Omega_h$ in a (here: the north-west) corner we have

$$\frac{1}{h^2}\begin{bmatrix} & 0 & \\ 0 & 4 & -1 \\ & -1 & \end{bmatrix}_h u_{i,j} = +f_{i,j}+\frac{1}{h^2}[\phi_{i-1,j}+\phi_{i,j-1}]$$

For this model problem, the Dirichlet BC are eliminated. For more complicated BC, or domains, *numerical boundary schemes* are preferred. To show

that the five-point formula is consistent, we define the local truncation error at a point $p \in \Omega_h$

$$\tau_p = \mathcal{L}_h u_p - f_p$$

Application of the the five point-formula to the Laplace equation gives the local truncation error

$$\tau_{ij} = \frac{u_{i,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}}{h^2} - f(x_i, y_j)$$

The expanison of the Taylor series around (x_i, y_j) gives

$$\tau_{i,j} = u_{xx} + u_{yy} - f + \frac{h^2}{12}(u_{4x} + u_{4y}) + \mathcal{O}(h^4),$$

thus the method is second order accurate. The method is also consistent since $\tau_{i,j} \overset{h\to 0}{\longrightarrow} 0$.

2.4.1 The Matrix Difference Equation

The five point operator for the poisson problem is

$$\delta_{xx}\mathbf{u} + \delta_{yy}\mathbf{u} = \frac{u_{i,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}}{h^2} \tag{2.23}$$

We seek the matrix operator. It is helpful to think of the unkonwns u_{ij} as occupying an N-by-N matrix U with entries u_{ij} and the right-hand sides $h^2 f_{ij}$ as similarly occupying an N-by-N matrix $h^2 F$. The trick is to write the matrix with (i, j) entry $4u_{ij} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1}$ in a simple way in terms of T_N and U. Since premultiplication with a matrix rearange rows, and postmultiplication rearange columns,

$$2u_{ij} - u_{i-1,j} - u_{i+1,j} = (T_N \cdot V)_{ij} \tag{2.24}$$
$$2u_{ij} - u_{i,j-1} - u_{i,j+1} = (V \cdot T_N)_{ij} \tag{2.25}$$

so adding these two equations yields

$$(T_N \cdot U + U \cdot T_N)_{ij} = 4u_{ij} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} = h^2 f_{ij} = (h^2 F)_{ij}$$

or

$$T_N \cdot V + V \cdot T_N = h^2 F \tag{2.26}$$

This is a matrix equation, but it is not in the usual '$Ax = b$' format. The matrix $T_N \cdot U + U \cdot T_N$ is a N-by-N matrix, so is the right hand side $h^2 F$. The eigenvalues and eigenvectors are the same as for the unerlying matrix

A, because '$Ax = \lambda x$' is equivalent to '$T_N \cdot U + U \cdot T_N = \lambda U$. If $T_N z_i = \lambda_i z_i$ and $T_N z_j = \lambda_j z_j$ are any two eigenpairs of T_N, and let $V = z_i z_j^T$. Then

$$\begin{aligned} T_N V + V T_N &= (T_N z_i) z_j^T + z_i (z_j^T T_N) \\ &= (\lambda_i z_i) z_j^T + z_i (z_j^T \lambda_j) \\ &= (\lambda_i + \lambda_j) z_i z_j^T \\ &= (\lambda_i + \lambda_j) V, \end{aligned} \tag{2.27}$$

so $V = z_i z_j^T$ is an eigenvector/eigenmatrix and $\lambda_i + \lambda_j$ is an eigenvalue. A second way to write the matrix operator is to write the unknowns $u_i j$ as a single long N^2-by-1 vector. This requires us to choose an order for them. The sparsity of the matrix operator depends on the ordering. For a column-wise or row-wise ordering of the nodes, also called *lexicographical* or *natural ordering* ordering, and with eliminated Dirichlet boundary conditions, the resulting matrix is block tridiagonal

$$T_{N\times N} = \frac{1}{h^2} \begin{bmatrix} T_N + 2I_N & -I_N & & \\ -I_N & \ddots & \ddots & \\ & \ddots & \ddots & -I_N \\ & & -I_N & T_N + 2I_N \end{bmatrix} \tag{2.28}$$

To see how this matrix came along, we consider the case $N = 3$. For $N = 3$, with row-wise ordering, the solution is a column 9-by-1 vector $\mathbf{u} = [u_1, u_2, ..., u_9]^T$ If we number f accordingly, we arive at the equation

$$\left[\begin{array}{ccc|ccc|ccc} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ \hline -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ \hline & & & -1 & & & 4 & -1 & \\ & & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{array}\right] \left[\begin{array}{c} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{array}\right] = \left[\begin{array}{c} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \\ f_9 \end{array}\right]$$

Each line of the matrix equation correspond to a point operator. The -1's imediately next to the diagonal corresponds to subtracting the left and right neighbors $-u_{i-1,j} - u_{i+1,j}$, The -1's $N = 3$ steps from the diagonal correspond to subtracting the top and bottom neighbors $-u_{i,j-1} - u_{i,j+1}$. The

elimination of the Dirichlet boundary conditions explains the rows in the difference matrix with only two or thre entries.
In the *red-black ordering* the red nodes are considered first, then the unknowns are at the black points. The difference matrix is now a block matrix with blocks $A_{rr}, A_{rb}, A_{br}, A_{bb}$. A_{rr} represent the connection of the red points

Figure 2.3: Lexicographic ordering Figure 2.4: red-black ordering

to the red points, A_{rb} the connection of the red points to the black points, A_{br} the conection of black points to red points, and A_{bb} the conection of black points to black points. With this notation the difference matrix reads

$$A_h = \begin{bmatrix} A_{rr} & A_{rb} \\ A_{br} & A_{bb} \end{bmatrix} \tag{2.29}$$

The five point formula for Δu gives diagonal matrices for the blocks A_{rr} and A_{bb}, with $4/h^2$ as diagonal elements. The blocd matrix $A_{rb} = A_{rb}^T$ is

$$A_{rb} = \frac{1}{h^2} \begin{bmatrix} -1 & 0 & -1 & & & & & \\ -1 & -1 & 0 & -1 & & & & \\ -1 & 0 & -1 & -1 & -1 & & & \\ & -1 & 0 & -1 & 0 & -1 & & \\ & & -1 & 0 & -1 & 0 & -1 & \\ & & & -1 & -1 & -1 & 0 & -1 \\ & & & & -1 & 0 & -1 & -1 \\ & & & & & -1 & 0 & -1 \end{bmatrix} \tag{2.30}$$

2.4.2 Expressing Poisson's Equation with Kronecker Products

The *Kronecker product*, sometimes called the tensor product, gives a systematic way to compute eigenvalues and eigenvectors of the Poisson matrix in any dimension. This survey relies on some new notation.

Definition 2.1 *Let X be m-by-n. Then* $\mathrm{vec}(X)$ *is defined to be a column vector of size $m \cdot n$ made of the columns of X stacked atop one another from left to right.*

Note that a N^2-by-1 vector $\mathbf{u}$ derived from a lexicographical ordering can be written as $\mathrm{vec}(V)$.

Definition 2.2 *Let A be an m-by-n matrix and B be a p-by-q. Then $A \otimes B$, the* Kronecker product *mA and B, is the $(m \cdot p)$-by-$(n \cdot q)$ matrix*

$$\begin{bmatrix} a_{1,1} \cdot B & \dots & a_{1,n} \cdot B \\ \vdots & & \vdots \\ a_{m,1} \cdot B & \dots & a_{m,n} \cdot B \end{bmatrix}$$

The following lemma tells us how to rewrite the Poisson equation in terms of Kronecker products and the *mathrmvec*$(\cdot)$ operator.

Lemma 2.2 *Let A be m-by-m, B be n-by-n, and X and C be m-by-n. Then the following properties holds*

1. $\mathrm{vec}(AX) = (I_n \otimes A) \cdot \mathrm{vec}(X)$.
2. $\mathrm{vec}(XB) = (B^T \otimes I_m) \cdot \mathrm{vec}(X)$.
3. *The Poisson equation $T_N V + V T_N = h^2 F$ is equivalent to*

$$T_{N\times N} \cdot \mathrm{vec(V)} := (I_N \otimes T_N + T_N \otimes I_N) \cdot \mathrm{vec}(V) = \mathrm{vec}(h^2 F). \qquad (2.31)$$

Proof We prove part 3. Vectorization of the Poisson matrix equation yields

$$\mathrm{vec}(T_N V + V T_N) = \mathrm{vec}(T_N V) + \mathrm{vec}(V T_N) = \mathrm{vec}(h^2 F).$$

By part 1 of the lemma

$$\mathrm{vec}(T_N V) = (I_N \otimes T_N)\mathrm{vec}(V).$$

By part 2 of the lemma and the symmetry of T_N,

$$\mathrm{vec}(V T_N) = (T_N^T \otimes I_N)\mathrm{vec}(V) = (T_N \otimes I_N))\mathrm{vec}(V).$$

Adding the last two expression completes the proof of part 3 □

In matrix notation the expression for $T_{N\times N}$ is

$$\begin{aligned} T_{N\times N} &= I_N \otimes T_N + T_N \otimes I_N \\ &= \begin{bmatrix} T_N & & & \\ & \ddots & & \\ & & \ddots & \\ & & & T_N \end{bmatrix} + \begin{bmatrix} 2I_N & -I_N & & \\ -I_N & \ddots & \ddots & \\ & \ddots & \ddots & -I_N \\ & & -I_N & 2I_N \end{bmatrix} \end{aligned}$$

The computation of the eigenvalues and eigenvctors for the Poisson matrix can be done in several ways. A general and convenient way uses the properties of Kronecker products.

Lemma 2.3 *The following facts about the Kronecker product hold:*

1. *Assume that the products $A \cdot C$ and $B \cdot D$ are well defined. Then $(A \otimes B) \cdot (C \otimes D) = (A \cdot C) \otimes (B \cdot D)$.*

2. *If A and B are invertible, then $(A \otimes B)^{-1} \otimes B^{-1}$.*

3. *$(A \otimes B)^T = A^T \otimes B^T$.*

Proposition 2.1 *Let $T_N = Z\Lambda Z^T$ be the eigendecomposition of T_N, with $Z = [z_1, ..., z_N]$ the orthogonal matrix whose columns are eigenvectors, and $\Lambda = \text{diag}(\lambda_1, ..., \lambda_N)$. Then the eigendecomposition of $T_{N\times N} = I \otimes T_N + T_N \otimes I$ is*

$$I \otimes T_N + T_N \otimes I = (Z \otimes Z) \cdot (I \otimes \Lambda + \Lambda \otimes I) \cdot (Z \otimes Z)^T. \tag{2.32}$$

$I \otimes \Lambda + \Lambda \otimes I$ is a diagonal matrix whose $(iN+j)$th diagonal entry, the (i,j)th eigenvalue of $T_{N\times N}$, is $\lambda_{i,j} = \lambda_i + \lambda_j$. $Z \otimes Z$ is an orthogonal matrix whose $(iN + j)$th column, the corresponding eigenvector, is $z_i \otimes z_j$.

Proof From part 1 and 3 of Lemma 2.3 it is easy to verify that $Z \otimes Z)$ is orthogonal since $(Z \otimes Z)(Z \otimes Z)^T = (Z \otimes Z)(Z^T \otimes Z^T) = (Z \cdot Z^T) \otimes (Z \cdot Z^T) = I \otimes I = I$. We can now verify (2.32):

$$\begin{aligned}
(Z \otimes Z) \cdot (I \otimes \Lambda + \Lambda \otimes I) \cdot (Z \otimes Z)^T & \\
&= (Z \otimes Z) \cdot (I \otimes \Lambda + \Lambda \otimes I) \cdot (Z^T \otimes Z^T) \\
&= (Z \cdot I \cdot Z^T) \otimes (Z \cdot \Lambda \cdot Z^T) + (Z \cdot \Lambda \cdot Z^T) \otimes (Z \cdot I \cdot Z^T) \\
&= I \otimes T_N + T_N \otimes I = T_{N\times N}
\end{aligned}$$

From the definition of the Kronecker product, it is easy to verify that $I \otimes \Lambda + \Lambda \otimes I$ is diagonal, with $(iN + j$ entry given by $\lambda_j + \lambda_i$, and that the corresponding eigenvector is the $iN + j$ column of $Z \otimes Z$. □

2.4.3 Implementation

It is always good practice to run computer progams to get a better 'feel' for what we are trying to say in our mathematical formulations. In this spirit

we apply the five-point operator (2.23) to the Dirichlet problem on the unit square

$$\Delta u = (1 - \frac{\pi}{4}) \sin\left(\frac{\pi x}{2}\right) \exp(y), \qquad 0 \leq x, y \leq 1$$

subjec to the boundary conditions

$$\begin{aligned} u(x,0) &= \sin\left(\frac{\pi x}{2}\right) \qquad u(x,1) = \sin\left(\frac{\pi x}{2}\right) \exp(1) \qquad 0 \leq x \leq y \\ u(0,y) &= 0 \qquad\qquad\qquad u(1,y) = \exp(y) \qquad\qquad\qquad 0 \leq y \leq 1 \end{aligned}$$

Figure 2.5: The solution of the Dirichlet problem and the Numerical error

2.4.4 Eigenvalues and Eigenvectors of the five-point Operator

It is instructive to see how the eigenvalues and eigenvectors of $T_N \times N$ are related to the eigenvalues and eigenfunctions of the Laplace operator Δ in the unit square. The function v is said to be an eigefunction of Δ in a domain Ω if

$$\begin{aligned} \Delta v &= \lambda v \qquad x, y \in \Omega \\ v &= 0 \qquad x, y \in \partial\Omega \end{aligned}$$

```
% Solution of the Poisson equation on the unit
% square. Use row wise ordering of vectors
N=30; h=1/(N+1); x=[h:h:1-h]'; y=x;
% Boundary conditions
gb=sin(0.5*pi*x); gt=sin(0.5*pi*x)*exp(1);
gl=zeros(N,1); gr=exp(y);
% The Poisson matrix
TN=2*eye(N)-diag(ones(N-1,1),1)-diag(ones(N-1,1),-1);
TNxN=kron(eye(N),TN)+kron(TN,eye(N));
A=-gallery('poisson',N);
% Form rhs
f=zeros(N*N,1);
for i=1:N
    for j=1:N
        f(i+N*(j-1))=sin(0.5*pi*x(i))*exp(y(j));
    end
end
f=h^2*(1-0.25*pi^2)*f;
% Include boundary conditions
for i=1:N
    f(i)=f(i)-gb(i); f((N-1)*N+i)=f((N-1)*N+i)-gt(i);
end
for j=1:N
    f((j-1)*N+1)=f((j-1)*N+1)-gl(j); f(j*N)=f(j*N)-gr(j);
end
% Find the solution
U=-1*TNxN\f; U=reshape(U,N,N); U=U';
% Add the boundary values
x=[0;x;1]; y=[0;y;1]; Unew=zeros(N+2,N+2);
Unew(2:N+1,2:N+1)=U; Unew(1,2:N+1)=gb'; Unew(N+2,2:N+1)=gt';
Unew(1:N+2,N+2)=exp(y); [x,y]=meshgrid(x,y);
% Plot results
figure(1); mesh(x,y,Unew,'EdgeColor','black');
title('Numerical Solution'); xlabel('x'); ylabel('y');
figure(2); Uexact=sin(0.5*pi*x).*exp(y);
mesh(x,y,abs(Uexact-Unew),'EdgeColor','black');
title('error'); xlabel('x'); ylabel('y');
format short e; error=max(max(abs(Uexact-Unew)));
```

It is easy to verify that the function $v(x, y) = sin(\alpha\pi x)sin(\beta\pi x),\ x, y \in [0, 1]$ satisfy $\Delta v = -(\alpha^2 + +\beta^2)\pi^2 v$ Thus v is an eigenfunction, and $\lambda = -(\alpha + \beta)$ is the corresponding eigenvalue. It can be proved that all eigenfunctions of Δ in $(0, 1)^2$ have this form.

Figure 2.6: Three-dimensional and contour plots of the first four eigenvectors of the 10-by-10 Poisson equation

The eigenvalues and eigenvectors of the five-point operator is given by Proposition 1.1. The eigenvalues are

$$\begin{aligned}\lambda_{\alpha,\beta} &= (\lambda_\alpha + \lambda_\beta) \qquad \alpha,\beta = 1,2,...,N \\ &= 2\left\{1-\cos\left(\frac{\pi\alpha}{N+1}\right)\right\} + 2\left\{1-\cos\left(\frac{\pi\beta}{N+1}\right)\right\} \\ &= -4\left\{\sin^2\left[\frac{\alpha\pi}{2(N+1)}\right] + \sin^2\left[\frac{\beta\pi}{2(N+1)}\right]\right\}, \end{aligned} \tag{2.33}$$

and the eigenvectors are

$$v_{k,l} = \sin\left(\frac{k\alpha\pi}{N+1}\right)\sin\left(\frac{l\beta\pi}{N+1}\right), \qquad k,l = 0,1,...,N+1 \tag{2.34}$$

The eigenvectors of the discrete operator is equal to the eigenfuntions evaluated at the points $\left\{\left(\frac{k}{N+1},\frac{l}{k+1}\right)\right\}_{k,l=0,1,...,m+1}$ (for $\alpha,\beta = 1,2,...,m$ only, the matrix $T_{N\times N}$, unlike Δ, is finite-dimensional!). If we expand the eigenvalues

in a power series and bearing in mind that $(N+1)\Delta x = 1$, we readily obtain

$$\begin{aligned}\frac{\lambda_{\alpha,\beta}}{\Delta x^2} &= -4\Bigg(\left\{\left[\frac{\alpha\pi}{2(N+1)}\right]^2 - \frac{1}{3}\left[\frac{\alpha\pi}{2(N+1)}\right]^4 + ...\right\} \\ &\quad + \left\{\left[\frac{\beta\pi}{2(N+1)}\right]^2 - \frac{1}{3}\left[\frac{\beta\pi}{2(N+1)}\right]^4 + ...\right\}\Bigg) \\ &= -(\alpha^2+\beta^2)\pi^2 + \frac{1}{12}(\alpha^4+\beta^4)\pi^4(\Delta x)^2 + \mathcal{O}((\Delta x^4)).\end{aligned}$$

Thus we see that $(\Delta x^2)\lambda_{\alpha,\beta}$ is a good approximation to $-(\alpha^2+\beta^2)\pi$ provided α and β are small in comparison with m.

2.5 Higher-order methods for $Deltau = f$

The Labplace operator Δ is an elliptic operator, but it is important to evolutionary problems as well. To mention just two, it plays a key role in the parabolic *diffusion* equation

$$\frac{\partial u}{\partial t} = \Delta u, \qquad u = u(x,y,t),$$

and the hyperbolic *wave* equation

$$\frac{\partial^2 u}{\partial t^2} = \Delta u, \qquad u = u(x,y,t).$$

The importance of the Laplace operator motivates a analysis of higher-order schemes. We commence by defining the following point-operators

the *shift* operator, $(\mathcal{E}\mathbf{z})_k = z_{k+1}$;

the *forward difference* operator, $(\Delta_+\mathbf{z})_k = z_{k+1} - z_k$;

the *backward difference* operator, $(\Delta_-\mathbf{z})_k = z_k - z_{k-1}$;

the *central difference* operator, $(\Delta_0\mathbf{z})_k = z_{k+\frac{1}{2}} - z_{k-\frac{1}{2}}$;

The last operation is not a operator when it acts alone since grid functions are defined only at grid points. The sequence $\mathbf{z} = \{z_k\}_{-\infty}^{\infty}$ orginates in the sampling of a function z at equispaced points h. Futher, we define

the *differntial* operator $(\mathcal{D}\mathbf{z})_k = z'(kh)$.

It can be verified that all these operators are linear, that is, given that

$$\mathcal{T} \in \{\mathcal{E}, \Delta_+, \Delta_-, \Delta_0, \mathcal{D}\}$$

and that $\mathbf{w}, \mathbf{z} \in \mathbb{R}^{\mathbb{Z}}, a, b \in \mathbb{R}$, it is true that

$$\mathcal{T}(a\mathbf{w} + b\mathbf{z}) = a\mathcal{T} + b\mathcal{T}$$

The differnce operators are related, they are all functions of the shift operator $\mathcal{E}$. Note that $\Delta_+ = (\mathcal{E} - \mathcal{I})$ and $\Delta_- = (\mathcal{I} - \mathcal{E}^{-1})$ where $\mathcal{I}$ is the identity. If we interprete $\mathcal{E}^{1/2}$ as a 'half shift', then $\mathcal{E}^{-1/2} + \mathcal{E}^{1/2}$. Finally, to express $\mathcal{D}$ in terms of $\mathcal{E}$ we do the following Taylor expansion

$$\mathcal{E}z(x) = z(x+h) = \sum_{j=0}^{\infty} \frac{1}{j!} \left[\frac{d^j z(x)}{dx^j} \right] h^j = \left[\sum_{j=0}^{\infty} \frac{1}{j!} (h\mathcal{D})^j \right] z(x) = e^{h\mathcal{D}} z(x),$$

and we deduce $\mathcal{E} = e^{h\mathcal{D}}$. Formal inversion yields

$$h\mathcal{D} = \ln \mathcal{E} \tag{2.35}$$

Because all operators are functions of $\mathcal{E}$ they commute. In consequence we need not bother with the order of their action whenever they are superposed. Let us express the differential operator $\mathcal{D}$ in terms of the other point operators. The relation $\mathcal{E} = \mathcal{I} + \Delta_+ = (\mathcal{I} - \Delta_-)^{-1}$ is obvious. The expression $\Delta_0 z_k = \mathcal{E}^{1/2} z_k - \mathcal{E}^{-1/2}$ yields the quadratic quadratic relation

$$(\mathcal{E}^{1/2})^2 - \Delta_0 \mathcal{E}^{1/2} - \mathcal{I} = 0$$

with two solutions, $\frac{1}{2}\Delta_0 \pm \sqrt{\frac{1}{4}\Delta_0^2 + \mathcal{I}}$. Since $\mathcal{E}z(x) = z(x+h)$ it follows that $(\mathcal{E} - \mathcal{I})z(x) = \mathcal{O}(h)$. From this we deduce that the correct formula is

$$\mathcal{E} = \left(\frac{1}{2}\Delta_0 + \sqrt{\mathcal{I} + \frac{1}{4}\Delta_0^2} \right)^2$$

From these expressions, and equation (2.35), the following relations hold

$$h\mathcal{D} = \ln(\mathcal{I} + \Delta_+) \tag{2.36}$$

$$h\mathcal{D} = -\ln(\mathcal{I} + \Delta_-) \tag{2.37}$$

$$h\mathcal{D} = 2\ln\left(\Delta_0 + \sqrt{\mathcal{I} + \frac{1}{4}\Delta_0^2} \right). \tag{2.38}$$

These relations allow us to approximate the differential operator $\mathcal{D}$ and it powers (which correspond to higher derivatives). For example, expanding (2.36) we obtain

$$\begin{aligned} \mathcal{D} &= \frac{1}{h}\ln(\mathcal{I} + \Delta_+) = \frac{1}{h}[\Delta_+ - \frac{1}{2}\Delta_+^2 + \frac{1}{3}\Delta_+^3 + \mathcal{O}(\Delta_+^4)] \\ &= \frac{1}{h}(\Delta_+ + \frac{1}{2}\Delta_+^2 + \frac{1}{3}\Delta_+^3 + \mathcal{O}(h^3)), \qquad h \to 0. \end{aligned}$$

To get an expression for the that approximates $d^s z(kh)/dx^s$, we operate s times

$$\mathcal{D}^s = \frac{1}{h^s}[\Delta_+^s - \frac{1}{2}\Delta_+^{s+1} + \frac{1}{24}s(3s+5)\Delta_+^{s+2}] + \mathcal{O}(h^3)$$

Similarly to (2.5 we can use (2.37) to approximate the differential operator using only grid points wholly to the left,

$$\mathcal{D}^s = \frac{(-1)^s}{h^s}[ln(\mathcal{I} + \Delta_-)]^s = \frac{1}{h^s}[\Delta_-^s - \frac{1}{2}\Delta_-^{s+1} + \frac{1}{24}s(3s+5)\Delta_-^{s+2}] + \mathcal{O}(h^3)$$

Central difference operators usually leads to more tractable linear systems because the grid points used in the approximation are close to the central node, hence reducing the bandwidth of the matrix operator. To writ the differential operator $\mathcal{D}$ in terms of the central difference operator Δ_0, we make use of Taylor series expansion of the function $g(\xi) := \ln(\xi + \sqrt{1+\xi^2}$. By the generalized binomial theorem,

$$g'(\xi) = \frac{1}{\sqrt{1+\xi^2}} = \sum_{j=0}^{\infty}(-1)^j \binom{2j}{j}(\frac{1}{2}\xi)^{2j},$$

where $\binom{2j}{j}$ is the binomial coefficient equal to $(2j)!/(j!)^2$. Since $g(0) = 0$ and the Taylor series converges uniformly for $|\xi| < 1$, integration yields

$$g(\xi) = g(0) + \int_0^{xi} g'(\tau)d\tau = 2\sum_{j=0}^{\infty}\frac{(-1)^j}{2j+1}\binom{2j}{j}(\frac{1}{4}\Delta_0)^{2j+1}.$$

Letting $\xi = \frac{1}{2}\Delta_0$, we thus deduce from (2.38) the formal expansion

$$\mathcal{D} = \frac{2}{h}g(\frac{1}{2}\Delta_0) = \frac{4}{h}\sum_{j=0}^{\infty}\frac{(-1)^j}{2j+1}\binom{2j}{j}(\frac{1}{4}\Delta_0)^{2j+1}. \tag{2.39}$$

Unfortunately odd powers of Δ_0 are undefined. Even powers of are defined because $\Delta_0^{2s} = (\Delta_0^2)^s = (z_{n-1} - 2z_n + z_{n+1})^s$. Thus raising (2.39) to an even power yields

$$\mathcal{D}^{2s} = \frac{1}{h^{2s}}[(\Delta_0^2)^s - \frac{1}{12}(\Delta_0^2)^{2+1} + \frac{1}{45}(\Delta_0^2)^{s+2}] + \mathcal{O}(h^6)$$

With the differential operator approximations available, we start our discussion of higher order methods. A popular approximation to $\Delta u = f$ at the (k,l)th grid point is given by the *nine-point formula*

$$\frac{1}{(\Delta x)^2}(\Delta_{0,x}^2 + \Delta_{0,y}^2 + \frac{1}{6}\Delta_{0,x}^2\Delta_{0,y}^2)u_{k,l} = f_{k,l} \tag{2.40}$$

The computational stencil for the nine-point operator is

Let us analyse the error. Recall that $\Delta_0 = \mathcal{E}^{1/2} - \mathcal{E}^{-1/2}$ and that $\mathcal{E} = e^{\Delta x \mathrm{D}}$. Thus the Taylor expansion of Δ_0 is

$$\begin{aligned} \Delta_0^2 &= \mathcal{E} - 2\mathcal{I} + \mathcal{E}^{-1} = \mathrm{e}^{\Delta x \mathcal{D}} - 2\mathcal{I} + \mathrm{e}^{-\Delta x \mathcal{D}} \\ &= h^2\mathcal{D}^2 + \frac{1}{12}(\Delta x)^4\mathcal{D}^4 + \mathcal{O}((\Delta x)^6). \end{aligned}$$

Substituting this into (2.40) yields

$$\begin{aligned} &\frac{1}{(\Delta x)^2}(\Delta_{0,x}^2 + \Delta_{0,y}^2 + \frac{1}{6}\Delta_0, x^2\Delta_0, y^2) \\ &= (\mathcal{D}_x^2 + \mathcal{D}_y^2) + \frac{1}{12}(\Delta x)^2(\mathcal{D}_x^2 + \mathcal{D}_y^2)^2 + \mathcal{O}((\Delta x)^4) \\ &= \Delta + \frac{1}{12}(\Delta x)^2\Delta^2 + \mathcal{O}((\Delta x)^4) \end{aligned} \tag{2.41}$$

It follows that the nine-point operator is an approximation of order $\mathcal{O}((\Delta x)^4)$ to the equation

$$[\Delta + \frac{1}{12}(\Delta x)^2\Delta^2]u = f \tag{2.42}$$

In the case $f = 0$, corresponding to the Laplace equation, the nine point formula bears an error $\mathcal{O}(\Delta x)^4)$. This follows from (2.41) and the identity $\Delta u = \Delta^2 u = 0$. Let us define the operator $\mathcal{M}_{\Delta x} := \mathcal{I} + \frac{1}{12}(\Delta x)^2\Delta$. Suppose that $\Delta x > 0$ is small enough, so that $\mathcal{M}^{-1}$ exists, and act with this opearator on both sides of (2.42). A new Poisson equation arise for which the nine-point formula produces an error proportional to Δx^4

$$\Delta u = \mathcal{M}_{\Delta x}^{-1} f,$$

The trick now is to replace f in the original equation with the modified function $\tilde{f}$

$$\begin{aligned} \tilde{f}(x,y) &= f(x,y) + \frac{1}{12}\Delta f(x,y) + \mathcal{O}((\Delta x)^4) \\ &= [\mathcal{I} + \frac{1}{12}\Delta]f + \mathcal{O}((\Delta x)^4). \end{aligned}$$

Thus $\mathcal{M}^{-1}\tilde{f} = f + \mathcal{O}((\Delta x)^4)$. In other words, the nine-point operator, when applied to $\Delta^2 u = \tilde{f}$, yields an $\mathcal{O}((\Delta x)^4)$ approximation to ther original Poisson equation $\Delta u = f$ with the same boundary conditions. We can approximate the function $\tilde{f}$ as follows

$$\tilde{f} = [\mathcal{I} + \frac{1}{12}(\Delta_{0,x}^2 + \Delta_{0,y}^2)]f_{k,l}$$

which we also can write as

This method is reffered to as a fourth order method in [1]. [1]

[1] Im not sure this is correct since $[\mathcal{I}+\frac{1}{12}(\Delta_{0,x}^2+\Delta_{0,y}^2)]f = f(x,y)+\Delta f(x,y)+\mathcal{O}((\Delta x)^2)$. For the method to be fourth order accurate the approximation of Δf should be fourth order accurate. The method is however fourth order accurate if we know Δf

Chapter 3

Evolutionary Equations

It is often useful to clasify partial differential equations into *steady-state* and *evolutionary* equations. The Poisson equation is an example of a stationary problem. Evolutionary equations, however, model systems that undergo change, such as wave dynamics, diffusion, and other transport phenomena. Linear PDEs are traditionally classified as *elliptic*, *parabolic*, or *hyperbolic* equations. We remark that elliptic equations are of the steady-state type, while parabolic and hyperbolic PDEs are evoultionary. Hyperbolic equations are *coservative* in that the 'energy' of the system is conserved over time. They are analogous to a system of ODEs whose matrix has purely imaginary eigenvalues, yielding a oscillatory solution that neither grows or decay with time. Parabolic PDEs are *dissipative* in that the 'energy' of the solution diminishes over time. Parabolic PDEs are analogous to linear systems of ODEs whose matrix has only eigenvalues with negativel real parts, yielding an exponentially decaying solution. Another characteristic feature that differe is the propogation speed. Hyperbolic PDEs propogate information at finite speed (wawe-solutions), whereas parabolic PDEs propagate information instantaneously (but the information decays exponentially). The differences between the types of PDEs has important theroretical and practical implications. For example, for a hyperbolic problem, it is desired that the numerical scheme is non-disspiative. But it is not enough to ensure a non-dissipativive scheme, the phase error is just as inportant. Hyperbolic can be wery difficult, especialy the treatment of the nonlinear phenomena that can occure (shock and rarefecation waves). Since hyperbolic problems are conservative, they are, in principle, reversible in time. Parabolic PDEs has a smoothing effect. Even nonsmooth initial conditions becomes smooth with time. It is not possible to determine the history of a parabolic equation from its current state. That is, you cannot determine the initial temperature from its current temperature distribution by integrating the heat equation backwards in time, that is, the

heat equation integrated backward in time is ill-posed. The challanges in solving parabolic or hyperbolic PDEs numerically are analogous to those in solving ODEs that are stiff because of eigenvalues with large negative real parts (parabolic) or large imaginary parts (hyperbolic).
A linear parabolic equation takes the form

$$\sigma \frac{\partial u}{\partial t} = \mathcal{L}u \tag{3.1}$$

where $\mathcal{L}$ is a elliptic operator. To solve the problem we must be provided with initial and boundary values.

3.1 The Diffusion Equation

Our model problem for a parabolic equatoin is the *heat equation* in one, or two, dimensions. It is also known as the *diffusion equation*. In one dimension it is given by

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \qquad 0 \leq x \leq 1, \quad t \geq 0 \tag{3.2}$$

The accompanying boundary conditions are,

$$u(0,t) = \phi_0(t), \qquad u(1,t) = \phi_1(t), \qquad t \geq 0 \tag{3.3}$$

and the initial condition is

$$u(x,0) = g(x), \qquad 0 \leq x \leq 1 \tag{3.4}$$

Our quest is to approximate the heat equation on the strip

$$\{(x,t) : \ x \in [0,1], \ t \geq 0\}$$

We let $\Delta h = 1/(1+N)$ be the steplength in the x-direction and k the steplength in the t-direction such that

$$\begin{aligned} t_n &= n \cdot k, \qquad n = 0,1,2,... \\ x_i &= i \cdot h, \qquad i = 0,1,...,N+1 \end{aligned}$$

The approximation of $u(i \cdot h, n \cdot k)$ is denoted by u_i^n. Replaceing the second derivative with the second-order point operator δ_{xx} and the time derivative with the forward difference results in

$$\frac{u_i^{n+1} - u_i^n}{k} = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2}$$

rearanging yields

$$u_i^{n+1} = u_i^n + r(u_{i+1}^n - 2u_i^n + u_{i-1}^n), \qquad i = 1, ..., N, \quad k = 0, 1, ...M. \tag{3.5}$$

where the ration

$$r = \frac{k}{h^2}$$

is the *Courant number*. The recursive relation (3.5) is refered to as the *explicit Euler* method. There are two important considerations when choosing a time marching method. Stability and accuracy. We consider the numerical accuracy now, and stability later on.
The local truncation error is derived by inserting the exact solution into the approximation scheme

$$\begin{aligned} \tau_i^n &= \frac{u(x,t+k) - u(x,t)}{k} - \frac{u(x-h,t) - 2u(x,t) + u(x+h,t)}{h^2} \\ &= \frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} + \frac{1}{2}k\frac{\partial^2 u}{\partial t^2} + \frac{1}{12}h^2\frac{\partial^4 u}{\partial x^4} + \mathcal{O}(k^2 + h^4) \\ &= \frac{1}{2}k\frac{\partial^2 u}{\partial t^2} + \frac{1}{12}h^2\frac{\partial^4 u}{\partial x^4} + \mathcal{O}(k^2 + h^4) \end{aligned}$$

Since $\tau_i^n = \mathcal{O}(h^2) + \mathcal{O}(k)$ we say that the method is of order $(2, 1)$. The method is consistent since

$$\tau_i^n \overset{h,k\to 0}{\longrightarrow} 0$$

Let (ih, nk) be a grid-node and $\mathbb{G} = \{(ih, nk) : i = 0, ..., N+1 \; k = 0, ...M\}$ be a computational grid. We refer to a method as convergent if,

$$u_i^n \overset{h,k\to 0}{\longrightarrow} u(ih, nk) \qquad (ih, kn) \in \mathbb{G}$$

and we say that $u_i^n \overset{h,k\to 0}{\longrightarrow} u(ih, nk)$ for all $(ih, kn) \in \mathbb{G}$ if

$$\max_{0\le n\le M} \max_{0\le i\le N+1} |e_i^n| \overset{h,k\to 0}{\longrightarrow} 0,$$

where $e_i^n = u(ih, nk) - u_i^n$ is the global error. Let $\tilde{u}_i^n = u(ih, nk)$. We have that

$$\begin{aligned} u_i^{n+1} &= u_i^n + r(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \\ \tilde{u}_i^{n+1} &= \tilde{u}_i^n + r(\tilde{u}_{i+1}^n - 2\tilde{u}_i^n + \tilde{u}_{i-1}^n) + k\tau_i^n \end{aligned}$$

such that

$$\begin{aligned} e_i^{n+1} &= (1-2r)e_i^n + re_{i+1}^n + re_{i-1}^n + k\tau_i^n \\ &\quad \text{with} \quad e_i^0 = 0, \; e_0^n = e_N + 1^n = 0. \end{aligned}$$

Assume that $r \leq \frac{1}{2}$, and that $|\tau_i^n| \leq C(k+h^2)$. Therefore, by the triangle inequality

$$\begin{aligned} |e_i^{n+1}| &\leq (1-2r)|e_i^n| + r(|e_{i+1}^n| + |e_{i-1}^n|) + C(k^2+kh^2) \\ &\leq \max_{i=0,\ldots,N+1} |e_i^n| + C(k^2+kh^2) \end{aligned}$$

Let $E^n = \max_{i=0,\ldots,N+1}$ such that

$$\begin{aligned} E^{n+1} &\leq E^n + C(k^2+kh^2) \\ &\leq E^{n-1} + 2A(k^2+kh^2) \\ &\leq \ldots \leq E^0 + (N+1)C(k^2+kh^2) \end{aligned}$$

Hence $E^n \leq nkC(k+h^2) = t_nC(k+h^2)$, and we can conclude that the solution of the Euler scheme converges to the true solution if $r \leq 1/2$.
The restriction on r is painfull. It means that,each time we refine h, we need to amend k such that the ratio $r = k/h^2$ remains constant, which means that k is likely to be considerably smaller than h. For example, if $N = 19$ and $r = \frac{1}{2}$, then $h = 1/20$ and $k = 1/800$, leading to a very large computational cost.
There are methods that converge for all $r \geq 0$. The *implicit Euler method* is obtained if we use the the backward difference to approximate the time derivative, and the thre-point δ_{xx} at $(ih, (n+1)k)$

$$\frac{u_i^{n+1} - u_i^n}{k} = \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}n + 1}{h^2}$$

or

$$-ru_{i+1}^{n+1} + (1+2r)u_i^{n+1} - ru_{i-1}^{n+1} = u_i^n, \qquad i = 1, ..., N \tag{3.6}$$

with $u_0^{n+1} = \phi_0(k*(n+1))$ and $u_{N+1}^{n+1} = \phi_1(k*(n+1))$. The implicit Euler method is like its relative, the explicit Euler method, only first order accurate in time. A method that is second order accurate in time, and convergent for all $r \geq 0$ is the *Crank-Niclolson method*

$$\frac{u_i^{n+1} - u_i^n}{k} = \frac{1}{2}\left[\frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{h^2} + \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^{n+1}}{h^2}\right]$$

An alternative expression is

$$\frac{1}{2}ru_{i+1}^{n+1} + (1+r)u_i^{n+1} - \frac{1}{2}ru_{i-1}^{n+1} = \frac{1}{2}ru_{i+1}^n + (1-r)u_i^n + \frac{1}{2}ru_{i-1}^n \qquad i = 1, ..., N \tag{3.7}$$

We mention that all the time-marching methods mentioned so far are special cases of the *θ-method*

$$\frac{u_i^{n+1}-u_i^n}{k}=\theta\frac{u_{i+1}^{n+1}-2u_i^{n+1}+u_{i-1}^{n+1}}{h^2}+(1-\theta)\frac{u_{i+1}^{n}-2u_i^{n}+u_{i-1}^{n}}{h^2}$$

for $0 \leq \theta \leq 1$.

3.1.1 The Thomas Algorithm

When we approximate the time-derivative with a implicit method, we have to solve a linear system of equation at each time step. Why then bother to use a implicit method, when a explicit method is available, with the same accuracy. The answer is stability. One might have to use very small time steps for a explicit method to converge. This fact was demonstrated with the explicit Euler method. Due to their superior stability properties, the restriction on the time step is determined by accuracy alone.
Consider the Crank-Nicolson mehtod applied to the diffusion equation

$$\frac{1}{2}ru_{i+1}^{n+1}+(1+r)u_i^{n+1}-\frac{1}{2}ru_{i-1}^{n+1}=\frac{1}{2}ru_{i+1}^{n}+(1-r)u_i^{n}+\frac{1}{2}ru_{i-1}^{n}, \qquad i=1,...,N.$$

with the Dirichlet boundary conditions $u_0^{n+1}=u_{N+1}^{n+1}=0$. This system of equations can be rewritten as

$$\begin{bmatrix} a_0 & a_1 & & \\ a_{-1} & \ddots & \ddots & \\ & \ddots & \ddots & a_1 \\ & & a_{-1} & a_0 \end{bmatrix} \cdot \begin{bmatrix} u_1^{n+1} \\ \vdots \\ \vdots \\ u_N^{n+1} \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ \vdots \\ b_N \end{bmatrix}$$

Thus we have to solve a tridiagonal $B(a_{-1},a_0,a_1)=\mathbf{f}$ at each time step. This can be done with a tridiagonal solver. The matrices L and U of the LU factorization of a general tridiagonal matrix $B(\mathbf{a},\mathbf{b},\mathbf{c}$ are bidiagonal matrices of the form

$$L=\begin{bmatrix} 1 & & & \\ \beta_2 & 1 & & \\ & \ddots & \ddots & \\ & & \beta_n & 1 \end{bmatrix} \quad U=\begin{bmatrix} \alpha_1 & c_1 & & \\ & \alpha_2 & \ddots & \\ & & \ddots & c_{n-1} \\ & & & \alpha_n \end{bmatrix}$$

The coefficients α_i and β_i can be computed from the relation

$$\alpha_1=a_1, \quad \beta_i=\frac{b_i}{\alpha_{i-1}}, \quad \alpha_i=a_i-\beta_i c_{i-1}, \ i=2,...,n.$$

```
 % Solve the 1D heat equation with the Crank-Nicolson method
alpha=1; N=48; L=1; h=L/(N+1); u=zeros(N+2,1); unew=zeros(N+2,1);
u_init=u; x=(0:h:(N+1)*h)'-L/2; u=exp(-5*x.^2); tend=0.5;
k=alpha*h^2; nsteps=ceil(tend/k); nplots=25; iplot=1;
plotsteps=ceil(nsteps/nplots);
% Time stepping
a=(alpha/2)*ones(N+2,1); b=(1+alpha)*ones(N+2,1); c=a;
a(1)=0; c(N+2)=0; B=spdiags([-a,b,-a],[-1,0,1],N+2,N+2);
for i=1:nsteps
    f=(alpha/2)*u(1:N)+(1-alpha)*u(2:N+1)+...
        (alpha/2)*u(3:N+2);
    f1=(1-alpha)*u(1)+(alpha/2)*u(2);
    fNp2=(alpha/2)*u(N+1)+(1-alpha)*u(N+2); f=[f1;f;fNp2];
    unew=B\f; u=unew;
    if(rem(i,plotsteps)<1)
        tempplot(:,iplot)=u(:); tplot(iplot)=i*k;
        iplot=iplot+1;
    end
    end
figure(1);mesh(x,tplot,tempplot','EdgeColor','black');
xlabel('x'); ylabel('t'); zlabel('Temperature');
title('Heat conduction with Dirichlet bc');
```

To solve the system $B\mathbf{u} = LU\mathbf{u} = \mathbf{f}$ we solve $L\mathbf{y} = \mathbf{f}$ by forward elimination, then $Ux = y$ by backward substitution. This direct method is known as the *Thomas algorithm*. For the solution to be well posed we require that B is *diagonally dominant*, that is, we require that $b_j > a_j + c_j$. This is clearly the case here since $(1 + r) > r/2 + r/2$. Lets do another computer experiment. Let us solve the heat equation with Dirichlet boundary conditions using the Crank-Nicolson method.

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{\partial^2 u}{\partial x^2}, \qquad -0.5 \le x \le 0.5 \\ u(x,0) &= \exp(-5x^2) \end{aligned}$$

Figure 3.1: Model problem solved with the Crank-Nicolson method

3.2 Semi-Discrtization

Given a initial/boundary value problem

$$\begin{aligned} \frac{\partial u}{\partial t} &= \mathcal{L}u, \qquad 0 \le x \le 1 \\ u(x,0) &= g(x), \quad u(0,t) = \phi_0(t), \quad u(1,t) = \phi_1(t) \end{aligned}$$

Let $v_i(t) \approx u(x_i, t),\ i = 1, ..., N$, and $v_0(t) = \phi_0(t),\ v_{N+1} = \phi_1(t)$, then a semi-discretization reduces our PDE to the following system of ODEs

$$\begin{aligned} \frac{dv_i}{dt} &= \mathcal{L}v_i \qquad i = 1, 2, ..., N \\ v_i(0) &= g(x_i) \end{aligned}$$

For example, a semidiscretization of the heat equation is given by

$$\frac{dv_i}{dt} = \frac{1}{h^2}(v_{i+1} - 2v_i + v_{i-1}), \qquad i = 1, ..., N \tag{3.8}$$

where we have replaced $\mathcal{L}u(ih, t)$ with the point operator approximation $(\delta_{xx}v)_i$. We can solve (3.8) using an ODE solver. If the PDE is parabolic,

the resulting sytem is particulary stiff, so we need a method for solving stiff equations. If we use the trapezoid [1] rule to evaluate the the time derivative we arive at

$$u_i^{n+1} = u_i^n + \frac{k}{2h^2}(u_{i+1}^{n+1} - 2u_i^{n+1} - u_{i-1}^{n+1}) + \frac{k}{2h^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$$

which we recognize as the Crank-Nicolson method. If we have a Neumann boundary condition, we also have to solve a differential equation at the boundary node.

3.2.1 Stability

let $\mathbf{u}^n = [u_1^n, u_2^n, ..., u_N^n]$ be the numerical solution at time kn For a *two-level* scheme we can write the difference approximation as

$$A\mathbf{u}^{n+1} = B\mathbf{u}^n + \mathbf{c}^n \tag{3.9}$$

where $\mathbf{c}^n$ contains the source function, and known boundary values. This scheme can also be written as

$$\begin{aligned} \mathbf{u}^{n+1} &= Q\mathbf{u}^n + \mathbf{q}^n \\ Q &= A^{-1}B, \quad \mathbf{q}^n = A^{-1}\mathbf{c} \end{aligned} \tag{3.10}$$

Definition 3.1 *The difference equation* $\mathbf{w}^{n+1}Q = Q\mathbf{w}^n$ *is* stable *with respect to the norm* $\|\cdot\|$ *if there exist a constant* $L \geq 0$ *independent of* h, k, *and* n, *such that*

$$\|\mathbf{w}^n\| \leq L \cdot \|\mathbf{w}^0\| \qquad \textit{for all} \quad n \leq \frac{T}{h} \tag{3.11}$$

no matter choice of $\mathbf{w}^0$.

Lemma 3.1 *The difference equation* $\mathbf{w}^{n+1}Q = Q\mathbf{w}^n$ *is stable if* $\|Q^n\| \leq L$

Proof Assume that $\|Q^n\| \leq L$. Then

$$\|\mathbf{w}^n\| = \|Q^n\mathbf{w}^0\| \leq \|Q^n\| \cdot \|\mathbf{w}^0\| \leq L \cdot \|\mathbf{w}^0\|.$$

Assume that $\|\mathbf{w}^n\| \leq L \cdot \|\mathbf{w}^0\|$, that is, assume that the difference equation is stable. Since $\mathbf{w}^0$ can be chosen arbitrary, choose $\mathbf{w}^0$ such that

$$\|Q^n\mathbf{w}^0\| = \|Q^n\| \cdot \|\mathbf{w}^0\| = \|\mathbf{w}^n\| \leq L \cdot \|\mathbf{w}^0\|.$$

This implies that $\|Q^n\| \leq L$. □

[1] If we integrate a differential equation $\dot{u} = f(u)$ from t_n to t_{n+1} using the trapezoid rule, we obtain the following equation for $u(t_{n+1})$: $u_{n+1} = u_n + \frac{1}{2}k(f(u_n) + f(u_{n+1})$

Lemma 3.2 (Sufficent conditon for stability) *If there exist a constant $c \geq 0$, independent of h, k and n such that $\|Q\| \leq 1 + ck$, then the difference equation is stable*

Proof We prove that $\|Q^n\| \leq L$

$$\begin{aligned} \|Q^n\| &\leq \|Q\|^n \leq (1+ck)^n \leq (1+ck)^{T/k} \\ &\leq [\exp(ck)]^{T/k} = \exp(cT) = L \end{aligned}$$

□

Lemma 3.3 (Necessary condition for stability) *For a differnce equation (3.10) to be stable the iteration matrix must satisfy $\rho(Q) \leq 1 + \mu k$ for some constant $\mu \geq 0$, where $\rho(Q)$ denotes the spectral radius of Q.*

Proof Assume that $L \geq 1$. Since $\rho(Q^n) = (\rho(Q))^n \leq \|Q\|^n \leq L$ and $n = T/k$ we have that

$$\begin{aligned} \rho(Q) &\leq L^{1/n} = L^{k/T} = \Big[\exp\Big(\ln L\Big)\Big]^{k/T} \\ &= \exp\Big(\frac{\ln L}{T}k\Big) = 1 + k\frac{\ln L}{T}\exp\Big(\frac{\ln L}{T}\theta k\Big), \qquad 0 \leq \theta \leq 1 \end{aligned}$$

And since $k \leq T$, we deduce that $\rho(Q) \leq 1 + \mu k$. If $Q = Q^T$, then $\rho(Q) \leq 1 + \mu K$ is a necessary and sufficient condition for stability. □

Let us illustrate the theory with a example. If we apply the θ- method to the heat equation, we get

$$(I - \theta r B)\mathbf{u}^{n+1} = (I + (1-\theta)rB)\mathbf{u}^n + \mathbf{c}^n$$

where $B = B(1, -2, 1)$. B is symmetric, it has orthogonal eigenvectors, and the eigenvalues are

$$\lambda_i = -4\sin^2\Big(\frac{i\pi}{2(N+1)}, \qquad i = 1, ..., N$$

If $\Lambda = \text{diag}(\lambda_i)$, and P is a matrix whose columns are the eigenvectors of B. Then $PP^T = I$ and $B = P\Lambda P^T$ is the spectral decomposition of B. We recognize the iteration matrix Q from the θ-method

$$\begin{aligned} Q &= (I - \theta r S)^{-1}(I + (1-\theta)rS) \\ &= [P(I - \theta r\Lambda)P^T]^{-1}P(I + (1-\theta)r\Lambda)P^T \\ &= P(I - \theta r\Lambda)P^T)]^{-1}P^T P(I + (1-\theta)r\Lambda)P^T \\ &= PDP^T \end{aligned}$$

where D is a diagonal matrix with the elements

$$d_i = \frac{1 + (1-\theta)r\lambda_i}{1 - \theta r \lambda_i}$$

A sufficient condition for stability in $\|\cdot\|$ or $\|\cdot\|$ is

$$\max_i |d_i| \leq 1 + \mu k \qquad \mu \geq 0$$

Let $\mu = 0$, then we have

$$d_i = \frac{1 - (1-\theta)r4\sin^2\left(\frac{i\pi}{2(N+1)}\right)}{1 + \theta r 4 \sin^2\left(\frac{i\pi}{2(N+1)}\right)}\Big)$$

We always have $d_i \leq 1$. The condition $-1 \geq d_i$ is satisfied when

$$\begin{aligned} - \quad & \left[1 + 4r\theta \sin^2\left[\frac{i\pi}{2(N+1)}\right]\right] \leq 1 - 4r(1-\theta)\sin^2\left[\frac{i\pi}{2(N+1)}\right] \\ \Longrightarrow \quad & 2 - 4r(2\theta - 1)\sin^2\left[\frac{i\pi}{2(N+1)}\right] \geq 0 \end{aligned}$$

This condition is always satisfied when $\theta \geq \frac{1}{2}$. If $\theta \leq \frac{1}{2}$. Using the inequality $0 < \sin^2\left(\frac{i\pi}{2(N+1)}\right) < 1$ yields

$$2 - 4r(1 - 2\theta) \geq 0$$

We can conclude tha the θ-method is stable when

$$\begin{aligned} \theta \quad &\geq \quad \frac{1}{2} \qquad \text{or when} \\ \theta \quad &< \quad \frac{1}{2} \quad \text{and} \quad r \leq \frac{1}{2(1-2\theta)} \end{aligned}$$

3.3 Fourier Analysis

The error given by the truncation error is of limited use, we wish to analyse the error futher using Fourier series.
Assume that we have an inititial-value problem

$$\begin{aligned} \frac{\partial u}{\partial t} &= \mathcal{L}u, \qquad -\inf < x < \inf \\ u(x,0) &= g(x) \end{aligned} \tag{3.12}$$

For time dependent problem to be stable, we require that

$$\|\mathbf{u}\| \leq L\|\mathbf{u}_0\| \tag{3.13}$$

for some finite $L \in \mathbb{R}$, and $\mathbf{u}_0 = \mathbf{u}(\mathbf{x}, 0)$. It is, however, not easy to find a bound on the solution, or equivalently, the error, for a coupled system of equations. Luckily we are not totally lost. If the linear system has a complete set of eigenvectors, then we can use the discrete Fourier transform to uncouple the equaitons. From the Parsevals identity,

$$\|\hat{\mathbf{u}}\|_2 = \|\mathbf{u}\|_2$$

it follows that $\|\hat{\mathbf{u}}\|_2 \leq L\|\hat{\mathbf{u}}_0\|_2$ is equivalent to (3.13).For the Fourier stability analysis to be accurate we require an infinite or periodic grid (no influence from boundary values) and a uniform mesh.
An arbitrary complex function $\hat{\omega}(\theta),\ \theta \in [0, 2\pi]$ has the Fourier expansion

$$\hat{w}(\theta) = \sum_{m=-\infty}^{\infty} w_m \mathrm{e}^{-\mathrm{i}m\theta} \tag{3.14}$$

where $\mathrm{i} := \sqrt{-1}$, and

$$w_m = \frac{1}{2\pi}\int_0^{2\pi} \hat{w}(\theta)\mathrm{e}^{\mathrm{i}m\theta}d\theta \tag{3.15}$$

To illustrate the use of Fourier series, consider the difference equation

$$u_j^{n+1} = u_j^n + r(u_{j+1}^n - 2u_j^n + u_{j-1}^n)$$

with $r = k/h^2$. From the definition of the Fourier transform it follows that

$$\begin{aligned}
\hat{\mathbf{u}}^{n+1} &= \sum_{j=-\infty}^{\infty} u_j^{n+1}\exp(-\mathrm{i}j\theta) \\
&= \sum_{j=-\infty}^{\infty} [ru_{j+1}^n\mathrm{e}^{-\mathrm{i}j\theta} + (1-2r)u_j^n\mathrm{e}^{-\mathrm{i}j\theta} + ru_{j-1}^n\mathrm{e}^{-\mathrm{i}j\theta}] \\
&= r\sum_{j=-\infty}^{\infty} u_j^n\mathrm{e}^{-\mathrm{i}(m-1)\theta} + (1-2r)\sum_{j=-\infty}^{\infty} u_j^n\mathrm{e}^{-\mathrm{i}j\theta} + r\sum_{j=-\infty}^{\infty} u_j^n\mathrm{e}^{-\mathrm{i}(m+1)\theta} \\
&= r\mathrm{e}^{\mathrm{i}\theta}\hat{\mathbf{u}}^n(\theta) + (1-2r)\hat{\mathbf{u}}^n(\theta) + r\mathrm{e}^{-\mathrm{i}\theta}\hat{\mathbf{u}}^n(\theta) \\
&= (1 - 2r + 2r\cos\theta)\hat{\mathbf{u}}^n(\theta) = (1 - 4r\sin^2\frac{\theta}{2})\hat{\mathbf{u}}^n
\end{aligned}$$

Let $\rho(\theta) = 1 - 4r\sin^2\frac{\theta}{2}$. Then

$$\hat{\mathbf{u}}^n = \rho(\theta)^n \hat{\mathbf{u}}^0(\theta)$$

Clearly the method is stable if $|\rho(\theta)| \leq 1$. In our example this is true whenever $-1 \leq 1 - 4r\sin^2\frac{\theta}{1}2 \leq 1$. The last inequality is always true. The first implies

$$r\sin^2\frac{\theta}{2} \leq \frac{1}{2},$$

which is true for all θ if r. This is a neccesary and sufficient condition for stability.
To determine if a method is stable or not, it is sufficient to study $\rho(\theta)$, defined by the relation $\hat{\mathbf{u}}^{n+1} = \rho(\theta)\hat{\mathbf{u}}^n$. One finds $\rho(\theta)$ by taking the DFT of all terms in the difference expression.
Another way of determing the stability is to express $\mathbf{u}$ in terms of its frequency components. From the *inverse* DFT we have

$$u_j = \frac{1}{2\pi}\sum_{k=0}^{0} \mathrm{e}^{\mathrm{i}kx_j}\hat{u}_k = \frac{1}{2\pi}\sum_{k=0}^{0} \rho^n \mathrm{e}^{\mathrm{i}kx_j}\hat{u}_0, \qquad j = 1, ..., N$$

If we guess a solution $u_j^n = \rho^n \mathrm{e}^{\mathrm{i}j\theta}$, and insert this into the difference equation we obtain an expression for ρ.

3.3.1 Lax Equivalence Theorm

The connection between convergence, consistency and stability is given by Lax[2] equivalence theorem

Theorem 3.1 (Lax Equivalence Theorem) *A consistent, two level difference scheme for a well-posed linear initial-value problem is convergent if and only if it is stable.*

In other words

$$\text{consistency} + \text{stability} \Leftrightarrow \text{convergence}$$

Let us prove that consitency and stability implies convergence. Assume we have the following two level scheme

$$\mathbf{v}^{n+1} = Q\mathbf{v}^n + k\mathbf{G}^n$$

[2]Peter D. Lax was born in Budapest in 1926. His family moved to New York in 1941, where Lax studied applied mathematics under John von Neumann and Richare Courant. Because of his mathematicall skills he was sendt to Los Almos to work on the Manhattan-project (the atomic bomb). He later returned to New York. Lax received the Abel prize in 2005 for his contributions to applied mathematics

If $u = u(x,t)$ is the exact solution of the initial-value problem, then the difference scheme is accurate of order (p,q) if

$$\mathbf{u}^{n+1} = Q\mathbf{u}^n + k\mathbf{G}^n + k\mathbf{tau}^n$$

with $\|\tau^n\| = \mathcal{O}(h^p) + \mathcal{O}(k^q)$ and $\mathbf{u}^n$ is the solution at the grid points. The method is consistent with respect to the norm $\|\cdot\|$ if

$$\|\tau^n\| \overset{h,k\to 0}{\longrightarrow} 0.$$

In addition we know that the method is stable with respect to the norm $\|\cdot\|$ if and only if there exist a constand L independent of k, h and n such that $\|Q^n\| \leq L$ for all n. Define the error $\mathbf{e}^n = \mathbf{u}^n - \mathbf{v}^n$. From this definition and the definition of the difference scheme it follows that

$$\mathbf{e}^{n+1} = Q\mathbf{e}^n + k\tau^n = Q^n\mathbf{e}^0 + k\sum_{j=0}^{N} Q^j\tau^{n-j} \leq (N+1)kLC(\Delta h^p + \Delta k^q) = \mathcal{O}(h^p + k^q)$$

Thus the method is convergent of order (p,q). This proves the following theorem

Theorem 3.2 (Lax Theorem) *If a two-level difference scheme*

$$\mathbf{v}^{n+1} = Q\mathbf{v}^n + k\mathbf{G}^n$$

is accurate of order (p,q) in the norm $\|\cdot\|$ to a well-posed linear initial-value problem and is stable with respect to the norm $\|\cdot\|$, then it is convergent of order (p,q) with respect to the norm $\|\cdot\|$

Chapter 4

Split and Factored Forms

Implisit Euler and Crank-Nicolson gives unconditionally stable schemes, but not for free. Each time step requires the solution of a large and sparse sytem of equations. This can be avoided with *split and factor forms*. Split and factored forms provide efficient solvers for practical multidimensional applications, they are especially useful for the derivation of practical algorithms that use implicit methods

4.1 The Concept

A general system of ODEs resulting from the semi-discrete approximation to a PDE is

$$\frac{d\mathbf{u}}{dt} = A\mathbf{u} - \mathbf{f}$$

A splitting of the matrix A result in

$$\frac{d\mathbf{u}}{dt} = [A_1 + A_2]\mathbf{u} - \mathbf{f}$$

where $A = [A_1 + A_2]$. If we use the forward difference approximation to the time-deivative we can express new data $\mathbf{u}^{n+1}$ in terms of old $\mathbf{u}_n$

$$\mathbf{u}^{n+1} = [I + kA_1 + kA_2]\mathbf{u}^n - k\mathbf{f} + \mathcal{O}(k^2) \tag{4.1}$$

or its equivalent

$$\mathbf{u}^{n+1}[[I + kA_1][I + kA_2] - k^2A_1A_2]\mathbf{u} - k\mathbf{f} + \mathcal{O}(k^2)$$

If we dropp higher order terms we obtain

$$\mathbf{u}^{n+1} = [I + kA_1][I + kA_2]\mathbf{u} - k\mathbf{f} + \mathcal{O}(k^2) \tag{4.2}$$

The methods (4.2) and (4.1) has the same formal order of accuracy, However, their stability properties differe. For a method to be useful it should be accurate and stable. Stability analysis of split and factored forms can be found in [2].

4.2 Factoring Physical Representations

Suppose we semi-disretize a convection diffusion equation, then a possible representation of the system of equations is

$$\frac{d\mathbf{u}}{dt} = A_c\mathbf{u} + A_d\mathbf{u} + \mathbf{bc} \tag{4.3}$$

with A_c and A_d representing convection and diffusion terms, respectively. If we again rely on the forward difference approximation to approximate the time derivative, the time marching difference equation becomes

$$\mathbf{u}_{n+1} = [I + kA_d + kA_c]\mathbf{u}^n + h\mathbf{bc} + \mathcal{O}(k^2). \tag{4.4}$$

Now consider the factored form

$$\begin{aligned} \mathbf{u}^{n+1} &= [I + kA_d]\Big([I + kA_c]\mathbf{u}^n + k\mathbf{bc}\Big) \\ &= \underbrace{[I + kA_d + kA_c]\mathbf{u}^n + k\mathbf{bc}}_{\text{original unfactored terms}} \\ &\quad \underbrace{+ k^2A_d\Big(A_c\mathbf{u}^n + \mathbf{bc}\Big)}_{\text{higher-order terms}} \end{aligned} \tag{4.5}$$

and we se that (4.5) and the original unfactored form (4.4) have identical orders of accuracy in the time approximation. We can write (4.5) as a predictor-corrector sequence

$$\begin{aligned} \tilde{\mathbf{u}}^{n+1} &= [I + kA_c]\mathbf{u} + \mathbf{bc} \\ \mathbf{u}^{n+1} &= [I + kA_d]\tilde{\mathbf{u}}^{n+1} \end{aligned} \tag{4.6}$$

Another way to approximate (4.3) is given by the expression

$$\begin{aligned} \mathbf{u}^{n+1} &= [I - kA_d]^{-1}\Big([I + kA_c]\mathbf{u}^n + k\mathbf{bc}\Big) \\ &= \underbrace{[I + kA_d + kA_c]\mathbf{u}^n + k\mathbf{bc}}_{\text{original unfactored terms}} + \mathcal{O}(k^2) \end{aligned} \tag{4.7}$$

where in this approximation we have used the fact that

$$[I - kA_d]^{-1} = I + kA_d + k^2 A_d^2 + ...$$

if $k \cdot \|A_d\| < 1$. This can be expressed as a predictor corrector sequence

$$\begin{aligned} \tilde{\mathbf{u}}^n &= [I + kA_c]\mathbf{u}^n + h\mathbf{bc} \\ [I - kA_d]^{-1}\,\mathbf{u}^{n+1} &= \tilde{\mathbf{u}}^{n+1}. \end{aligned} \tag{4.8}$$

The convection operatro is applied explicitly, as before, but the diffusion operator is now implicit, requiring a tridiagonal solver if the diffusion term is central differenced. Since numerical stiffness is generally much more severe for the diffusion process (parabolic), than for the convection process (hyperbolic), this factored form would appear to be supperior to the explicit method provided by (4.6). However, the important aspect of stability has yet to be discussed. An alternative derivation of the factored form (4.8) is obtained if we apply the following implicit time-marching method

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{k} = A_c\mathbf{u}_n + A_d\mathbf{u}_{n+1} + \mathbf{bc} + \mathcal{O}(k^2).$$

If we reorder this equation we get

$$[I - kA_d]\mathbf{u}^{n+1} = [I + kA_c]\mathbf{u}^n + k\mathbf{bc}$$

which is identical to (4.8).

4.3 Factoring Space Matrix Operators in 2D

Let us analyze how to apply a factored form to the model diffusion problem in two dimensions. The diffusion equation is, as before, given by

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \tag{4.9}$$

To illustrate the concepts, we disretize this PDE on a rectangular 3×4 point mesh.

		$\odot$	$\odot$	$\odot$	$\odot$	
N_y	$\odot$	13	23	33	43	$\odot$
k	$\odot$	12	22	32	42	$\odot$
1	$\odot$	11	21	31	41	$\odot$
		$\odot$	$\odot$	$\odot$	$\odot$	
		1	j	...	N_x	

Here the numbers represent the location in the mesh corresponding to the variable bearing the same index. For example, the variable associated with $j = 3$ and $k = 2$ is u_{32}. M_x and M_y is the number of intrior x and y points, respectively.

4.3.1 Ordering of Unknowns

The representation and storage of data is crucial for an efficient computer code. PDE code is no exception. In the following the *data-base* referre to the storage of a dimensioned array. We consider two row-wise, and column-wise ordering of the unknowns. We refer to each row or column group as a *space vector* The space vector is labeled $U^{(x)}$ for a row wise ordering, and similar, $U^{(y)}$ for a column ordering. The dimension of U is $[N_x \cdot N_y, 1]$ where N_x is the number of internal x points, and N_y is the number of internal y points. The space vectors $U^{(x)}$ and $U^{(y)}$are related by a permutation matrix P_{xy} defined by

$$U^{(x)} = P_{xy}U^{(y)} \qquad U^{(y)} = P_{yx}U^{(x)}, \tag{4.10}$$

where

$$P_{yx} = P_{xy}^T = P_{xy}^{-1}$$

We commence with a spatial discretization of our model problem (4.9) using the centered three-point operator for both directions. The matrix representation of the model problem can be expressed as

$$\frac{dU}{dt} = A_{x+y}U + \mathbf{bc} \tag{4.11}$$

where the structure of A_{x+y} depends on the ordering of U, they are however related by the permutation matrix

$$A_{x+y}^{(x)} = P_{xy}A_{x+y}^{y} \cdot P_{yx}. \tag{4.12}$$

4.4 Space Splitting and Factoring

The matrix A_{x+y} is can be split into two matrices. For row-wise ordering yields

$$A_{x+y}^{(x)} = A_x^{(x)} + A_y^{(x)}. \tag{4.13}$$

Similarily, for column-wise ordering

$$A_{x+y}^{(y)} = A_x^{(y)} + A_y^{(y)} \tag{4.14}$$

$$A^{(x)}_{x+y} \cdot U^{(x)} = \left[\begin{array}{cccc|cccc|cccc}
\bullet & x & & & o & & & & & & & \\
x & \bullet & x & & & o & & & & & & \\
 & x & \bullet & x & & & o & & & & & \\
 & & x & \bullet & & & & o & & & & \\
\hline
o & & & & \bullet & x & & & o & & & \\
 & o & & & x & \bullet & x & & & o & & \\
 & & o & & & x & \bullet & x & & & o & \\
 & & & o & & & x & \bullet & & & & o \\
\hline
 & & & & o & & & & \bullet & x & & \\
 & & & & & o & & & x & \bullet & x & \\
 & & & & & & o & & & x & \bullet & x \\
 & & & & & & & o & & & x & \bullet
\end{array}\right] \cdot \left[\begin{array}{c} u_{11} \\ u_{21} \\ u_{31} \\ u_{41} \\ -- \\ u_{12} \\ u_{22} \\ u_{32} \\ u_{42} \\ -- \\ u_{13} \\ u_{23} \\ u_{33} \\ u_{43} \end{array}\right]$$

Figure 4.1: Elements in two dimensional, central-difference, matrix operator, A_{x+y}, for 3×4 mesh. Data-base composed of M_y x-vectors stored in $U^(x)$. Entries for $x \rightarrow x$, for $y \rightarrow 0$, for both $\rightarrow \bullet$

$$A^{(y)}_{x+y} \cdot U^{(y)} = \left[\begin{array}{ccc|ccc|ccc|ccc}
\bullet & o & & x & & & & & & & & \\
o & \bullet & o & & x & & & & & & & \\
 & o & \bullet & & & x & & & & & & \\
\hline
x & & & \bullet & o & & x & & & & & \\
 & x & & o & \bullet & o & & x & & & & \\
 & & x & & o & \bullet & & & x & & & \\
\hline
 & & & x & & & \bullet & o & & x & & \\
 & & & & x & & o & \bullet & o & & x & \\
 & & & & & x & & o & \bullet & & & x \\
\hline
 & & & & & & x & & & \bullet & o & \\
 & & & & & & & x & & o & \bullet & o \\
 & & & & & & & & x & & o & \bullet
\end{array}\right] \cdot \left[\begin{array}{c} u_{11} \\ u_{12} \\ u_{13} \\ -- \\ u_{21} \\ u_{22} \\ u_{23} \\ -- \\ u_{31} \\ u_{32} \\ u_{33} \\ -- \\ u_{41} \\ u_{42} \\ u_{43} \end{array}\right]$$

Figure 4.2: Elements in two dimensional, central-difference, matrix operator, A_{x+y}, for 3×4 mesh. Data-base composed of M_x y-vectors stored in $U^{(x)}$. Entries for $x \rightarrow x$, for $y \rightarrow 0$, for both $\rightarrow \bullet$

$$A_x^{(x)} \cdot U^{(x)} = \left[\begin{array}{cccc|cccc|cccc}
x & x & & & & & & & & & & \\
x & x & x & & & & & & & & & \\
 & x & x & x & & & & & & & & \\
 & & x & x & & & & & & & & \\
\hline
 & & & & x & x & & & & & & \\
 & & & & x & x & x & & & & & \\
 & & & & & x & x & x & & & & \\
 & & & & & & x & x & & & & \\
\hline
 & & & & & & & & x & x & & \\
 & & & & & & & & x & x & x & \\
 & & & & & & & & & x & x & x \\
 & & & & & & & & & & x & x
\end{array}\right] \cdot U^{(x)}$$

$$A_y^{(x)} \cdot U^{(x)} = \left[\begin{array}{cccc|cccc|cccc}
o & & & & o & & & & & & & \\
 & o & & & & o & & & & & & \\
 & & o & & & & o & & & & & \\
 & & & o & & & & o & & & & \\
\hline
o & & & & o & & & & o & & & \\
 & o & & & & o & & & & o & & \\
 & & o & & & & o & & & & o & \\
 & & & o & & & & o & & & & o \\
\hline
 & & & & o & & & & o & & & \\
 & & & & & o & & & & o & & \\
 & & & & & & o & & & & o & \\
 & & & & & & & o & & & & o
\end{array}\right] \cdot U^{(x)}$$

Figure 4.3: The splitting of $A_{x+y}^{(x)}$

$$A^{(y)}_{x+y} \cdot U^{(y)} = \left[\begin{array}{ccc|ccc|ccc|ccc}
x & & & x & & & & & & & & \\
 & x & & & x & & & & & & & \\
 & & x & & & x & & & & & & \\
\hline
x & & & x & & & x & & & & & \\
 & x & & & x & & & x & & & & \\
 & & x & & & x & & & x & & & \\
\hline
 & & & x & & & x & & & x & & \\
 & & & & x & & & x & & & x & \\
 & & & & & x & & & x & & & x \\
\hline
 & & & & & & x & & & x & & \\
 & & & & & & & x & & & x & \\
 & & & & & & & & x & & & x
\end{array}\right] \cdot U^{(y)}$$

$$A^{(y)}_{y} \cdot U^{(y)} = \left[\begin{array}{ccc|ccc|ccc|ccc}
o & o & & & & & & & & & & \\
o & o & o & & & & & & & & & \\
 & o & o & & & & & & & & & \\
\hline
 & & & o & o & & & & & & & \\
 & & & o & o & o & & & & & & \\
 & & & & o & o & & & & & & \\
\hline
 & & & & & & o & o & & & & \\
 & & & & & & o & o & o & & & \\
 & & & & & & & o & o & & & \\
\hline
 & & & & & & & & & o & o & \\
 & & & & & & & & & o & o & o \\
 & & & & & & & & & & o & o
\end{array}\right] \cdot U^{(y)}$$

Figure 4.4: The splitting of $A^{(y)}_{x+y}$

The splittings in (4.13) and (4.14) can be combined with factoring. Recall the semidiscrete equation

$$\frac{dU}{dt} = A_{x+y}U + \mathbf{bc} = \frac{dU^{(x)}}{dt} = [A_x^{(x)} + A_y^{(x)}]U^{(x)} + \mathbf{bc}.$$

If we approximate the time derviative with the implicit Euler method we obtain

$$U_{n+1}^{(x)} = U_n^{(x)} + k\left[A_x^{(x)} + A_y^{(x)}\right] U_{n+1}^{(x)} + k\mathbf{bc} + \mathcal{O}(k^2)$$

or

$$\left[I - kA_x^{(x)} - kA_y^{(x)}\right] U_{n+1}^{(x)} = U_n^{(x)} + k\mathbf{bc} + \mathcal{O}(k^2). \tag{4.15}$$

Factoring this yields

$$\left[I - kA_x^{(x)}\right]\left[I - kA_y^{(x)}\right] U_{n+1}^{(x)} = U_n^{(x)} + k\mathbf{bc} + \mathcal{O}(k^2). \tag{4.16}$$

The corresponding predictor-corrector formulation is given by

$$\begin{aligned} \left[I - kA_x^{(x)}\right] \tilde{U}^{(x)} &= U_n^{(x)} + k\mathbf{bc} \\ \left[I - kA_y^{(x)}\right] U_{n+1}^{(x)} &= \tilde{U}^{(x)} \end{aligned}$$

We can reduce the bandwidth of the last equation with a permutation of the data-base. Note that the permutation relation also holds for the split matrices, so

$$A_y^{(x)} = P_{xy}A_y^{(y)}P_{yx}$$

and

$$A_x^{(x)} = P_{xy}A_x^{(y)}P_{yx}.$$

Inserting this into the previous predictor-corrector equation yields

$$\begin{aligned} \left[I - kA_x^{(x)}\right] \tilde{U}^{(x)} &= U_n^{(x)} + k\mathbf{bc} \\ \left[I - kA_y^{(y)}\right] U_{n+1}^{y} &= \tilde{U}^{(y)} \end{aligned} \tag{4.17}$$

4.5 Second-Order Factored Implicit Methods

A second order approximation in time can be obtained if we apply the trapezoidal method to (4.11) where the matrix operator have been split as in (4.13) or (4.14). The result is

$$\left[I - \frac{1}{2}kA_x - \frac{1}{2}kA_y\right] U_{n+1} = \left[I + \frac{1}{2}kA_x + \frac{1}{2}kA_y\right] U_n + k\mathbf{bc} + \mathcal{O}(k^3). \tag{4.18}$$

Factor both sides to obtain

$$\left[\left[I-\frac{1}{2}kA_x\right]\left[I-\frac{1}{2}kA_y\right]-\frac{1}{4}k^2A_xA_y\right]U_{n+1}$$
$$=\left[\left[I+\frac{1}{2}kA_x\right]\left[I+\frac{1}{2}kA_y\right]-\frac{1}{4}k^2A_xA_y\right]U_n+k\mathbf{bc}+\mathcal{O}(k^3)$$

The cross term $\frac{1}{4}[A_xA_y](U_{n+1}-U_n$ is proportional to k^3 and can be canceled without loss of accuracy. The split and factored scheme now reads

$$\left[I-\frac{1}{2}kA_x\right]\left[I-\frac{1}{2}kA_y\right]U_{n+1}=\left[I+\frac{1}{2}kA_x\right]\left[I+\frac{1}{2}kA_y\right]U_n+\mathbf{bc} \quad (4.19)$$

This is a particular case of the classical ADI (alternating direction implicit) method, usually written as

$$\begin{aligned}\left[I-\frac{1}{2}kA_x\right]\tilde{U} &= \left[I+\frac{1}{2}kA_y\right]U_n+\frac{1}{2}F_n\\ \left[I-\frac{1}{2}kA_y\right]U_{n+1} &= \left[I+\frac{1}{2}kA_x\right]\tilde{U}+\frac{1}{2}kF_{n+1}+\mathcal{O}(k^3)\end{aligned} \quad (4.20)$$

The difference between (4.19) and (4.20) is the introduction of a time dempendent source.
It is time to do a numerical simulation. We consider a two-dimensional heat conduction problem with Dirichlet or Neumann boundary condition. The solver is our new friend ADI. Recall that for a Dirichlet problem the centered three-point matrix operator is given by

$$\Delta_0^2x=\frac{1}{\Delta x^2}B(1,-2,1)+\mathbf{bc}$$

Thus, if we use the three-point operator to approximate spatial derivatives we can write (4.20) as

$$\begin{aligned}\left[I-\frac{1}{2}\alpha_x\Delta_{0,x}^2\right]\tilde{U} &= \left[I+\frac{1}{2}\alpha_y\Delta_{0,y}^2\right]U_n\\ \left[I-\frac{1}{2}\alpha_y\Delta_{0,y}^2\right]U_{n+1} &= \left[I+\frac{1}{2}\alpha_x\Delta_{0,x}^2\right]\tilde{U}+\mathcal{O}(k^3)\end{aligned} \quad (4.21)$$

where $\alpha_x=\frac{k}{\Delta x^2}$ and $\alpha_y=\frac{k}{\Delta y^2}$. The initial condition is given by a double-Gaussian profile

$$\begin{aligned}u(x,y,0) &= \exp\left\{-100\left[(x+\frac{L_x}{4})^2+(y-\frac{L_y}{4})^2\right]\right\}\\ &\quad+\frac{1}{2}\exp\Big(-100\left[x^2+y^2\right]\Big)\end{aligned}$$

Figure 4.5: Initial conditions

Figure 4.6: Temperature profile at $t = [0.001, 0.005, 0.01, 0.05]$

```
  % ADI demonstration
clear;
nx=48; ny=nx; Lx=1; Ly=Lx; hx=Lx/(nx+1); hy=Ly/(ny+1);
alphax=1; alphay=alphax*(hx/hy)^2;k=alphax*hx^2;
x=(0:nx+1)*hx-Lx/2; y=(0:ny+1)*hy-Ly/2;
xdiff1=x+Lx/4; ydiff1=y-Ly/4; xdiff2=x; ydiff2=y;
% d_squared is an nx by ny matrix containing the distance
% squared from the origin to each of the nx by ny points.
d_squared=(xdiff1.^2)'*ones(1,ny+2)+ones(nx+2,1)*(ydiff1.^2);
u=exp(-100*d_squared);
d_squared=(xdiff2.^2)'*ones(1,ny+2)+ones(nx+2,1)*(ydiff2.^2);
u=u+0.5*exp(-100*d_squared); uplot=1;
figure(1); mesh(x,y,u','EdgeColor','black');
for tend=[0.001 0.005 0.01 0.05]
uhalf=u; unew=u; iplot=1; nsteps=ceil(tend/k);
nplots=25; plotsteps=ceil(nsteps/nplots);
% Set up the arrays for the ADI method
ahalf=-(alphax/2)*ones(nx,1); bhalf=(1+alphax)*ones(nx,1);
chalf=ahalf; ahalf(1)=0; chalf(nx)=0;
A1=spdiags([ahalf,bhalf,chalf],[-1,0,1],nx,nx);
afull=-(alphay/2)*ones(ny,1); bfull=(1+alphay)*ones(ny,1);
cfull=afull; afull(1)=0; cfull(ny)=0;
A2=spdiags([afull,bfull,cfull],[-1,0,1],ny,ny);
% Solve the system of equations
for istep=1:nsteps
     for j=2:ny+1
        fhalf(1:nx)=(alphay/2)*u(2:nx+1,j-1)...
            +(alphay/2)*u(2:nx+1,j+1)+(1-alphay)*u(2:nx+1,j);
        utemp=A1\fhalf'; uhalf(2:nx+1,j)=utemp';
     end
     for i=2:nx+1
        ffull(1:ny)=(alphax/2)*uhalf(i-1,2:ny+1)...
            +(alphax/2)*uhalf(i+1,2:ny+1)+...
              (1-alphax)*uhalf(i,2:ny+1);
        utemp=A2\ffull'; unew(i,2:ny+1)=utemp;
    end
    u=unew; uhalf=u;
end
figure(2); subplot(2,2,uplot); uplot=uplot+1;
mesh(x,y,u','EdgeColor','black');
end
```

For a Neumann problem we have to alter the matrix difference operators $\Delta_{0,x}$ and $\Delta_{0,y}$. At the right boundary we use a first-order point operator

$$(\Delta_{0,x}^2)_N = \frac{1}{3\Delta x^2}(2u_{N-1} - 2u_N) + \frac{2}{3\Delta x}\Big(\frac{\partial u}{\partial x}\Big)_{N+1}.$$

At the left boundary the point operator reads

$$(\Delta_{0,x}^2)_1 = \frac{1}{3\Delta y^2}(-2u_1 + 2u_2) - \frac{2}{3\Delta x}\Big(\frac{\partial u}{\partial x}\Big)_0$$

If we use a similar approximation at the top and bottom we obtain the following matrix operators

$$A_x = \frac{1}{\Delta x^2}B(\mathbf{a}, \mathbf{b}, \mathbf{c}) \quad A_y = \frac{1}{\Delta y^2}B(\mathbf{a}, \mathbf{b}, \mathbf{c})$$

where

$$\begin{aligned} \mathbf{a} &= [1, 1, ..., 2/3]^T \\ \mathbf{b} &= [-2/3, -2, -2, ..., -2/3]^T \\ \mathbf{c} &= [2/3, 1, 1, ..., 1]^T \end{aligned}$$

Here we have assumed zero flux at all boundaries. The initial condition corresponds to a uniform temperature with superimposed hot and cold spots.

$$\begin{aligned} u(x, y, 0) &= 1 + 2\exp\left\{-10\left[(x + \frac{L_x}{2})^2 + (y - \frac{L_y}{4})^2\right]\right\} \\ &\quad - \exp\left\{-100\left[(x - \frac{L_x}{4})^2 + (y + \frac{L_y}{4})^2\right]\right\} \end{aligned}$$

For stiff equations, where implicit methods are required to permit reasonably large time steps, the use of factored forms becomes a very valuable tool for realistic problems. For example, for the unfactored trapezoidal method given by

$$\left[I - \frac{1}{2}hA_{x+y}\right]U_{n+1} = \left[I + \frac{1}{2}hA_{x+y}\right]U_n + h\mathbf{bc}$$

To find U_{n+1} requires the solution of a sparse, but very large system of equations. Futhermore, if we solve this equation with a direct method, the entries between the main and outermost diagonal are filled with nonzero values. This is not desired, because more nonzero values implies more computing and more storage. The ADI method inherit the stability properties of the Crank Nicolson method, but instead of solving one large spase system of equation

at each iteration it solves several tridiagonal systems. For higher dimensions this strategy is more efficient. The predictor step involves the solution of N_y tridiagonal systems, each of size N_x. The corrector step involves the solution of Nx tridiagonal systems each of size N_y. By way of contrast, the Crank-Nicolson solves pentadiagonal system of size N_xN_y at each time step (A_{x+y} is a $N_xN_y \times N_xN_y$ matrix).

4.6 Analysis of Split and Factored Forms

To estimate the stability and steady-state properties of split and factored forms the spectral decomposition is introduced. The results found from the analysis is not necessary or sufficient to guarantee stability. It only indicates the behaivour of practical, or not so practical methods. Stability analysis is usually done under the assumption of a infinite domain, which implies that the boundary values has no influence on the analysis. Alternatively the boundary conditions are periodic. From a physical point of view, a periodic boundary condition indicates a solution that repats itself. Periodic boudary conditons leads to *circulant* matrix difference operators.
A general $d \times d$ circulant matrix is a matrix whose jth row, $j = 2, 3, ..., d$, is a 'right-rotated' $(j-1)$th row

$$C(\kappa) = \begin{bmatrix} \kappa_0 & \kappa_1 & \kappa_2 & \dots & \kappa_{d-1} \\ \kappa_{d-1} & \kappa_0 & \kappa_1 & \dots & \kappa_{d-2} \\ \kappa_{d-2} & \kappa_{d-1} & \kappa_0 & \dots & \kappa_{d-2} \\ \vdots & & & & \vdots \\ \kappa_1 & \kappa_2 & \kappa_3 & \dots & \kappa_0 \end{bmatrix}$$

Two properties of the circulant matrix are essential for the stability and convergence analysis of time-marching schemes. Circulant matrices commute, and have a common set of eignvectors. This is the conclusion of the following lemma.

Lemma 4.1 *The eigenvalues of* $C(\kappa)$ *are* $\kappa(\omega_d^j)$, $j = 0, 1, ..., d-1$, *where*

$$\kappa(z) := \sum_{l=0}^{d-1} \kappa_l z^l, z \in \mathbb{C}$$

and $\omega_d = \exp(2\pi \mathrm{i}/d)$ *is the* d*th primitive root of unity. To each* $\lambda_j = \kappa(\omega_d^j)$

Figure 4.7: Initial conditions

Figure 4.8: Temperature profile at $t = [0.0010.0050.010.05]$

there corresponds the eigenvector

$$\omega_j = \begin{bmatrix} 1 \\ \omega_d^j \\ \omega_d^{2j} \\ \vdots \\ \omega_d^{(d-1)j} \end{bmatrix}, \qquad j = 0, 1, ..., d-1.$$

Proof We show directly that $C(\kappa)\omega_j = \lambda_j \omega_{\mathbf{j}}$ for all $j = 0, 1, ..., d-1$. Observe that the mth component of $C(\kappa)\omega_j$ can be written as

$$\sum_{l=0}^{d-1} c_{m,l}\omega_{j,l} \sum_{l=0}^{m-1} \kappa_{d-m+l}\omega_d^{jl} + \sum_{l=m}^{d-1} \kappa_{l-m}\omega_d^{jl}.$$

Since $w_d^d = w_d^{-d} = 1$ we can rewrite this as follows

$$\begin{aligned} \sum_{l=0}^{d-1} c_{m,l}\omega_{j,l} &= \sum_{l=d-m}^{d-1} \kappa_l \omega_d^{j(l-d+m)} + \sum_{l=0}^{d-1-m} \kappa_l \omega_d^{j(l+m)} \\ &= \left(\sum_{l=0}^{d-1} \kappa_l \omega_d^{jl} \right) \omega_d^{jm} = \lambda_j \omega_{j,m}, \qquad m = 0, 1, ..., d-1. \end{aligned}$$

We conclude that the ω_j are indeed eigenvectors corresponding to the eigenvalues $\kappa(\omega_d^j)$, $j = 0, 1, ..., d-1$, respectively. We note that the eigenvectors depends only on the dimension d of the matrix, hence all $d \times d$ circulant matrices share the same eigenvectors, hence *all such matrices commute*. Also, the eigenvector matrix

$$\begin{bmatrix} \omega_{\mathbf{0}} & \omega_{\mathbf{1}} & \dots & \omega_{d-1} \end{bmatrix}$$

is *unitary* since it is trivial to prove that

$$\langle \omega_j, \omega_l \rangle = \bar{\omega}_j^T \omega_l = 0, \qquad j, l = 0, 1, ..., d-1, \quad j \neq l$$

Therefore each and every circulant matrix is *normal*. □

Consider the following semi-discrete equation

$$\frac{d\mathbf{u}}{dt} = A_1\mathbf{u} + A_2\mathbf{u} - \mathbf{f}$$

where A_1 and A_2 are circulant matrices. Since A_1 and A_2 are circulant they have the same complete eigensystem. To uncouple the equations we do a

spectraldecomposition. Premultiply with the left eigenvector matrix X^{-1} and use the identity $XX^{-1} = I$ to obtain the relation

$$\begin{aligned} X^{-1}\frac{d\mathbf{u}}{dt} &= X^{-1}A_1X \cdot X^{-1}\mathbf{u} + X^{-1}A_2X \cdot X^{-1}\mathbf{u} - X^{-1}\mathbf{f} \\ &= \Lambda_1 X^{-1}\mathbf{u} + \Lambda_2 X^{-1}\mathbf{u} - X^{-1}\mathbf{f} \end{aligned} \tag{4.22}$$

Finally, define the variables $\mathbf{w}$ and $\mathbf{g}$ such that

$$\mathbf{w} = X^{-1}\mathbf{u}, \qquad \mathbf{g} = X^{-1}\mathbf{f}$$

Then equation (4.22) can be written as

$$\frac{d\omega}{dt} = \Lambda_1\omega + \Lambda_2\omega - \mathbf{g} \tag{4.23}$$

The mth component of this equation is $\dot{\omega}_m = (\lambda_1 + \lambda_2)_m\omega_m - g_m(t)$, and the corresponding solution is given by

$$\omega_m(t) = c_m \mathrm{e}^{(\lambda_1+\lambda_2)_m t} + P.S$$

where P.S is the particular solution. Physically $\omega_m(t)$ is the evolution of the mth frequency mode. This frequency or wave solution is related to the real space solution: For any circulant system the following holds:

$\omega = X^{-1}\mathbf{u}$ is a discrete Fourier transform from real space to wave space (or eigenspace)

$\mathbf{u} = X\omega$ is a discrete Fourier synthesis from wave space back to real space.

Now consider the linear convection-diffusion equation

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}. \tag{4.24}$$

Let the semi-discrete approximation be

$$\frac{d\mathbf{u}}{dt} = -\frac{a}{2\Delta x}B_p(-1,0,1)\mathbf{u} + \frac{\nu}{\Delta x^2}B_p(1,-2,1)\mathbf{u} \tag{4.25}$$

where B_p is a circulant matrix operator. When the standard three-point central-differencing operators for the first and second derivative are used they take the form

$$B(-1,0,1)_p = \frac{1}{2\Delta x}\begin{bmatrix} 0 & 1 & & & -1 \\ -1 & 0 & 1 & & \\ & & \ddots & & \\ & & -1 & 0 & 1 \\ 1 & & & -1 & 0 \end{bmatrix}$$

and

$$B(1,-2,1)_p = \frac{1}{\Delta x^2}\begin{bmatrix} -2 & 1 & & & 1 \\ 1 & -2 & 1 & & \\ & & \ddots & & \\ & & 1 & -2 & 1 \\ 1 & & & 1 & -2 \end{bmatrix}$$

From the results in lemma 4.1 it can be shown that the eigenvalues λ_c and λ_d of the convection and diffusion matrix is given by

$$\begin{aligned} (\lambda_c)_m &= \frac{\mathrm{i}a}{\Delta x}\sin(\theta_m) \\ (\lambda_d)_m &= -\frac{4\nu}{\Delta x^2}\sin^2\frac{\theta_m}{2}. \end{aligned}$$

In these equations, $\theta_m = 2m\pi/N$, and $m = 0, 1, ..., N-1$, so that $\theta_m \leq \theta_m \leq 2\pi$.
Hopefully, we are now fit to do stability analysis of split and factored forms. First we scrutinize the explicit-implicit method

$$\begin{aligned} \tilde{\mathbf{u}}^{n+1} &= [I + kB_p(-1,0,1)]\mathbf{u}^n + h\mathbf{bc} \\ [I - kB_p(1,-2,1)]\,\mathbf{u}^{n+1} &= \tilde{\mathbf{u}}^{n+1}. \end{aligned}$$

Using the shift operator $\mathcal{E}$ we can write this equation as

$$\Big([I - kB_p(1,-2,1)]\mathcal{E} - [I + kB_p(-1,0,1)]\Big)\mathbf{u}^n = h\mathbf{bc}$$

A spectral decomposition of this equation yields the characteristic polynom

$$P(\mathcal{E}) = (1 - k\lambda_d)\mathcal{E} - (1 + k\lambda_c)$$

whose *principal* σ-root is given by

$$\sigma = \frac{1 + \mathrm{i}\frac{ak}{\Delta x}\sin\theta_m}{1 + 4\frac{k\nu}{\Delta x^2}\sin^2\frac{\theta_m}{2}}$$

The principal σ-root is the numerical approximation to $\mathrm{e}^{\lambda h}$. There is always one σ-root for every λ-root, in addition, the numerical approximation might produce *spurious* σ-roots. Spurious roots arise if a method uses data from time level $n-1$ or earlier to advance the solution from time level n to $n+1$. In other words, if the numerical approximation produces a characteristic polynomial of higher degree than the actual characteristic polynomial, there will be spurious roots. Spurious roots has nothing to do with the ODE being

solved.
If we introduce the dimensionless numbers

$$\begin{aligned} C_n &= ak\Delta x, \qquad \text{Courant number} \\ R_\Delta &= \frac{a\Delta x}{\nu}, \qquad \text{mesh Reynolds number} \end{aligned}$$

we can write the absolute value of σ as

$$|\sigma| = \frac{\sqrt{1 + C_n^2 \sin^2 \theta_m}}{1 + 4\frac{C_n}{R_\Delta} \sin^2 \frac{\theta_m}{2}} , \qquad 0 \leq \theta_m \leq 2\pi.$$

When θ_m is near zero, $|\sigma|$ has a maximum. We are interested in the condition on C_n and R_Δ that makes $|\sigma| \approx 1$, thus we have the relation

$$[1 + C_n^2 \sin^2 \epsilon] = \left[1 + 4\frac{C_n}{R_\Delta} \sin^2 \frac{\epsilon}{2}\right]^2 .$$

As $\epsilon \to 0$ this gives the stability region

$$C_n < \frac{2}{R_\Delta}$$

A Similar analysis of the explicit-explicit method

$$\begin{aligned} \tilde{\mathbf{u}}^{n+1} &= [I + kB_p(-1, 0, 1)] + h\mathbf{bc} \\ \mathbf{u}^{n+1} &= [I + kB_p(1, -2, 1)]\tilde{\mathbf{u}}^{n+1} \end{aligned}$$

reveals the following absolute value of the σ-root.

$$|\sigma| = \sqrt{1 + C_n^2 \sin^2 \theta_m} \left[1 - 4\frac{C_n}{R_\Delta} \sin^2 \frac{\theta_m}{2}\right] , \qquad 0 \leq \theta_m \leq 2\pi.$$

The critica values of θ_m are 0 and π. The conditions on C_n and R_Δ that makes $|\sigma| \approx 1$ are

$$\begin{aligned} C_n &< \frac{1}{2}R_\Delta \quad \text{for} \quad R_\Delta \leq 2 \\ C_n &< \frac{2}{R_\Delta} \end{aligned}$$

For more on the analysis of split and factored forms [2] is an excellent reference.

```
 % Program to solve the heat conduction problem
% with Neumann BC.
nx=48; ny=48; Lx=1; Ly=1; hx=Lx/(nx+1); hy=Ly/(ny+1);
alphax=1; alphay=alphax*(hx/hy)^2;
k=alphax*hx^2; x=(0:nx+1)*hx-Lx/2; y=(0:ny+1)*hy-Ly/2;
xdiff1=x+Lx/4; ydiff1=y-Ly/4; xdiff2=x-Lx/4; ydiff2=y+Ly/4;
% d_square is an nx times ny matrix containing the distance
% squared from the origin to each of the nx by ny points
d_squared=(xdiff1.^2)'*ones(1,ny+2)+ones(nx+2,1)*(ydiff1.^2);
u=1+2*exp(-10*d_squared);
d_squared=(xdiff2.^2)'*ones(1,ny+2)+ones(nx+2,1)*(ydiff2.^2);
u=u-exp(-100*d_squared);figure(1);mesh(x,y,u','EdgeColor','black');
uhalf=u; unew=u; iplot=1; tend=0.05; uplot=1;
for tend=[0.001 0.005 0.01 0.05]
nsteps=ceil(tend/k); nplots=25; plotstep=ceil(nsteps/nplots);
% Set up the arrays for the ADI method
ahalf=-(alphax/2)*ones(nx+2,1); bhalf=(1+alphax)*ones(nx+2,1);
bhalf(1)=1+(alphax/3); bhalf(nx+2)=1+(alphax/3); chalf=ahalf;
ahalf(nx+1)=-(alphax/3); chalf(2)=-(alphax/3);
A1=spdiags([ahalf,bhalf,chalf],[-1,0,1],nx+2,nx+2);
afull=-(alphay/2)*ones(ny+2,1); bfull=(1+alphay)*ones(ny+2,1);
bfull(1)=1+(alphay/3); bfull(ny+2)=1+(alphay/3);
cfull=afull;  cfull(2)=-(alphay/3); afull(ny+1)=-(alphay/3);
A2=spdiags([afull,bfull,cfull],[-1,0,1],ny+2,ny+2);
%Solve equations at nsteps time steps
for istep=1:nsteps
    for j=2:ny+1
        fhalf=(alphay/2)*u(1:nx+2,j-1)...
            +(alphay/2)*u(1:nx+2,j+1)+(1-alphay)*u(1:nx+2,j);
        utemp=A1\fhalf; uhalf(1:nx+2,j)=utemp;
    end
    % Solve equation for the insulated boundary y=-Ly/2
    fhalf=(alphay/2)*u(1:nx+2,2)...
        +(1-(alphay/2))*u(1:nx+2,1);
    utemp=A1\fhalf; uhalf(1:nx+2,1)=utemp;
    % Solve equation for the insulated boundary y=Ly/2
    fhalf=(alphay/2)*u(1:nx+2,ny+1)...
        +(1-(alphay/2))*u(1:nx+2,ny+2);
    utemp=A1\fhalf; uhalf(1:nx+2,ny+2)=utemp;
```

```
    % Solve equation at interior points
    for i=2:nx+1
        ffull=(alphax/2)*uhalf(i-1,1:ny+2)...
            +(alphax/2)*uhalf(i+1,1:ny+2)+...
            (1-alphax)*uhalf(i,1:ny+2);
        utemp=A2\ffull'; unew(i,1:ny+2)=utemp';
    end
    % Solve equation for the insulated boundary x=-Lx/2
    ffull=(alphax/2)*uhalf(2,1:ny+2)...
        +(1-(alphax/2))*uhalf(1,1:ny+2);
    utemp=A2\ffull'; unew(1,1:ny+2)=utemp';
   % Solve equation for the insulated boundary x=Lx/2
    ffull=(alphax/2)*uhalf(nx+1,1:ny+2)...
        +(1-(alphax/2))*uhalf(nx+2,1:ny+2);
    utemp=A2\ffull'; unew(nx+2,1:ny+2)=utemp'; u=unew; uhalf=u;
end
    figure(2); subplot(2,2,uplot); mesh(x,y,u','EdgeColor','black');
    uplot=uplot+1;
end
```

Chapter 5

The Fast Fourier Transform

For a positive integer n, the complex numbers $\{1, \omega, \omega^2, ..., \omega^{n-1}\}$, where

$$\omega = \mathrm{e}^{2\pi \mathrm{i}/n} = \cos\frac{2\pi}{n} + \mathrm{i}\sin\frac{2\pi}{n}$$

are called the **n**th *roots of unity* because they represent all the solutions to $z^n = 1$. Geometrically, they are the vertices of a regular polygon of n sides as depicted in figure 5 for $n = 3$ and $n = 6$. The roots of unity are cyclic

Figure 5.1: Roots of unity

in the the sense that $\omega^k = \omega^{k(\mathrm{mod}\ n)}$ where $k(\mathrm{mod}\ n)$ denotes the remainder when k is divided with n. For example, when $n = 6$, $\omega^6 = 1$, $\omega^7 = \omega$, ...
The numbers $\{1, \xi, \xi^2, ..., \xi^{n-1}\}$, where

$$\xi = \mathrm{e}^{-2\pi \mathrm{i}/n} = \cos\frac{2\pi}{n} - \mathrm{i}\sin\frac{2\pi}{n} = \bar{\omega}$$

are also the nth roots of unity, but they are now numbered clockwise as depicted in figure

We now derive two identities that will be useful in our derivation of the FFT. If k is an integer, then $1 = |\xi^k|^2 = \xi^k\bar{\xi^k}$ implies that

$$\xi^{-k} = \bar{\xi^k} = \omega^k. \tag{5.1}$$

Figure 5.2: Roots of unity

Furthermore, the fact that

$$\xi^k\Big(1+\xi^k+\xi^{2k}+\cdot\,..+\xi^{(n-2)k}+\xi^{(n-1)k}\Big)=\xi^k+\xi^{2k}+...+\xi^{(n-1)k}+1$$

implies $(1+\xi^k+\xi^{2k}+...+\xi^{(n-1)k})(1-\xi^k)=0$ and, consequently,

$$1+\xi^k+\xi^{2k}+...+\xi^{(n-1)k}=0 \quad \text{whenever} \quad \xi^k\neq 1 \tag{5.2}$$

Definition 5.1 (Fourier Matrix) *The $n\times n$ matrix whose (j,k)-entry is $\xi^{jk}=\omega^{-jk}$ for $0\leq j,k\leq n-1$ is called the* Fourier Matrix *of order n, and it has the form*

$$\mathbf{F}_n=\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \xi & \xi^2 & \dots & \xi^{n-1} \\ 1 & \xi^2 & \xi^4 & \dots & \xi^{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \xi^{n-1} & \xi^{n-2} & \dots & \xi \end{bmatrix}$$

The Fourier matrix is a special case of the Vandermonde matrix. If we take the inner product of any two columns in F_n, say, the rth and sth, there results

$$\mathbf{F}_{*,r}^*\mathbf{F}_{*,s}=\sum_{j=0}^{n-1}\bar{\xi}^{jr}\xi^{js}=\sum_{j=0}^{n-1}\xi^{-jr}\xi^{js}=\sum_{j=0}^{n-1}\xi^{j(s-r)}=0.$$

This follows from the equaitons (5.1) and (5.2). In other words, the columns in $\mathbf{F}$ are mutually orthogonal. Furthermore, each column iF_n has norm $\sqrt{n}$ because

$$\|\mathbf{F}_{*k}\|_2^2=\sum_{j=0}^{n-1}|\xi^{jk}|^2=\sum_{j=0}^{n-1}1=n,$$

Consequently $(1/\sqrt{n})\mathbf{F}_n$ is a unitary matrix. Since it is also true that $\mathbf{F}_n^T = \mathbf{F}_n$, we have

$$\left(\frac{1}{\sqrt{n}}\mathbf{F}_n\right)^{-1} = \left(\frac{1}{\sqrt{n}}\right)^* = \frac{1}{\sqrt{n}}\bar{\mathbf{F}}_n,$$

and therefore $\mathbf{F}_n^{-1} = \bar{\mathbf{F}}_/n$. Since equation (5.1) says that $\bar{\xi}^k = \omega^k$, it follows that

$$\mathbf{F}_n^{-1} = \frac{1}{n}\bar{\mathbf{F}}_n = \frac{1}{n}\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{n-2} & \dots & \omega \end{bmatrix}$$

5.1 The Discrete Fourier Transform

Given a vector $\mathbf{x}_{n\times 1}$ the product $\mathbf{F}_n\mathbf{x}$ is called the *discrete Fourier transform* of $\mathbf{x}$, and $\mathbf{F}^{-1}$ is called the *inverse Fourier transform* of $\mathbf{x}$. The kth entries in $\mathbf{F}_n\mathbf{x}$ and $\mathbf{F}_n^{-1}\mathbf{x}$ are given by

$$[\mathbf{F}_n\mathbf{x}]_k = \sum_{j=0}^{n-1} x_j\xi^{jk} \qquad \text{and} \qquad [\mathbf{F}_n^{-1}\mathbf{x}]_k = \frac{1}{n}\sum_{j=0}^{n-1} x_j\omega^{jk}. \tag{5.3}$$

An algorithm that computes the discrete fourier transform of a vector $\mathbf{x}$ can also be used to compute the inverse transform of $\mathbf{x}$. Let us call such an algorithm FFT. Since

$$\mathbf{F}_n^{-1}\mathbf{x} = \frac{\bar{\mathbf{F}}_n\mathbf{x}}{n} = \frac{\overline{\mathbf{F}_n\bar{\mathbf{x}}}}{n}$$

FFT will return the inverse transform of $\mathbf{x}$ by executing the following three steps

1. $\mathbf{x} \longleftarrow \bar{\mathbf{x}}$ (compute$\bar{\mathbf{x}}$)
2. $\mathbf{x} \longleftarrow \text{FFT}(\mathbf{x})$ (compute $\mathbf{F}_n\bar{\mathbf{x}}$).
3. $\mathbf{x} \longleftarrow (1/n)\bar{\mathbf{x}}$ (compute $n^{-1}\overline{\mathbf{F}_n\bar{\mathbf{x}}} = \mathbf{F}_n^{-1}\mathbf{x}$).

To perform a discrete Fourier transform of order $2n$ by standard matrix-vector multiplication requires $4n^2$ scalar multiplications. It was not until 1965 that two the two Americans, J. Cooley and J. W. Tukey, came up with the ingenious *fast Fourier transform* (FFT), an algorithm that requires only on the order of $(n/2)\log_2 n$ scalar multiplications to compute $\mathbf{F}_n$. The magic

of the fast Fourier transform algorithm emanates from the fact that if n is a power of two, then a discrete Fourier transform of order n can be executed by performing two transforms of order $n/2$. To illustrate this note that when $^n = 2^r$ we have $(\xi^j)^n = (\xi^{2j})^{n/2}$, so

$$\{1, \xi, \xi^2, \xi^3, ..., \xi^{n-1}\} = \text{ the } n\text{th roots of unity}$$

$$\text{if and only if}$$

$$\{1, \xi^2, \xi^4, \xi^6, ..., \xi^{n-2}\} = \text{the } (n/2)\text{th roots of unity.}$$

This means that the (j,k)-entries on the Fourier matrices $\mathbf{F}_n$ and $\mathbf{F}_{n/2}$ are

$$[\mathbf{F}_n]_{jk} = \xi^{jk} \quad \text{and} \quad [\mathbf{F}_{n/2}]_{jk} = (\xi^2)^{jk} = \xi^{2jk}. \tag{5.4}$$

If the columns of $\mathbf{F}_n$ are permuted so that columns with even subscripts are listed before those with odd subscript, and if $\mathbf{P}_n^T$ is the corresponding permutation matrix, then we can partition $\mathbf{F}_n\mathbf{P}_n^T$ as

$$\mathbf{F}_n\mathbf{P}_n^T = [\mathbf{F}_{*0}\mathbf{F}_{*2}...\mathbf{F}_{*n-2}|\mathbf{F}_{*1}\mathbf{F}_{*3}...\mathbf{F}_{*n-1}] = \begin{bmatrix} \mathbf{A}_{\frac{n}{2}\times\frac{n}{2}} & \mathbf{B}_{\frac{n}{2}\times\frac{n}{2}} \\ \mathbf{C}_{\frac{n}{2}\times\frac{n}{2}} & \mathbf{D}_{\frac{n}{2}\times\frac{n}{2}} \end{bmatrix}$$

By using equation (5.4) together with the fact that

$$\xi^{nk} = 1 \quad \text{and} \quad \xi^{n/2} = \cos\frac{2\pi(n/2)}{n} - \mathrm{i}\sin\frac{2\pi(n/2)}{n} = -1$$

we see that the entries in A, B, C, and G are

$$\begin{aligned}
\mathbf{A}_{jk} &= \mathrm{F}_{j,2k} = \xi^{2jk} = [\mathbf{F}_{n/2}]_{jk}, \\
\mathbf{B}_{jk} &= \mathbf{F}_{j,2k+1} = \xi^{j(2k+1)} = \xi^j\xi^{2jk} = \xi^j[\mathbf{F}_{n/2}]_{jk}, \\
\mathbf{C}_{jk} &= \mathbf{F}_{\frac{n}{2}+j,2k} = \xi^{(\frac{n}{2}+j)2k} = \xi^{nk}\xi^{2jk} = \xi^{2jk} = [\mathbf{F}_{n/2}]_{jk}, \\
\mathbf{G}_{jk} &= \mathbf{F}_{\frac{n}{2}+j,2k+1} = \xi^{(\frac{n}{2}+j)(2k+1)} = \xi^{nk}\xi^{n/2}\xi^j\xi^{2jk} = -\xi^j\xi^{2jk} = [\mathbf{F}_{n/2}]_{jk}.
\end{aligned}$$

In other words, if $\mathbf{D}_{n/2}$ is the diagonal matrix

$$\mathbf{D}_{n/2} = \begin{bmatrix} 1 & & & & \\ & \xi & & & \\ & & \xi^2 & & \\ & & & \ddots & \\ & & & & \xi^{\frac{n}{2}-1} \end{bmatrix}$$

then

$$\mathbf{F}_n\mathbf{P}_n^T = \begin{bmatrix} \mathbf{F}_{n/2} & \mathbf{D}_{n/2}\mathbf{F}_{n/2} \\ \mathbf{F}_{n/2} & -\mathbf{D}_{n/2}\mathbf{F}_{n/2} \end{bmatrix} \tag{5.5}$$

The decomposition in (5.5) says that a discrete Fourier transform of order 2^r can be accomplished by two Fourier transfoms of order $n/2 = 2^{r-1}$, and this leads to the FFT algorithm. To illustrate how the FFT works, consider the case when $n = 8$. If

$$\mathbf{x}_8 = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \end{bmatrix}^T,$$

then

$$\mathbf{P}_8\mathbf{x}_8 = \begin{bmatrix} x_0 & x_2 & x_4 & x_6 & | & x_1 & x_3 & x_5 & x_7 \end{bmatrix}^T = \begin{bmatrix} \mathbf{x}_4^{(0)} \\ - \\ \mathbf{x}_4^{(1)} \end{bmatrix}$$

so

$$\mathbf{F}_8\mathbf{x}_8 = \begin{bmatrix} \mathbf{F}_4 & \mathbf{D}_4\mathbf{F}_4 \\ \mathbf{F}_4 & -\mathbf{D}_4\mathbf{F}_4 \end{bmatrix} \begin{bmatrix} \mathbf{x}_4^{(0)} \\ \mathbf{x}_4^{(1)} \end{bmatrix} = \begin{bmatrix} \mathbf{F}_4\mathbf{x}_4^{(0)} + \mathbf{D}_4\mathbf{F}_4\mathbf{x}_4^{(1)} \\ \mathbf{F}_4\mathbf{x}_4^{(0)} - \mathbf{D}_4\mathbf{F}_4\mathbf{x}_4^{(1)} \end{bmatrix} \tag{5.6}$$

But

$$\mathbf{P}_4\mathbf{x}_4^{(0)} = \begin{bmatrix} x_0 \\ x_4 \\ - \\ x_2 \\ x_6 \end{bmatrix} = \begin{bmatrix} \mathbf{x}_2^{(0)} \\ - \\ \mathbf{x}_2^{(1)} \end{bmatrix} \quad \text{and} \quad \mathbf{P}_4\mathbf{x}_4^{(1)} = \begin{bmatrix} x_1 \\ x_5 \\ - \\ x_3 \\ x_7 \end{bmatrix} = \begin{bmatrix} \mathbf{x}_2^{(2)} \\ - \\ \mathbf{x}_2^{(3)} \end{bmatrix}$$

so

$$\mathbf{F}_4\mathbf{x}_4^{(0)} = \begin{bmatrix} \mathbf{F}_2 & \mathbf{D}_2\mathbf{F}_2 \\ \mathbf{F}_2 & -\mathbf{D}_2\mathbf{F}_2 \end{bmatrix} \begin{bmatrix} \mathbf{x}_2^{(0)} \\ \mathbf{x}_2^{(1)} \end{bmatrix} = \begin{bmatrix} \mathbf{F}_2\mathbf{x}_2^{(0)} + \mathbf{D}_2\mathbf{F}_2\mathbf{x}_2^{(1)} \\ \mathbf{F}_2\mathbf{x}_2^{0} - \mathbf{D}_2\mathbf{F}_2\mathbf{x}_2^{(1)} \end{bmatrix} \tag{5.7}$$

and

$$\mathbf{F}_4\mathbf{x}_4^{(0)} = \begin{bmatrix} \mathbf{F}_2 & \mathbf{D}_2\mathbf{F}_2 \\ \mathbf{F}_2 & -\mathbf{D}_2\mathbf{F}_2 \end{bmatrix} \begin{bmatrix} \mathbf{x}_2^{(2)} \\ \mathbf{x}_2^{(3)} \end{bmatrix} = \begin{bmatrix} \mathbf{F}_2\mathbf{x}_2^{(2)} + \mathbf{D}_2\mathbf{F}_2\mathbf{x}_2^{(3)} \\ \mathbf{F}_2\mathbf{x}_2^{2} - \mathbf{D}_2\mathbf{F}_2\mathbf{x}_2^{(3)} \end{bmatrix}$$

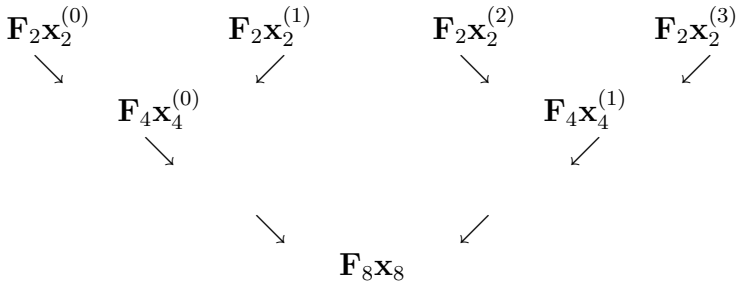
Now, since $\mathbf{F}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, it is trivial to compute the terms

$$\mathbf{F}_2\mathbf{x}_2^{(0)},\ \mathbf{F}_2\mathbf{x}_2^{(1)},\ \mathbf{F}_2\mathbf{x}_2^{(2)},\ \mathbf{F}_2\mathbf{x}_2^{(3)}.$$

To actually carry out the computation, we need to work backward through the preceding sequences of steps. That is we start with

$$\begin{aligned} \tilde{\mathbf{x}}_8 &= \begin{bmatrix} \mathbf{x}_2^{(0)} & | & \mathbf{x}_2^{(1)} & | & \mathbf{x}_2^{2} & | & \mathbf{x}_2^{(3)} \end{bmatrix}^T \\ &= \begin{bmatrix} x_0 & x_4 & | & x_2 & x_6 & | & x_1 & x_5 & | & x_3 & x_7 \end{bmatrix}^T \end{aligned}$$

and use the equations (5.6) and (5.7) to work downward in the following three.



In order to work downward through this tree, we cannot start directly with $\mathbf{x}_8$, we must start with the permutation $\tilde{\mathbf{x}_8}$. The entries in $\tilde{\mathbf{x}_8}$ where obtained by first sorting the x_js into two groups-the entries in the even were separated from those in the odd positions. Then each grop was broken into two more groups by again separating the entries in the even position from those in the odd positions

$$\begin{array}{cccccccc} \big(0 & 1 & 2 & 3 & 4 & 5 & 6 & 7\big) \\ & & & \swarrow & \searrow & & & \\ \big(0 & 2 & 4 & 6\big) & \big(1 & 3 & 5 & 7\big) \\ & \swarrow & \searrow & & & \swarrow & \searrow & \\ \big(0 & 4\big) & \big(2 & 6\big) & \big(1 & 5\big) & \big(3 & 7\big) \end{array} \tag{5.8}$$

In general, this even-odd sorting process (sometimes called a *perfect shuffle*) produces the permutation necessary to initiate the algorithm. A clever way to do perform a perfect shuffle is to use binary representations and observe that the first level of sorting in (5.8) is determined according to wether the least significant bit is 0 or 1, the second level of sorting is determined by the second least significant bit, and so on.

Natural order	**First level**	**Second level**
$0 \leftrightarrow 000$	$0 \leftrightarrow 00\mathbf{0}$	$0 \leftrightarrow 0\mathbf{0}0$
$1 \leftrightarrow 001$	$2 \leftrightarrow 01\mathbf{0}$	$\underline{4 \leftrightarrow 1\mathbf{0}0}$
$2 \leftrightarrow 010$	$4 \leftrightarrow 10\mathbf{0}$	$2 \leftrightarrow 0\mathbf{1}0$
$3 \leftrightarrow 011$	$\underline{6 \leftrightarrow 11\mathbf{0}}$	$\underline{6 \leftrightarrow 1\mathbf{1}0}$
$4 \leftrightarrow 100$	$1 \leftrightarrow 00\mathbf{1}$	$1 \leftrightarrow 0\mathbf{0}1$
$5 \leftrightarrow 101$	$3 \leftrightarrow 01\mathbf{1}$	$\underline{5 \leftrightarrow 1\mathbf{0}1}$
$6 \leftrightarrow 110$	$5 \leftrightarrow 10\mathbf{1}$	$3 \leftrightarrow 0\mathbf{1}1$
$7 \leftrightarrow 111$	$7 \leftrightarrow 11\mathbf{1}$	$7 \leftrightarrow 1\mathbf{1}1$

But all intermediate levels in this sorting process can be eliminated because

something very nice occurs. Examination of the last column in Table 5.1 reveals that the binary bits in the perfect shuffle ordering are exactly the reversal of the binary bits in the natural ordering. In other words, to generate the perfect shuffle of the numbers $0, 1, 2, ..., n-1$, simply reverse the bits in the binary representation of each number. We can summarize the fast Fourier transform as follows: For a given input vector $\mathbf{x}$ containing $n = 2^r$ components, the discrete Fourier transform $\mathbf{F}_n\mathbf{x}$ is the result of successively creating the following arrays.

$$\mathbf{X}_{1\times n} \longleftarrow \text{rev}(\mathbf{x}) \qquad \text{(bit reverse the subscripts)}$$

For $j = 0, 1, 2, ..., r-1$

$$\mathbf{D} \longleftarrow \begin{bmatrix} 1 \\ e^{-\pi i/2^j} \\ e^{-2\pi i/2^j} \\ \vdots \\ e^{-2^j-1)\pi i/2^j} \end{bmatrix}_{j+1\times 1}$$

$$\mathbf{X}^{(0)} \longleftarrow \left(\mathbf{X}_{*0}\ \mathbf{X}_{*2}\ \mathbf{X}_{*4}\ ...\mathbf{X}_{*2^{r-j}-2}\right)_{2^j\times 2^{r-j-1}}$$

$$\mathbf{X}^{(1)} \longleftarrow \left(\mathbf{X}_{*1}\ \mathbf{X}_{*3}\ \mathbf{X}_{*5}\ ...\mathbf{X}_{*2^{r-j}-1}\right)_{2^j\times 2^{r-j-1}}$$

$$\mathbf{X} \longleftarrow \begin{bmatrix} \mathbf{X}^{(0)} + \mathbf{D}\times\mathbf{X}^{(1)} \\ \mathbf{X}^{(0)} - \mathbf{D}\times\mathbf{X}^{(1)} \end{bmatrix} \quad \times \text{ denotes entry-by- entry product}$$

5.2 Fast Poisson solvers

We now show how the FFT can be used to solve the discretized Poisson equatoin. In the pressent section we assume that the Poisson equation with Dirichlet boundary coditions is solved with in a rectangle with either the five-point formula

$$\frac{1}{(\Delta x)^2}(\Delta_{0,x}^2 + \Delta_{0,y}^2)u_{k,l} = f_{k,l} \tag{5.9}$$

or the nine-point operator

$$\frac{1}{\Delta x^2}(\Delta_{0,x}^2 + \Delta_{0,y}^2 + \frac{1}{6}\Delta_{0,x}^2\Delta_{0,y}^2)u_{k,l} = f_{k,l}, \tag{5.10}$$

or, for that matter, the modified nine-point operator

$$\frac{1}{(\Delta x)^2}(\Delta_{0,x}^2 + \Delta_{0,y}^2 + \frac{1}{6}\Delta_{0,x}^2\Delta_{0,y}^2)u_{k,l} = [\mathcal{I} + \frac{1}{12}(\Delta_{0,x}^2 + \Delta_{0,y}^2)]f_{k,l}. \tag{5.11}$$

In either case we assume that the equations have been assembled in *natural ordering*.
The resulting linear system $A\mathbf{x} = \mathbf{b}$ can be written in a block-TST form. For a $m_1 \times m_2$ grid there results

$$\begin{bmatrix} S & T & O & \dots & O \\ T & S & T & \ddots & \vdots \\ O & \ddots & \ddots & \ddots & O \\ \vdots & \ddots & T & S & T \\ O & \dots & O & T & S \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{m_2} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_{m_2} \end{bmatrix}, \tag{5.12}$$

For a column-wise ordering $\mathbf{x}_l$ and $\mathbf{b}_l$ correspond to the variables and to the portion of $\mathbf{b}$ along the lth column of the grid, respectively:

$$\mathbf{x}_l = \begin{bmatrix} u_{1,l} \\ u_{2,l} \\ \vdots \\ u_{m_1,l} \end{bmatrix}, \qquad \mathbf{b}_l = \begin{bmatrix} b_{1,l} \\ b_{2,l} \\ \vdots \\ b_{m_1,l} \end{bmatrix} \qquad l = 1, 2, ..., m_2$$

Both S and T are themselves $m_1 \times m_1$ TST matrices:

$$S = \begin{bmatrix} -4 & 1 & 0 & \dots & 0 \\ 1 & -4 & 1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 1 & -4 & 1 \\ 0 & \dots & 0 & 1 & -4 \end{bmatrix} \qquad \text{and} \qquad T = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 1 \end{bmatrix}$$

for the five-point formula and

$$S = \begin{bmatrix} -\frac{10}{3} & \frac{2}{3} & 0 & \dots & 0 \\ \frac{2}{3} & -\frac{10}{3} & \frac{2}{3} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \frac{2}{3} & -\frac{10}{3} & \frac{2}{3} \\ 0 & \dots & 0 & \frac{2}{3} & -\frac{10}{3} \end{bmatrix} \qquad \text{and} \qquad T = \begin{bmatrix} \frac{2}{3} & \frac{1}{6} & 0 & \dots & 0 \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ 0 & \dots & 0 & \frac{1}{6} & \frac{2}{3} \end{bmatrix}$$

in the case of the nine-point formula. We rewrite (5.12) in the form

$$T\mathbf{x}_{l-1} + S\mathbf{x}_l + T\mathbf{x}_{l+1} = \mathbf{b}_l, \qquad l = 1, 2, ..., m_2, \tag{5.13}$$

where $\mathbf{x}_0, \mathbf{x}_{m_2+1} := \mathbf{0} \in \mathbb{R}^{m_1}$. We know that the eigenvalues and eigenvectors of a TST matrix depends only on its dimension. In particular, the spectral decomposition of S and T are given by

$$S = QD_SQ, \qquad T = QD_TQ, \tag{5.14}$$

where

$$q_{j,l} = \sqrt{\frac{2}{m_1+1}} \sin\Big(\frac{\pi jl}{m_1+1}\Big), \qquad j,l = 1,2,...,m_1$$

This follows directly from Lemma (2.1), and the fact that Q is orthogonal and symmetric ($S = QD_SQ^{-1} = QD_SQ^T = QD_SQ$). The $m_1 \times m_1$ matrices D_S and D_T are diagonal and their entries contain the eigenvalues of S and T respectively. We substitute (5.14) into (5.13) and premultiply with $Q = Q^{-1}$. The outcome is

$$D_T\mathbf{y}_{l-1} + D_S\mathbf{y}_l + D_T\mathbf{y}_{l+1} = \mathbf{c}_l, \qquad l = 1,2,...,m_2, \tag{5.15}$$

where

$$\mathbf{y}_l := Q\mathbf{x}_l, \quad \mathbf{c}_l := Q\mathbf{b}_l, \quad l = 1,2,...,m_2.$$

We observe that we now have a diagonal matrices rather than TST, matrices. We also observe that the bandwidth is m_1. We can reduce the bandwidth of this matrix. Let us consider the first equation in each of the m_2 blocks

$$\begin{aligned}
\lambda_1^{(S)}y_{1,1} + \lambda_1^{(T)}y_{1,2} &= c_{1,1} \\
\lambda_1^{(T)}y_{1,l-1} + \lambda_1^{(S)}y_{1,l} + \lambda_1^{(T)}y_{1,l+1} &= c_{1,l} \qquad l = 2,3,...,m_2-1, \\
\lambda_1^{(T)}y_{1,m_2-1} + \lambda_1^{(S)}y_{1,m_2} &= c_{1,m_2}.
\end{aligned}$$

If we reorder the $\mathbf{y}_l$ and the $\mathbf{c}_l$ *by rows*,

$$\tilde{\mathbf{y}}_j := \begin{bmatrix} y_{j,1} \\ y_{j,2} \\ \vdots \\ y_{j,m_2} \end{bmatrix}, \quad \tilde{\mathbf{c}}_j := \begin{bmatrix} c_{j,1} \\ c_{j,2} \\ \vdots \\ c_{j,m_2} \end{bmatrix}, \qquad j = 1,2,...,m_1$$

then we can write

$$\begin{bmatrix}
\lambda_1^{(S)} & \lambda_1^{(T)} & 0 & \dots & 0 \\
\lambda_1^{(T)} & \lambda_1^{(S)} & \lambda_1^{(T)} & \ddots & \vdots \\
0 & \ddots & \ddots & \ddots & 0 \\
\vdots & \ddots & \lambda_1^{(T)} & \lambda_1^{(S)} & \lambda_1^{(T)} \\
0 & \dots & 0 & \lambda_1^{(T)} & \lambda_1^{(S)}
\end{bmatrix}$$

Likewise, the equation for the jth row can be written as

$$\Gamma_j \tilde{\mathbf{y}}_j = \tilde{\mathbf{c}}_j \qquad j = 1, 2, ..., m_1, \tag{5.16}$$

where

$$\Gamma_j := \begin{bmatrix} \lambda_j^{(S)} & \lambda_j^{(T)} & 0 & \dots & 0 \\ \lambda_j^{(T)} & \lambda_j^{(S)} & \lambda_j^{(T)} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \lambda_j^{(T)} & \lambda_j^{(S)} & \lambda_j^{(T)} \\ 0 & \dots & 0 & \lambda_j^{(T)} & \lambda_j^{(S)} \end{bmatrix}, \qquad j = 1, 2, ..., m_2.$$

Hence, *switching from column-wise to row-wise ordering cuncouples the* $(m_1 m_2) \times (m_1 m_2)$ *system (5.15) into* m_2 *systems, each of dimension* $m_1 \times m_1$. The matrices Γ_j are also TST, and the each of the m_2 linear systems (5.16) can be solved with a banded LU factorization at the cost of $\mathcal{O}(m_1)$ operations. Having solved the tridiagonal systems for the vectors $\tilde{\mathbf{y}}_1, \tilde{\mathbf{y}}_2, ..., \tilde{\mathbf{y}}_{m_1}$, which we then permute back to $\tilde{\mathbf{y}}_1, \tilde{\mathbf{y}}_2, ..., \tilde{\mathbf{y}}_{m_2}$. Finally, we obtain x_l from the matrix-vector product $\mathbf{x}_l = Q\mathbf{y}_l,\ l = 1, 2, ..., m_2$. This product can be obtained at by use of the FFT.
Let us summarize by reviewing the stages in the fast Poisson sollver

- To start with we have the vectors $\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_{m_2} \in \mathbb{R}^{m_1}$ and we form the products $\mathbf{c}_l = Q\mathbf{b}_l,\ l = 1, 2, ..., m_2$. This costs $\mathcal{O}(\frac{m_1 m_2}{2} \log_2 m_1)$ operations if the FFT is used.

- Change the ordering from column-wise to row-wise. That is, rearrange the columns $\mathbf{c}_l \in \mathbb{R}^{m_1}, l = 1, 2, ..., m_2$, into the row vectors $\tilde{\mathbf{c}}_j = \in \mathbb{R}^{m_2}, j = 1, 2, ..., m_2$. This reordering is just a change of notaion, and, in principle, free of any computational cost.

- The tridiagonal systems $\Gamma_j \tilde{\mathbf{y}}_j = \tilde{\mathbf{c}}_j$, are solved by banded LU factorization at the cost of $\mathcal{O}(m_1 m_2)$ operations.

- Reorder the vectors once again. This time we rearrange rows into columns, i.e $\tilde{\mathbf{y}}_j \in \mathbb{R}^{m_2}, j = 1, 2, ..., m_1$ into $y_l \in \mathbb{R}^{m_1}, l = 1, 2, ..., m_2$

- Solve the system of equations $\mathbf{x}_l = Q\mathbf{y}_l, l = 1, 2, ..., m_2$. If the FFT is hired the cost is $\mathcal{O}(\ fracm_1 m_2 2 \log_2 m_1$ operations

Now that we know the theory, it is good practice to run a few simple programs. Our first test problem is the Laplace equation $\Delta u = 0$ on the unit

square $0 \leq x, y \leq 1$. The boundary conditions are

$$\begin{aligned} u(x,0) &= 0, & u(x,1) &= \frac{1}{(1+x)^2+1} \\ u(0,y) &= \frac{y}{1+y^2}, & u(1,y) &= \frac{y}{4+y^2} \end{aligned}$$

The exact solution of this problem is $u = \frac{y}{(1+x)^2+y^2}$.

Figure 5.3: Error of the five-point (first row) and nine-point (second row) scheme applied to the the Laplace equation when $m = [11, 23, 47]$. Note that the five-point scheme caonveges at a rate $\mathcal{O}(h^2)$ while the nine-point scheme converges at a rate $\mathcal{O}(h^4)$

Next we consider the Poisson equation $\Delta u = \mathrm{e}^{xy},\ 0 \leq x, y \leq 1$ wit the boundary conditions

$$\begin{aligned} u(x,0) &= \frac{1}{2}\sin(6\pi x) & u(x,1) &= \sin(\pi 2x) \\ u(0,y) &= 5(y^2 - y) & u(1,y) &= -y(y-1)^4 \end{aligned}$$

Figure 5.4: The solution of the poisson equation using a modified nine-point scheme and m=60.

And finally we solve the poisson problem $\Delta u = x^2+y^2$ with the boundary conditions

$$\begin{aligned} u(x,0) &= 0 & u(x,1) &= \frac{1}{2}x^2 \\ u(0,y) &= \sin \pi y & u(1,y) &= \mathrm{e}^{\pi} \sin \pi y + \frac{1}{2}y^2 \end{aligned}$$

The exact solution of this problem is $\mathrm{e}^{\pi x} \sin \pi y + \frac{1}{2}(xy)^2$. From the plots we see that the error associated with the five-point and nine-point scheme decays by roughly four with each halving of h. The error costant is however less for the nine-point scheme. The modified scheme decays roughly with 64 whenever h is halved, which correspond to a accuracy of $\mathcal{O}(h^4)$

Figure 5.5: The exact solution of the second test problem

Figure 5.6: The error for the five-point, nine-point, and modified nine-point schemes

```
% Fast poisson solver in a square with
% continuous Dirichlet bc.
 clear; endpt=1; N=9; splot=4;
% mxm internal nodes. endpt is the endpoint of x and y.
% N=5 for the five-point scheme. N=9 and N=10 for
% the 9 and modified 9-point scheme respectively.
for m=[11 23 47]
h=endpt/(m+1); x=0:h:endpt; y=x;
[gb,gt,gl,gr]=dirichletBc(x,y);
[x,y]=meshgrid(x,y); f=ffunc(x,y); [Gamma]=formGamma(m,N);
b=formRhs(m,h,f,gb,gt,gl,gr,N); bFourier=fastSineTransform(m,b)';
% Reorder c by rows. Find the Fourier modes by solving
% a tridiagonal system of equations .........
BFourier=reshape(bFourier,m,m); BFourier=BFourier';
bFourier=BFourier(:); uFourier=bandSolve(Gamma,bFourier);
% Reorder uFourier by columns. reshape is a Matlab function
UFourier=reshape(uFourier,m,m); UFourier=UFourier';
uFourier=UFourier(:); u=fastSineTransform(m,uFourier)';
% Rearange x into matrix format, add boundary values,
% and plot solution
for j=1:m
    Unew(1:m,j)=u((j-1)*m+1:(j-1)*m+m);
end
Unew=[gb(2:length(gt)-1);Unew;gt(2:length(gb)-1)]; Unew=[gl',Unew,gr']
%figure(1); mesh(x,y,Unew,'EdgeColor','black');
%figure(2); mesh(x,y,Uexact,'EdgeColor','black');
%xlabel('x'); ylabel('y'); zlabel('Solution, u ');
Uexact=y./((1+x).^2+y.^2); figure(1); subplot(2,3,splot)
mesh(x,y,abs(Unew-Uexact),'EdgeColor','black');  splot=splot+1;
end
```

```
 function s=fastSineTransform(m,p)
 const=-sqrt(2/(m+1));
 for k=1:m
    % Take the kth vector from the long vector, p,
    % and set it to q. Take the 2m+2 fft of [0;q];
    q=p((k-1)*m+1:k*m); w=const*(fft([0;q],2*m+2));
    % Take the imaginary part of w. We dont need
    % the first element and last m+1 elements
    s((k-1)*m+1:k*m)=imag(w(2:m+1));
 end

 % rhs of the Poisson problem
 function f=ffunc(x,y)
 f=exp(x.*y);

 % Sparse LU solver of Ax=b
function [x]=bandSolve(A,b)
[m,n]=size(A); % Square matrix for simplicity
if m~=n
    error('Matrix not square');
end
% Find the band of A
for i=1:n
    if  A(m,i)~=0
        s=n-i; break;
    end
end
% Do the sparse LU decomposition for a matrix of band s
for j=1:m
    if j>=m-s
        L(j:m,j)=A(j:m,j)./A(j,j);
        U(j,j:m)=A(j,j:m);
        A(j:m,j:m)=A(j:m,j:m)-L(j:m,j)*U(j,j:m);
    else
        L(j:j+s,j)=A(j:j+s,j)./A(j,j);
        U(j,j:j+s)=A(j,j:j+s);
        A(j:j+s,j:j+s)=A(j:j+s,j:j+s)-L(j:j+s,j)*U(j,j:j+s);
    end
end
L=sparse(L); U=sparse(U); y=L\b; x=U\y;
```

```
 function b=formRhs(m,h,f,gb,gt,gl,gr,N)
g=f(2:m+1,2:m+1); len=length(g(:));
% fbc is the correction vector we will subtract
% from f due to boundary conditions
if N==5
    fbc=gl(2:length(gl)-1);
    fbc(len-m+1:len)=gr(2:length(gr)-1);
    for k=1:m
        fbc((k-1)*m+1)=fbc((k-1)*m+1)+gb(k+1);
        fbc(m*k)=fbc(m*k)+gt(k+1);
    end
    fbc=fbc';
elseif N==9 | N==10
    fbc=zeros(len,1);
    for k=1:m
        fbc(k)=(1/6)*gl(k)+(2/3)*gl(k+1)+...
            (1/6)*gl(k+2);
        fbc(len-m+k)=(1/6)*gr(k)+(2/3)*gr(k+1)+...
            (1/6)*gr(k+2);
    end
    for k=1:m
        fbc((k-1)*m+1)=fbc((k-1)*m+1)+...
            (1/6)*gb(k)+(2/3)*gb(k+1);
        fbc(m*k)=fbc(m*k)+(1/6)*gt(k)+...
            (2/3)*gt(k+1)+(1/6)*gt(k+2);
    end
    fbc(1)=fbc(1)-(1/6)*gl(1);
    fbc(m)=fbc(m)-(1/6)*gl(m+2);
    fbc(len-m+1)=fbc(len-m+1)-(1/6)*gr(1);
    fbc(end)=fbc(end)-(1/6)*gr(end);
end
f=modifyRhs(f,m,N);
b=(h^2).*f-fbc;
```

```
 function [Gamma]=formGamma(m,N)
for j=1:m
    for k=1:m
        Q(j,k)=sqrt(2/(m+1))*sin(pi*j*k/(m+1));
    end
end
if N==9 | N==10
    S=diag((-10/3)*ones(m,1))+diag((2/3)*ones(m-1,1),1)+...
        diag((2/3)*ones(m-1,1),-1);
    T=diag((2/3)*ones(m,1))+diag((1/6)*ones(m-1,1),1)+...
        diag((1/6)*ones(m-1,1),-1);
elseif N==5
    S=diag((-4)*ones(m,1))+diag((1)*ones(m-1,1),1)+...
        diag((1)*ones(m-1,1),-1);
    T=diag((1)*ones(m,1));
elseif N==11
    S=diag((2/3)*ones(m,1))+diag((1/12)*ones(m-1,1)+...
        diag((1/12)*ones(m-1,1),-1));
    T=diag((1/12)*ones(m,1));
end

lambdaS=diag(Q*S*Q); lambdaT=diag(Q*T*Q);
Gamma = sparse(zeros(size(Q)));
for j=1:m
    temp=(j-1)*m+1;
    Gamma(temp:j*m,temp:j*m)=diag(lambdaS(j)*ones(m,1))...
        +diag((lambdaT(j))*ones(m-1,1),1)+...
        diag((lambdaT(j))*ones(m-1,1),-1);
end
Gamma=sparse(Gamma);
```

```
 function f=modifyRhs(f,m,N)
if N==5|N==9
    f=f(2:m+1,2:m+1); f=f(:);
elseif N==10 % Modified step for rhs
    Gamma=formGamma(m,11);
    % Remove boundaries of f since we will apply a
    % five-point stencil to it
    fbot=f(1,1:end); fleft=f(1:end,1); ftop=f(end,1:end);
    fright=f(1:end,end);
    % After removing boundaries, make f one long vector
    f=f(2:m+1,2:m+1); f=f(:); len=length(f);
    % As in 5-point stencil, make a new vector of
    % boundary points.
    fbc=fleft(2:length(fleft)-1);
    fbc(len-m+1:len)=fright(2:length(firght)-1);
    for k=1:m
        fbc((k-1)*m+1)=fbp((k-1)*m+1)+fbot(k+1)
        fbc(m*k)=fbc(m*k)+ftop(k+1);
    end
    c=fastSineTransform(m,f)';
    % Reorder c by rows
    C=reshape(c,m,m); C=C'; c=C(:); v=Gamma*c;
    % Reorder v by columns
    V=reshape(v,m,m); V=V'; v=V(:);
    f=fastSineTransform(m,v)'; f=f+(1/12).*fbc;
end

 % Boundary Condition for poisson.m
 function [gb,gt,gl,gr]=dirichletBc(x,y)
 gb=zeros(1,length(x)); gt=1./(2+2*x+x.^2);
 gl=y./(1+y.^2); gr=y./(4+y.^2);
```

Chapter 6

Solution by Iteration

If the coefficient matrix is sparse and large, direct methods might not be the most efficient equation solver. Direct methods requires large amount of memmory, and they will destroy the sparsity pattern. Even though $A = LU$ is sparse, the individual factors L and U may not be as sparse as A.
After spatial discretization of a PDE, a linear, or nonlinear system of ODE equations arise

$$\frac{du}{dt} = F(u, t).$$

The classical schemes such as point-Jacobi, Gauss-Seidel, and SOR (*successive over-relaxation* are *fixed-point* iterations. To analyze fixed-point equations we need the *Banach fixe-point* theorem

6.1 The Banach Fixed-Point Theorem

Fixed point operator equations take the form

$$u = T(u), \qquad u \in K$$

Here, K is a subspace of the Banach space V, and $T : K \to V$ is an operator defined on K. A natural numerical approximation is defined by the iteration formula

$$u_{n+1} = T(u_n), \qquad n = 0, 1, ...$$

If the operator T is contractive, the itertion converge according to the Banach fixed-point theorem

Theorem 6.1 (Banach Fixed-Point Theorem) *Assume that K is a non-empty closed set in the Banach space V, and further, that $T : K \to K$ is a contractive mapping with contractivity constant α, $0 \leq \alpha < 1$. Then the following result hold.*

1. *Existence and uniquness: There exists a unique $u \in K$ such that*

$$u = T(u)$$

2. *Convergence and error estimates of the iteration: For any $u_0 \in K$, the sequence $\{u_n\} \subseteq K$ defined by $u_{n+1} = T(u_n)$, $n = 0, 1, ...$, converges to u:*

$$\|u_n - u\|_V \to 0 \qquad asn \to \inf$$

For the error, the following bounds are valid:

$$\begin{aligned} \|u_n - u\|_V &\leq \frac{\alpha^n}{1-\alpha}\|u_0 - u_n\|_V, \\ \|u_n - u\|_V &\leq \frac{\alpha}{1-\alpha}\|u_{n-1} - u_n\|_V, \\ \|u_n - u\|_V &\leq \alpha\|u_{n-1} - u\|_V \end{aligned}$$

Proof Since $T : K \to K$, the sequence u_n is well-defined. From the contractivity of T it follows that u_n is a Cauchy sequence

$$\|u_{n+1} - u_n\|_V = \|T(u_n) - T(u_{n-1})\|_V \leq \alpha\|u_n - u_{n-1}\|_V \leq ... \leq \alpha^n\|u_1 - u_0\|_V.$$

Then for any $m \geq n \geq 1$,

$$\begin{aligned} \|u_m - u_n\|_V &\leq \sum_{j=0}^{m-n-1} \|u_{n+j+1} - u_{n+j}\|_V \\ &\leq \sum_{j=0}^{m-n-1} \alpha^{n+j}\|u_1 - u_0\|_V \\ &\leq \frac{\alpha^n}{1-\alpha}\|u_1 - u_0\|_V \end{aligned}$$

Since $\alpha \in [0, 1)$, $\|u_m - u_n\|_V \to 0$ as $n \to \inf$. Thus $\{u_n\}$ is a Cauchy sequence; and since K is a closed set in the Banach space V, $\{u_n\}$ has a limit $u \in K$. As $n \to \inf$ in $u_{n+1} = T(u_n)$ we obtain the fixed point equation $u = T(u)$.
To prove uniqueness assume that $u_1, u_2 \in K$ are fixed points of T, then

$$\|u_1 - u_2\| = \|T(u_1) - T(u_2)\|_V \leq \alpha \|u_1 - u_2\|_V$$

which is a contradiction unless $u_1 = u_2$. We now prove the error estimates. The first one is allready proved, the third follows from the fixed point property of T

$$\|u_n - u\|_V = \|T(u_{n-1}) - T(u)\|_V \leq \alpha \|u_{n-1} - u\|_V.$$

The second follows from the third estimate together with

$$\|u_{n-1} - u\|_V \leq \|u_{n-1} - u_n\|_V + \|u_n - u\|_V$$

We recgonice the last inequality as a generalized triangular inequality

6.1.1 Classical Iteration methods for Linear Systems

When devising iterative methods it common practice to use a matrix splitting

$$A = N - M$$

By doing so, we can reformulate the system of equations $Ax = (N-M)x = b$ as a fixed point equation

$$x = N^{-1}Mx + N^{-1}b,$$

and the corresponding iterative approximation

$$x_{n+1} = N^{-1}Mx_n + N^{-1}b.$$

The matrix $G = N^{-1}M$ is called the iteration matrix, it determines the the convergence rate of the method. To see this we write down the error equation

$$x - x_n = G(x - x_{n-1}) = G^n(x - x_0)$$

Unfortunately the error $e_n = x - x_n$ is not available unless the exact solution is known. However, if the system of equations are well conditioned, the *residual* $r_n = b - Ax_n$ is a good measure of how well x_n approximate the true solution x. We have the following important relation between r and e:

$$Ae = r$$

We call this relation the *residual* equation, or the *deffect* equation.
For spatial discretization it is appropriate to write A as

$$A = D + L + U$$

Figure 6.1: Convergence rate for the Jacobi and Gauss-Seidel scheme applied to the 2D Poisson equation

where D is the diagonal of A, U and L are the strict upper and lower triangular parts. If we let $N = D$ and $M = -(L + U)$ we obtain the matrix representation of the Jacobi method

$$Dx_{n+1} = b - (L + U)x_n.$$

In component form the equation is expressed as

$$x_i^{n+1} = \frac{1}{a_{i,i}}\Big(b_i - \sum_{j \neq i} a_{ij}x^n\Big).$$

The Gauss-Seidel iteration is defined by

$$(D + L)x_{n+1} = b - Ux_n$$

This corresponds to $N = D + L$. The equivalent component form is

$$x_i^{n+1} = \frac{1}{a_{i,i}}\Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{n+1} - \sum_{j=i+1}^{m} a_{ij}x_j^{n-1}\Big)$$

We see that contrary to the Jacobi iteration, the Gauss-Seidel iteration include updated values for x as they become available. A close relative to the

Gauss-Seidel method is the SOR method. The method of SOR is usually exprssed in two equations. First a Gauss-Seidle iteration is made, then a weighted average is computed

$$\begin{aligned} z_{n+1} &= D^{-1}[b - Lx_{n+1} - Ux_n], \\ x_{n+1} &= \omega z_{n+1} + (1 - \omega)x_n \end{aligned}$$

Figure 6.2: The convergence rate for the SOR scheme on the 2D Poisson problem with overrelaxation parameter ω_{opt}

where $\omega \in (1, 2]$. This splitting is obtained by settingn

$$N = \frac{1}{\omega} D + L, \qquad M = \frac{1-\omega}{\omega} D - U.$$

The component formula of the SOR method is

$$x_i^{n+1} = x_i^n + \omega\Big(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{n+1} - \sum_{j=i+1}^{m} a_{ij} x_j^n\Big), \quad 1 \leq i \leq m$$

The convergence rate of Jacobi and Gauss-Seidel scheme is shown in figure 6.1.1. Note that the error drops rapidly the first few iterations, but altogether the convergence is very slow. Also note that Gauss-Seidel converges about twice as fast as Jacobi. The convergence of SOR depends crusially on the overrelaxation parameter ω. Convergence is slow relative to Gauss-Seidel if $\omega < 1$ while the scheme is unstable if $\omega > 2$ On a square, uniform $n \times n$ grid it can be shown that the best choice of ω is

$$\omega_{opt} = \frac{2}{1 + \sin\frac{\pi}{n}}$$

With this choice for ω SOR converges much faster than Jacobi and Gauss-Seidel on the Poisson problem.

For a more comprehensive introduction of classical iterative methods, their convergence properties, and parameter dependence, consult [1], [2], [3], or your favourit numerical analysis book.

```
function x=wjac(A,b,x0,niter,omega)
D=diag(diag(A)); LU=-(tril(A,-1)+triu(A,1)); x=x0;
for n=1:niter
    x=omega*(D\(LU*x+b))+(1-omega)*x;
end

function x=sor(A,b,x0,niter,omega)
DL=diag(diag(A))+tril(A,-1); U=-triu(A,1); x=x0;
for n=1:niter
    u=omega*(DL\(U*x+b))+(1-omega)*x;
end
```

6.1.2 Implementation

The matrix formulation of the classical iteration methods can be used to create a vectorized algorithm in Matlab. The following code implements SOR and the ω-damped Jacobi method. The GS and JAC method is obtained by setting $\omega = 1$ in SOR and ω-JAC respectively

Chapter 7

Multigrid

If we apply the weighted-Jacobi or Gauss-Seidel method to solve the Poisson equation, or perhaps another PDE, the error becomes *smooth* after a few iterations. The norm of the residual drops dramatically the first iterations, but then stalls. This smooth residual can be approximated on a coarser grid, without any essential loss of information. These observations provides a clue about how to accelerate iterative schemes for linear algebraic equations. Multigrid methods build on two basic principles, namely, *smoothing* and *coarse grid approximation*. To illustrate these principles, let us consider the Fourier expansion of the approximation error

$$\mathbf{e}_h(x,y) = \sum_{k,l}^{n-1} \alpha_{k,l} \sin k\pi x \sin l\pi y \qquad x,y \in \Omega_h.$$

The fact that this error becomes smooth after some iteration steps means that the *high frequency components* becomes small after a few iterations wheras the *low frequency components* hardly change. To be more specific, we look at the discrete Poisson equation on a fine grid Ω_h with mesh size $h = 1/N, N \in \mathbb{Z}$, and on a coarser grid Ω_H, where $H = 2h$. It is also assumed that N is an even number. For $(x,y) \in \Omega_h$, the functions

$$q_h^{k,l}(x,y) = \sin k\pi x \sin l\pi y \quad k,l = 1,...,N-1$$

are the discrete eigenfunctions of the discrete five-point operator $\frac{1}{h^2}(\Delta_{0,x}^2 + \Delta_{0,y}^2)$. For a given pair (k,l), we consider the four eigenfunctions

$$q^{k,l}, \quad q^{N-k,N-l}, \quad q^{N-k,l} \quad q^{k,N-l}$$

On the coarse grid Ω_{2h} these functions conicide in the following sence:

$$q^{k,l}(x,y) = -q^{N-k,l}(x,y) = -q^{k,N-l}(x,y) = q^{N-k,N-l}(x,y) \quad \text{for } x,y \in \Omega_{2h}$$

Figure 7.1: High and low frequency modes for N=8

This means that these four eigenvalues cannot be distinguished on Ω_{2h}, a phenomenon known as *aliasing*. This gives rise to the following definition of low and high frequencies

Definition 7.1 *The function* $q^{k,l},\ k,l = 1,2,...,N-1$ *is said to be a low frequency component if* $\max(k,l) < N/2$ *and a high frequency component if* $N/2 \leq \max(k,l) < N$

The low frequencies can be represented on the coarse grid Ω_{2h}, but the high frequencies are invisible since they coincide with low frequency modes at the grid nodes. This is illustrated in figure 7

7.1 From Two Grids to Multigrid

If N, the number of grid spacings, is a power of 2 meaning that $h = 2^{-p}$. Then we can from the grid sequence

$$\Omega_h, \Omega_{2h}, \Omega_{4h}, ..., \Omega_{h_0}$$

by doubling the mesh size successively. The coarsest grid Ω_{h_0} might contain only one interior grid point as shown in figure 7.1 The error can be decomposed into high and low frequencies On Ω_h there results

$$\sum_{k,l=1}^{N-1} \alpha_{k,l} q^{k,l} \sum_{k,l=1}^{N/2-1} \alpha_{k,l} q^{k,l} + \sum_{\substack{k,l \\ N/2 \leq \max(k,l)}}^{N-1} \alpha_{k,l} q^{k,l}$$

A similar distinction is made between high and low frequencies on $\Omega_{2h}, \Omega_{4h}, ...$ The terms 'high frequency' and 'low frequency' are related to the grids that

Figure 7.2: A sequence of coarser grids

we consider. For example, the high frequency components on Ω_{2h}, derived from the low frequency components on Ω_h, is not vissible on Ω_{4h}.
If a few iteration steps with a smoother is performed on the finest grid Ω_h, the high frequency error becomes small. The remaining low frequency components are visible on Ω_{2h} and can thus be approximated there. Now apply the smoother on $\Omega_2 h$ to filter out the high frequency components there. We continue this procedure until we arive at the coarsest grid. This travel down the hierarchy of successively coarser grid is reffered to as *coarsening*. There is also a opposite travel, ascending from a coarser to a finer grid. This is known as *refinement*. A typical multigrid algorithm travel up and down the grid hierarchy until sufficiently many frequency components of the error are determined.
It appears that many standard iterative methods possess the smoothing property. As we know, most classical relaxation schemes suffer in presence of smooth components of the error. These smooth components are however oscillatory on a coarser grid, also, relaxation on a coarser grid is less expensive because there are fewer unknowns to update.

7.1.1 Smoothing Properties of Relaxation Schemes

To illustrate the smoothing properties of some common smoothers the following computer program performs a few iterations on a model problem. The user can select weighted Jacobi, Gauss-Seidel or Red-Black Gauss-Seidel as the smoother.

Figure 7.3: Smoothing properties of weighted Jacobi

```
%function demo1_run(smoother,iter)
iter=6; smoother=1
 N=23; h=1/(N+1);NN=N*N; [A]=sp_laplace(N); xt=2*rand(NN,1)-1;
b=A*xt; x=zeros(NN,1); error=xt-x; ERROR=reshape(error,N,N);

figure(1); subplot(3,2,1); mesh(ERROR,'EdgeColor','black');
hold on; title('Initial error, Physical Space')
hold off; ERROR_COEF=sint2(ERROR);
subplot(3,2,2); mesh(ERROR_COEF,'EdgeColor','black');
hold on; title('Initial error, Fourier Space'); hold off;
```

```
 % Weighted Jacobi
if smoother==1 D=0.95*diag(diag(A));
  % Gauss Seidel
elseif smoother==2  L=tril(A);
   % Red Black Gauss Seidel
elseif smoother==3
    red=[1:2:NN]; black=[2:2:NN]; NR=length(red);
    NB=length(black);p=[red,black]; ip=zeros(NN,1);
    for i=1:NN, ip(p(i))=i; end
    R=A(red,red); B=A(black,black); C=A(red,black);
    b_p=b(p); br=p_p(1:NR); bb=b_p(NR+1:NN);
    xr=zeros(NR,1); xb=zeros(NB,1);
end

j=0;
while j<iter
    j=j+1;
    if smoother==1 x=x+D\(b-A*x);
    elseif smoother==2 x=x+L\(b-A*x);
    elseif smoother==3 xr=R\(br-C*xb);
        xb=B\(bb-C'*xr); x=x_p(ip);
    end

    error=xt-x; ERROR=reshape(error,N,N);
    subplot(3,2,3); mesh(ERROR,'EdgeColor','black'); hold on
    if smoother==1
     title(['Physical Space',num2str(j),' iters, weighted Jacobi.']);
    elseif smoother==2
    title(['Physical Space',num2str(j),' iters, Gauss Seidel.']);
    elseif smoother==3
    title(['Physical Space',num2str(j),' iters RB Gauss Seidel']);
    end

    hold off
    ERROR_COEF=sint2(ERROR); subplot(3,2,4);
    mesh(ERROR_COEF,'EdgeColor','black');
```

```
   hold on
    if smoother==1
        title(['Fouirer Space',num2str(j),' iters weighted Jacobi.']);
    elseif smoother==2
    title(['Fourier Space',num2str(j),' iters Gauss Seidel.']);
    elseif smoother==3
    title(['Fourier Space',num2str(j),' iters RB Gauss Seidel.']);
    end

    hold off
    j=j+1;
    if smoother==1, x=x+D\(b-A*x);
    elseif smoother==2, x=x+L\(b-A*x);
    elseif smoother==3, xr=R\(br-C*xb);
      xb=B\(bb-C'*xr);  x_p=[xr;xb]; x=x_p(ip);
    end

    error=xt-x; subplot(3,2,5); ERROR=reshape(error,N,N);
    mesh(ERROR,'EdgeColor','black'); hold on;
    if smoother==1;
     title(['Physical Space,',num2str(j),'iters, weighted Jacobi.']);
    elseif smoother==2
        title(['Physical Space,',num2str(j),' iters Gauss Seidel.']);
    elseif smoother==3
     title(['Physical Space,',num2str(j),' iters, RB Gauss Seidel']);
    end

    hold off
    ERROR_COEF=sint2(ERROR); subplot(3,2,6);
    mesh(ERROR_COEF,'EdgeColor','black');
    if smoother==1
      title(['Fourier Space,', num2str(j),' iters weighted jacobi']);
    elseif smooterh==2
    title(['Fourier Space', num2str(j),' iters Gauss Seidel']);
    elseif smoother==3
    title(['Fourier Space,',num2str(j),' iters RB Gauss Seidel']);
    end
    hold off; hold off
    pause(4)
end
hold off
```

```
 function [A]=sp_laplace(N)
T=2*eye(N)-diag(ones(N-1,1),1)-diag(ones(N-1,1),-1);
A=kron(eye(N),T)+kron(T,eye(N));

% SINT Sine transform
% Returns the sine transform of vector X. The length of X must
% be one less than the power of two.
% If X is a matrix, the SINT operation is applied to each
% column
function sx=sint(x)
[m,n]=size(x); y=zeros(2*(m+1),n);
y(2:m+1,1:n)=x; z=fft(y); sx=-imag(z(2:m+1,1:n));

function W=sint2(U)
% See fft2. Does a non-Hermitian transpose
W=sint(sint(U.').');
```

7.2 Two-Grid Cycle

The question we want to answer now is. How do we travel up and down the grid hierarchy? We start out with a linear system of equations

$$A\mathbf{u} = \mathbf{f}$$

Let us repeat some important definitions. For any approximation $\mathbf{v}$ of the solution $\mathbf{u}$, the error is given by $\mathbf{e} = \mathbf{u} - \mathbf{v}$, and the residual (or deffect) is given by $\mathbf{r} = \mathbf{f} - A\mathbf{v}$. Also recall that the residual equation is given by

$$A\mathbf{e} = \mathbf{r} \tag{7.1}$$

If we define an iteration operator by

$$\mathbf{v}_{n+1} = G\mathbf{v}_n - H^{-1}\mathbf{f}$$

where H depends upon the iterative method, and $G = I + H^{-1}A$ it follows that

$$\mathbf{e}_{n+1} = G\mathbf{e}_n$$

From this we deduce that $\mathbf{r}_{n+1} = AGA^{-1}\mathbf{r}_n$. Note that the role of the iteration operator is not to solve the system of equations, but to smooth the

error
Suppose we are solving the equation $A^h\mathbf{v}^h = \mathbf{f}^h$ on the fine grid. To smooth the error a few iterations with the iteration operator is performed, this is known as *relaxation*. Next the resulting residual $\mathbf{r}^h = A\mathbf{v}^h - \mathbf{f}^h$ needs to be translated into the coarser grid. This is done by means of a *restriction* operator

$$I_h^{2h} : \mathcal{G}(\Omega_h) \to \mathcal{G}(\Omega_{2h})$$

where $O\mathcal{G}(\Omega_h)$ denotes the linear space of grid functions on Ω_h. On the coarser grid Ω_{2h} we seek the solution of the equation $A\mathbf{e}^{2h} = \mathbf{r}^{2h}$. Once an approximation is made, we need to *prolongate* (or *interpolate*) the coarse-grid error to the fine grid. This mapping is done in terms of the prolongation operator

$$I_{2h}^h : \mathcal{G}(\Omega_2 h) \to \mathcal{G}(\Omega_h)$$

Altogether, a combination of smoothing and coarse grid correction result in a two-grid correction scheme (calculating v_{n+1}^h from v_n^h). It proceeds as follows

- Relax ν_1 times on $A^h\mathbf{v}^h = \mathbf{f}^h$ on Ω_h
- Compute the fine-grid residual $\mathbf{r}^h = \mathbf{f}^h - A^h\mathbf{v}^h$ and restrict it to the coarse grid by $\mathbf{r}^{2h} = I_h^{2h}\mathbf{r}^h$.
- Solve $A^{2h}\mathbf{e}^{2h} = \mathbf{r}^{2h}$ on Ω_{2h}.
- Interpolate the coarse-grid error to the fine grid by $\mathbf{e}^2 = I_{2h}^h\mathbf{e}^{2h}$ and correct the fine grid approximation by $\mathbf{v}^h = \mathbf{v}^h + \mathbf{e}^h$.
- Relax ν_2 times on $A^h\mathbf{v}^h = \mathbf{f}^h$.

This two-grid correction scheme depends crusially uppon the the relaxation (or smoothing) procdure, the numbers ν_1 and ν_2 of pre and post smoothing steps, the restriction operator I_h^{2h}, the coarse grid approximation A^{2h} and the prolongation operator I_{2h}^h These procedures are complementary. Relaxation on the fine grid eliminates high frequency components. This smooth error can be approximated on a coarser grid, but, assuming the residual equation can be solved accurately on Ω_{2h}, it is still important to transfer the error accurately back to the fine grid. To eliminate high frequency components caused by prolongation, postsmoothing is performed at the end of the cycle.

7.2.1 Multigrid Components

Having learnt the principles of multigrid, we now need to specify the multigrid components. The coarse grid operator A^{2h} is usually obtained by restricting the difference approximation to the coarse grid. The simplest restriction procedure is known as *injection*,

$$v^{2h}_{j,l} = v^h_{2j,2l}, \qquad j,l = 1,2,...,m$$

but a popular alternative is *full weighting*

$$\begin{aligned} v^{2h}_{j,l} &= \frac{1}{4}v^h_{2j,2l} + \frac{1}{8}(v^h_{2j-1,2l} + v^h_{2j,2l-1} + v^h_{2j,2l-1} + v^h_{2j+1,2l} + v_{2j,2l+1}) \\ &\quad + \frac{1}{16}(v^h_{2j-1,2l-1} + v^h_{2j+1,2l-1} + v^h_{2j-1,2l+1} + v^h_{2j+1,2l+1}), \quad j,l = 1,2,...,m. \end{aligned}$$

The latter can be rendered as a computational stencil in the form

The transfer of grid functions from a coarse to a fine grid is done by interpolation. For two-dimensional problems the component of $\mathbf{v}^h = I^h_{2h}\mathbf{v}^{2h}$ are given by

$$\begin{aligned} v^h_{2j-1,2l-1} &= v^{2h}_{j,l}, && j,l = 1,2,...,m; \\ v^h_{2j-1,2l} &= \frac{1}{2}(v^{2h}_{j,l} + v^{2h}_{j,l+1}), && j = 1,2,...,m-1, \quad l = 1,2,...,m; \\ v^h_{2j,2l-1} &= \frac{1}{2}(v^{2h}_{j,l} + v^{2h}_{j+1,l}), && j = 1,2,...,m, \qquad l = 1,2,...,m-1; \\ v_{2j,2l} &= \frac{1}{4}(v^{2h}_{j,l} + v^{2h}_{j,l+1} + v^{2h}_{j+1,l} + v^{2h}_{j+1,l+1}), && j,l = 1,2,...,m-1; \end{aligned}$$

Since we are dealing with residuals, the error along the boundary

7.3 The Multigrid Cycle

An ituitive approach to solve the coarse grid equation $A^{2h}\mathbf{e}^{2h} = \mathbf{r}^{2h}$ is to smooth the high frequency components on Ω_{2h} and then restrict the resulting

residual to a coarser grid Ω_{4h}. We continue this procedure until we reach the coarsest grid. On the coarsest grid the residual equation is solved using, for example, a direct method. Note that the dimension of A on the coarses grid is low, ideally just one. Next we travel up the grid hierarchy, correcting the residual on our way. After each prolongation, a smoother is used to remove added high frequency components.
The structure of a multigrid method depends on the the number of recursive calls μ on each grid level. In practice, μ is one or two. The V-cycle is obtained when μ is set to one. The W-cycle is obtained when μ is set to two.

Figure 7.4: Structure of a V-cycle and a W-cycle for a four-grid method. $\bullet$, smoothing; $\circ$, exact solution; \, fine-to-coarse; /, coarse-to- fine transfer;

The recursive definition of the V-Cyle is as follows

V-Cycle Scheme $\mathbf{v}^h \leftarrow V^h(\mathbf{v}^h, \mathbf{f}^h)$

1. Relax ν times on $A^h\mathbf{v}^h = \mathbf{f}^h$
2. If Ω^h = coarsest grid, then go to step 4.
   Else

$$\begin{aligned} \mathbf{f}^{2h} &\leftarrow I_h^{2h}(\mathbf{f}^h - A^h\mathbf{v}^h, \\ \mathbf{v}^{2h} &\leftarrow \mathbf{0}, \\ \mathbf{v}^{2h} &\leftarrow V^{2h}(\mathbf{v}^{2h}, \mathbf{f}^{2h}). \end{aligned}$$

3. Correct $\mathbf{v}^h \leftarrow \mathbf{v}^h + I_{2h}^h\mathbf{f}^{2h}$.
4. Relax ν_2 times on $A^h\mathbf{v}^h$

Note that here the residual $\mathbf{r}^{2h}$ is replaced with the vector $\mathbf{f}^{2h}$, and $\mathbf{e}^{2h}$ is replaced with $\mathbf{v}^{2h}$. The V-cycle scheme belongs to a family of multigrid schemes known as the μ-cycle method.

Figure 7.5: Schedule of grids for a five level FMG scheme

μ-Cycle Scheme $\mathbf{v}^h \leftarrow M\mu^h(\mathbf{v}^h, \mathbf{f}^h)$.

1. Relax μ_1 times on $A^h\mathbf{v}^h$
2. If Ω^h = coarsest grid, then go to step 4. Else

$$\begin{aligned} \mathbf{f}^{2h} &\leftarrow I_h^{2h}(\mathbf{f}^h - A^h\mathbf{v}^h) \\ \mathbf{v}^{2h} &\leftarrow \mathbf{0}, \\ \mathbf{v}^{2h} &\leftarrow M\mu^{2h}(\mathbf{v}^{2h}, \mathbf{f}^{2h}) \ \mu \text{ times} \end{aligned}$$

3. Correct $\mathbf{v}^h \leftarrow \mathbf{v}^h + I_{2h}^h\mathbf{v}^{2h}$.
4. Relax ν_2 times on $A^h\mathbf{v}^h = \mathbf{f}^h$

An obvious Achilles heel of all iterative methods is the choice of the starting vector $\mathbf{x}^{[0]}$. An initial approximation for iterative solvers, like multigrid, can be obtained with *nested iteration*. The idea is to approximate the initial conditons on a coarse grid , perform a few relaxation iterations, and then interpolate to obtain an improved initial guess on a finer grid. In general this is not sufficient since interpolation leads to nonnegligible high and low frequency components, thus we have to revisit the coarser grid to remove the low frequency components.
The combination of nested iteration and the V-cycle (correction-scheme) result in the *full multigrid* (FMG) method.

To get an idea of how multigrid code work, we make use of available code at MGNET.org [4]. MGLab is a public domain set of Matlab functions written by James Bordner and Faisal Saied. MGLab can solve two dimensional elliptic partial differential equations using finite differences and

includes several built-in problems (Poisson, Helmholtz, discontinuous coefficient problems, and nonselfadjoint problems). The GUI might not work on newer versions of Matlab, but it might be an exercise to make it work. Note that the matrices for the model problems are assembled with for loops. Matlab runs faster if you vectorize the for loops (if possible), and use build in sparse-matrix utilities. The function `multigrid_setup.m` assemble the modle-problem matrix for each level (the coarsest level here is 5) explicitly. Another approach is to use a loop (or vectorized loop) to assemble the matrices at each level, and store each of them in a cell structure. This approach is used in the multigrid/finite-element demo [5]. As an example we use MGLab to solve the Poisson problem on a 49×49 mesh with an initial guess of low and high frequencies. This demonstration uses a two-level V-cycle algorithm with $\nu_1 = 2$ presmoothings, and $\nu_2 = 2$ postsmoothings.

To use more than two levels set the global variable `coarse_level` to the desired level, but make sure that the dimension of the grid satisfy $2^k, k \in \mathbb{Z}$ so that the coarser grids are defined. If you want to solve another problem, for example the Helmholtz equation, change the `problem_flag` variable to `HELMHOLTZ`. Similarly, if you want to change some of the component functions, set the corresponding flag in the file `set_globals.m`. The code necessarry to perform a simple two-level V-Cycle on the Poisson problem is included here (several files are not included here, including the FMG code). If you are interested in multigrid algorithms you can download MGLab and do your own experiments, perhaps with a different model problem.

```
% demo2_run
clear;
include_globals; include_flags;
set_defaults;
coarse_level=2; % Two grid Algorithm
nx1=49; ny1=nx1;

smooth_flag=GAUSS_SEIDEL; nu1=0; nu2=4; % No pre-smoothing
%solver_flag=VMG;

prob_args=[10];
[A1,N1]=get_matrix(nx1,ny1); generate_matrix=0;

%if(solver_flag==VMG|precon_flag==MG_CYCLE)
    multigrid_setup;
%end
b1=get_rhs(nx1,ny1); h=1/(nx1+1);
[X,Y]=meshgrid([h:h:(1-h)]);
U_TRUE=sin(pi*X).*sin(pi*Y);
INIT_ERROR=sin(10*pi*X).*sin(10*pi*Y)+...
    sin(20*pi*X).*sin(20*pi*Y)+sin(30*pi*X).*sin(30*pi*Y)+...
    sin(40*pi*X).*sin(40*pi*Y);
init_error=reshape(INIT_ERROR,N1,1); u_true=reshape(U_TRUE,N1,1);
b1=A1*u_true; generate_rhs=0;

level=1;

u_rand=2*rand(nx1*nx1,1)-1;
u_in=u_true-init_error+0.5*u_rand; u_out=u_in;

RELRES=[];
jj=4;
for iter=1:jj
    u_out=demo2_Vcycle(level,b1,u_out,u_true,nx1,iter);
    relres=norm(b1-A1*u_out)/norm(b1); RELRES=[RELRES;relres];
    fprintf('Relative residual=%g, at iteration %g\n',relres,iter);
end

logrho=(1/(jj-2))*log10(RELRES(jj)/RELRES(2));
rho=10^logrho;
fprintf('Average residual reduction factor=%g.\n',rho);
hold off
```

```
% Multigrid V-Cycle Algorithm
function u_out=demo2_Vcycle(level,b,u_in,u_true,nx,iter)

% Use the zero vector for u_in as the default
if level==1 & iter ==1
    subplot(1,1,1), hold off, cla
    e=u_true-u_in; E=reshape(e,nx,nx);
    subplot(1,2,1), hold off, mesh(E); hold on;
    title(['Initial error']); subplot(1,2,2), hold off;
    surf(abs(sint2(E))); hold on;
    title(['Absolute value of initial error in Fourier space']);
    pause(4)
end

if nargin==2
    u_in=zeros(size(b));
end

if level==coarsest(level)
    u_out=coarse_grid_solve(level,b);
else
    u=smooth(level,b,u_in,'pre'); r=residual(level,b,u);
    b_c=restrict(level,r); u_c=demo2_Vcycle(level+1,b_c,...
        zeros(size(b_c)),u_true,nx,iter);
    correct=interpolate(level,u_c); u=u+correct;
    if level==1
        e=u_true-u; E=reshape(e,nx,nx); figure(iter+level);
        subplot(2,2,1), hold off, surf(E); hold on,
        title(['Error after coarse grid correction, iter= ',num2str(iter)]);
        subplot(2,2,2), hold off, surf(abs(sint2(E))); hold on
        title('Absolute value of error in Fourier space');
        pause(3)
    end
    u_out=smooth(level,b,u,'post');
    if level==1
     e=u_true-u_out; E=reshape(e,nx,nx);
     subplot(2,2,3), hold off, surf(E); hold on
     title(['Error after post-smoothing, iter =', num2str(iter)])
     subplot(2,2,4), hold off, surf(abs(sint2(E))); hold on
     title('Absolute value of error in Fourier space');
     pause(3)
    end
end
```

```
% INCLUDE FLAGS Flag variables defining current settings in MGLab
%
% Available problems
global problem_flag
POISSON=101; HELMHOLTZ=102; CONVECT_DIFFUSE=103;
POISSON_BOLTZMAN=104; CUT_SQUARE=105;
% Available interpolation operators
global interp_flag
CUBICINT=201; LINEAR=202; EXPLICIT_BILINEAR=203;
OPERATOR_BASED=204;
% Availabel restriction operators
global restrict_flag

INJECTION=301; HALF_WEIGHTING=302; FULL_WEIGHTING=303;
BILINEAR_ADJOINT=304;
% Available Smoothers
global smooth_flag
WEIGHTED_JACOBI=401; GAUSS_SEIDEL=402; RB_GAUSS_SEIDEL=403;
% Available preconditioners
global precon_flag
NONE=500; JACOVI=501; MG_CYCLE=502; ILU=505; SSOR=506;
BLOCK_JACOBI=507;
% Available solvers
global solver_flag
DIRECT=600; VMG=601; FMG=602; PCG=603; BICG_STAB=604;
GMRES=605; SMOOTHER=606; CGS=607; TFQMR=608; SOR=609;
% Available coarsenings
global coarsening_flag
STANDARD=701; GALERKIN=702; AVERAGING=703;
% Available coars grid solvers
global coarse_solver_flag
% Rhight hand side
global rhs_flag
% Available plotting axes
global x_axis_flag y_axis_flag
ITERATIONS=801; TIME=802; MFLOPS=803; RESIDUAL=804;
PRCON_RESIDUAL=805; ERROR=806;
% Available cylces
global cycle_flag
V_CYCLE=901; W_CYCLE=902; HALF_V_CYCLE=903;
```

```
% This M-file initializes global variables in
% include_globals.m, include_flags.m, and
% include_figs.m to their default values

include_globals; include_flags; %include_figs;

% Mesh parameter defaults
nx1=15; ny1=15;   % Set fine mesh size
coarse_level=3;   % max number of levels
% Problem parameter defaults
prob_args=[0 0];
% Smoother parameter defaults
nu1=1; nu2=1;     % number of pre and post-smoothings
wt=0.95;          % set smoother weight
% Selection flag defaults
problem_flag=POISSON;          % Select continuous problem
rhs_flag=1;                    % Select right-hand side
smooth_flag=WEIGHTED_JACOBI;   % Select smoother
restrict_flag=HALF_WEIGHTING;  % Select restriction operator
interp_flag=CUBICINT;             % Select interpolation operator
solver_flag=VMG;               % Select solver
precon_flag=MG_CYCLE;          % Select preconditioner
cycle_flag=V_CYCLE;            % Select MG cycle
coarsining_flag=0;             % Select coarsening
coarse_solver_flag=DIRECT;     % Select coarse grid solver
% Method parameter defaults
rtol=0;        % Set stopping tolerance on weighted residual
prtol=1e-4;    % Set stopping tol. on weighted pseudo-resid.
max_it=5;      % Set maximum number of iterations
max_time=0;    % Set maximum number of seconds
max_mflop=0;   % Set maximum number of mfplops
num_runs=1;    % Set number of experiments to run
restart=5;     % Set GMRES restart parameter
% Linear system defaults
generate_matrix=1;   % Must generate the matrix initially
generate_rhs=1;     % Must generate the right hand side initially
matrix_type=[];     % Matrix properties initially unknown
% Figures
%param_fig=0;
%main_position=[390,310,750,550]; param_position=[10 455 370 400];
```

```
% MULTIGRID_SETUP Generate the linear systems for the
% coarse multigrid levels
% Accesses global variables in "include_flags"
% Accesses global variables in "include_globals"
function multigrid_setup
include_flags; include_globals;
[X1,Y1]=meshgrid([0:nx1+1]/(nx1+1),[0:ny1+1]/(ny1+1));
ARRAY1=zeros(nx1+2,ny1+2);
if coarse_level>=2
    nx2=(nx1+1)/2-1; ny2=(ny1+1)/2-1;
    [A2,N2]=get_matrix(nx2,ny2);
    [X2,Y2]=meshgrid([0:nx2+1]/(nx2+1),[0:ny2+1]/(ny2+1));
    if interp_flag==EXPLICIT_BILINEAR
        ARRAY1=sp_prolong(nx1,ny1,nx2,ny2);
    else
        ARRAY2=zeros(nx2+2,ny2+2);
    end
end
if coarse_level>=3
    nx3=(nx2+1)/2-1; ny3=(ny2+1)/2-1; [A3,N3]=get_matrix(nx3,nx3);
    [X3,Y3]=meshgrid([0:nx3+1]/(nx3+1),[0:ny3+1]/(ny3+1));
    if interp_flag==EXPLICIT_BILINEAR
        ARRAY2=sp_prolong(nx2,nx2,nx3,nx3);
    else
        ARRAY3=zeros(nx3+2,ny3+2);
    end
end
if coarse_level >=4
    nx4=(nx3+1)/2-1; ny4=(ny3+1)/2-1; [A4,N4]=get_matrix(nx4,ny4);
    [X4,Y4]=meshgrid([0:nx4+1]/(nx4+1),[0:ny4+1]/(ny4+1));
    if interp_flag==EXPLICIT_BILINEAR
        ARRAY3=sp_prolong(nx3,ny3,nx4,ny4);
    else
        ARRAY4=zeros(nx4+2,ny4+2);
    end
end
if coarse_level >=5
    nx5=(nx4+1)/2-1; ny5=(ny4+1)/2-1; [A5,N5]=get_matrix(nx5,nx5);
    [X5,Y5]=meshgrid([0:nx5+1]/(nx5+1),[0:ny5+1]/(ny5+1));
    if interp_flag==EXPLICIT_BILINEAR
        ARRAY4=sp_prolong(nx4,nx4,nx5,nx5);
    else
        ARRAY5=zeros(nx5+2,ny5+2);
    end
end
```

```
% Generate a discrete linear operator
% [A,N]=get_matrix(NX,NY) generates the matrix A of
% order N for the problem defined by the global flag
% problem_flag and global parameters prob_args
function [A,N]=get_matrix(nx,ny)
include_flags; include_globals
tic
fprintf('Generating %g by %g matrix...',nx*ny,nx*ny)
if(problem_flag==POISSON)
    [A,N]=sp_laplace(nx,ny);
elseif(prolem_flag==CUT_SQUARE)
    val=prob_args(1); [A,N]=sp_cutsq2d(nx,ny,val);
elseif(problem_flag==HELMHOLTZ)
    lambda=0; sigma=prob_args(1); [A,N]=sp_convdiff(nx,ny,lambda,sigma);
elseif(problem_flag==CONVECT_DIFFUSE)
    lambda=prob_args(1); sigma=prob_args(2);
    [A,N]=sp_convdiff(nx,ny,lambda,sigma);
end
toc1=toc; fprintf('%g seconds.\n',toc1)

% Solve the coarse grid system
%
% coarse_grid_solve(level,b) solves the linear system at
% the grid level level with the right hand side b. How the
% system is solved depends on the global variable
% "coarse_solver_flag"
%
% DIRECT    -Sparse Gaussian elimination
% SMOOTHER  -A constant number of applications of the smoother
% PCG       -The PCG method
% BICG_STAB
% GMRES
function u_out=coarse_grid_solve(level,b)

include_flags; include_globals;

if(coarse_solver_flag==DIRECT)
    eval(['u_out=A', num2str(level), '\b;']);
elseif(coarse_solver_flag==SMOOTHER)
    u_out=smooth(level,b,b,'coarse');
%elseif(coarse_solver_flag==PCG)
end
```

```
% Coppy global variables associated with a given grid level
% to local variables

% Get FINE and COARSE array grids
cmd_str=['FINE=ARRAY',num2str(level),';']; eval(cmd_str)
cmd_str=['COARSE=ARRAY',num2str(level+1),';']; eval(cmd_str);
% Get X_f, Y_f, X_c, Y_c mesh point locations
cmd_str=['X_f=X',num2str(level),';']; eval(cmd_str)
cmd_str=['Y_f=Y',num2str(level),';']; eval(cmd_str)
cmd_str=['X_c=X',num2str(level+1),';']; eval(cmd_str)
cmd_str=['Y_c=Y',num2str(level+1),';']; eval(cmd_str)
% Get nx_f, ny_f, nx_c, ny_c
cmd_str=['nx_f=nx',num2str(level),';']; eval(cmd_str)
cmd_str=['ny_f=ny',num2str(level),';']; eval(cmd_str)
cmd_str=['nx_c=nx',num2str(level+1),';']; eval(cmd_str)
cmd_str=['ny_c=ny',num2str(level+1),';']; eval(cmd_str)
% Get N_f, N_c
cmd_str=['N_f=N',num2str(level),';']; eval(cmd_str)
cmd_str=['N_c=N',num2str(level+1),';'];eval(cmd_str)

% INTERPOLATE Transfer correction from the coarse grid to the
% current grid.
% Accesses global variables in "include_flags" and "include_globals"

function correct=interpolate(level,u_c_out)

include_flags; include_globals; extract_globals;
IX_c=2:(nx_c+1); IY_c=2:(ny_c+1); IX_f=2:(nx_f+1); IY_f=2:(ny_f+1);

if interp_flag==LINEAR
    COARSE(IX_c,IY_c)=reshape(u_c_out,nx_c,ny_c);
    FINE=interp2(X_c,Y_c,COARSE,X_f,Y_f);
    correct=reshape(FINE(IX_f,IY_f),N_f,1);
elseif interp_flag==CUBICINT
    COARSE(IX_c,IY_c)=reshape(u_c_out,nx_c,ny_c);
    FINE=interp2(X_c,Y_c,COARSE,X_f,Y_f,'cubic');
    correct=reshape(FINE(IX_f,IY_f),N_f,1);
%elseif interp_flag==EXPLICIT_BILINEAR
%    eval(['PROLONG=ARRAY',num2str(level),';']);
%    correct=PROLONG*u_c_out;
```

```
% This is a modified file (Assume that nx=ny)
function [A,N]=sp_laplace(nx,ny)
N=nx;
T=2*eye(N)-diag(ones(N-1,1),1)-diag(ones(N-1,1),-1);
A=kron(eye(N),T)+kron(T,eye(N));
N=nx*ny;

% SINT Sine transform
% Returns the sine transform of vector X. The length of X must
% be one less than the power of two.
% If X is a matrix, the SINT operation is applied to each
% column
function sx=sint(x)
[m,n]=size(x); y=zeros(2*(m+1),n);
y(2:m+1,1:n)=x; z=fft(y); sx=-imag(z(2:m+1,1:n));

% SINT Sine transform
% Returns the sine transform of vector X. The length of X must
% be one less than the power of two.
% If X is a matrix, the SINT operation is applied to each
% column
function sx=sint(x)
[m,n]=size(x); y=zeros(2*(m+1),n);
y(2:m+1,1:n)=x; z=fft(y); sx=-imag(z(2:m+1,1:n));

% RESIDUAL Compute the residual at the given level
% R=RESIDUAL(LEVEL,B,U) returns the residual R of the
% system Au=b at the given grid level
function r=residual(level,b,u)
include_globals
eval(['r=b-A',num2str(level),'*u;']);

% Get the rhs of a linear system
% B=get_rhs(nx,ny) generates a vector B of order nx*ny
% defined by the global flag rhs_flag
function b=get_rhs(nx,ny)
include_flags; N=nx*ny; [XB,YB]=meshgrid([1:nx]/(nx+1),[1:ny]/(ny+1));
B=sin(pi*XB).*sin(pi*YB).*sin(sqrt(2)*pi*XB).*sin(sqrt(3)*pi*YB);
b=reshape(B,N,1);
```

```
% INCLUDE GLOBALS Global matrices, vector and scalars
% Global variables associated with the MG meshes
global coarse_level
global nx1 ny1 N1 A1 ARRAY1 X1 Y1 b1
global nx2 ny2 N2 A2 ARRAY2 X2 Y2
global nx3 ny3 N3 A3 ARRAY3 X3 Y3
global nx4 ny4 N4 A4 ARRAY4 X4 Y4
global nx5 ny5 N5 A5 ARRAY5 X5 Y5
% Global parameters for the problems
global prob_args
% Global parameters for the solvers
global rtol prtol max_it max_time max_mflop num_runs...
    restart SOR_omega
% Global parameters for the smoother
global nu1 nu2 wt
% Global parameters for the linear system
global generate_matrix generate_rhs matrix_type dimensions
```

Chapter 8

Krylov Subspace Methods

A stationary iterative methods replace $Ax = b$ with $x = Gx + c$, where G is a constant matrix. $If x^{(0)} = 0$, then

$$\begin{aligned} x^{(1)} &= c \\ x^{(2)} &\in \text{span}c, Gc \\ x^{(3)} &\in \text{span}c, Gc, G^2c \\ x^{(k)} &\in \text{span}c, Gc, G^2c, ..., G^{k-1}c \\ &\equiv \mathcal{K}_k(G, c) \end{aligned}$$

and we call $\mathcal{K}_k(G, c)$ a *Krylov subpspace* [1] Note that $\mathcal{K}_k(G, c) = \mathcal{K}_k(\bar{G}, c)$ if $\bar{G} = I - G$. Note also that we expand the subspace $\mathcal{K}$ at each iteration. This means that after at most n iterations, Krylov subspace terminate with the true solution. This might sound good, but for large n (thousand, millions,etc.) this is not accetable. We can't afford more than a few hundred iterations.

The only way to make the iteration practical is to make a very clever choice for $\mathcal{K}$. If we use $c, Gc, G^2c, ...$, then after just a few iterations, our algorithm will lose accuracy. Instead of updating x with a staionary scheme, we can use the *variational approach*: Choose $x^{(x)} \in \mathcal{K}_k$ to minimize

$$\|x - x^*\|_Z$$

where $\|y\|_Z^2 = y^T Z y$ and Z is a symmetric positive definite matrix. Or we can use the *Galerkin approach*: Choose $x^{(k)} \in \mathcal{K}$ to make the residual

[1] Aleksei Nikolaevich Krylov (1863-1945) showed in 1931 how to use sequences of the form $b, Ab, A^2b, ...$ to construct the characteristic polynomial of a matrix. Krylov was a Russian applied mathematican whose scientific interests arose from his early training in naval science that involved the theories of buoyancy, stability, rolling and pitching, vibrations, and compass theories.

$r^{(k)} = b - Ax^{(k)}$ orthogonal to every vector in $\mathcal{K}_k$ for some choice of inner product.

8.1 Basis for the Krylov Space

An *orthonormal basis* for $\mathcal{K}$ makes the iteration practical. We say that a vector v is *B-orthogonal* to a vector u if

$$u^T Bv = 0 \tag{8.1}$$

where B is a symmetric positive definite matrix. Similarly we define $\|u\|_B = u^T Bu$.. The common way to construct a orthonormal basis is by the proceess of *Gram-Schmidt orthogonalization*. It proceed as follows. Let the first basic vector be $p^{(0)} = c/\|c\|_B$. Now suppose that we have $j+1$ basis vectors $p^{(0)}, ..., p^{(j)}$ for $\mathcal{K}_{j+1}(\hat{G}, c)$, and that we have some vector $r \notin \mathcal{K}_{j+1}(\hat{G}, c)$. The next basis vector is defined as

$$p^{j+1} = (r - h_{0,j}p^{(0)} - ... - h_{j,j}p^{(j)})/h_{j+1,j} \tag{8.2}$$

where $h_{i,j} = p^{(i)T} Br, i = 0, ..., j$, with $h_{j+1,j}$ chosen so that $p^{(j+1)T} Bp^{(j+1)} = 1$. Note that $r \in \mathcal{K}_{j+1}(\hat{G}, c)$ and that $p^{(j)}$ and $p^{(k)}$ are B-orthogonal, or conjugate if $j \neq k$. Ofent we let $r = \hat{G}p^{(j)} \in \mathcal{K}_{j+1}(\hat{G}, c)$. In matrix form we can express this relation as

$$\hat{G}p^{(j)} = [p^{(0)}p^{(1)}...p^{(j+1)}] \begin{bmatrix} h_{0,j} \\ h_{1,j} \\ \vdots \\ h_{j+1,j} \end{bmatrix}, \tag{8.3}$$

so after k steps we have

$$\hat{G}P_k = P_{k+1}H_k \tag{8.4}$$

where H_k is a $(k+1) \times k$ matrix with entries h_{ij} (zero if $i > j+1$) and P_k is $n \times k$ and contains the first k basis vectors as its columns. If n iterations are performed we obtain the following factorization of $\hat{G}$

$$\hat{G} = P_n H_n P_n^{-1}$$

Therefore, the matrix H_n is closely related to $\hat{G}$ -it has the same eigenvalues. In fact, the leading $k \times k$ piece of H_n (available after k steps) is in some sense a good approximation to $\hat{G}$. This is the key principle behind algorithms Krylov algorithms for

- solving linear systems of equations involving $\hat{G}$.
- finding approximations to eigenvalues and

eigenvectors of $\hat{G}$

We have just constructed the *Arnoldi algorithm*, an algoritm constructing an orthogonal basis for a Krylov space.

$[P, H] = \textbf{Arnoldi}(k, \hat{G}, B, p^{(0=)})$
Given a positive integer k, a symmetric definite matrix B, a matrix $\hat{G}$, and a initial vector $p^{(0)}$ with $\|p^{(0)}\| = 1$.

for $j = 0, 1, ..., k-1$,
  $p^{(j+1)} = \hat{G}p^j$
    for $i = 0, ..., j$,
    $h_{ij} = p^{(i)T} B p^{(j+1)}$
    $p^{(j+1)} - h_{ij}p^{(i)}$
  end
  $h_{j+1,j} = (p^{(j+1)T} B p^{(j+1)})^{1/2}$
  $p^{(j+1)} = p^{(j+1)}/h_{j+1,j}$.
end

In practice B is either the identity matrix (then the Gram-Schmidt procedure return orthogonal vectors). The Arnoldi algorithm perform one matrix-vector by $\hat{G}$ per iteration, after k iterations we have done $\mathcal{O}(k^2)$ inner products of length n each, and this work becomes significant as k increases. If $B\hat{G}$ is symmetric, then so is H_k, so all but two of the inner products at step j are zero. In this case we, can let the loop index $j = j-1 : j$ and the number of inner products drops to $\mathcal{O}(k)$. Then the Arnoldi algorithm is called *Lanczos tridiagonalization*. Now that we have a orthogonal basis for the Krylov space, let us find the next iteratate $x^{(k)}$. Following the variational approach we need to solve the minimization problem

$$\begin{aligned} &\text{minimize} && \|x - x^*\|_Z \\ &\text{subject to} && x^{(k)} \in \mathcal{K}_k \end{aligned} \tag{8.5}$$

Let $x^{(k)} = P_k y^{(k)}$ be the expansion of $x^{(k)}$ with respect to the basis $\mathcal{K}_k$. Then

$$\|x^{(k)} - x^*\|_Z^2 = (P_k y^{(k)} - x^*)^T Z (P_k y^{(k)} - x^*).$$

Differating with respect to the components of $y^{(k)}$, and setting the derivative to zero yields the necsessary condition for a minimum

$$P_k^T Z P_k y^k = P_k^T Z x^* \tag{8.6}$$

Since $y^{(k)}$ and x^* are both unknown, we usually can't solve this. But we can if we are clever about our choice of Z. Let $Z = \hat{G}^T B \hat{G}$ (This is symmetric, and positive definite if $\hat{G}$ is nonsingular). Then from equation (8.4) and the identity $\hat{G}x = (I - G)x = c$ it follows that

$$P_k^T Z x^* = P_k^T \hat{G}^T B \hat{G} x^* = P_k^T \hat{G}^T B \hat{G} x^* = P_k^T \hat{G} T B c = H_k^T P_k^T B c,$$

which is computable. From the B-orthogonality of the basis vectors we have that $P_{k+1} B P_{k+1} = I_{k+1}$. We can use this to simplify the left-hand side of equation (8.6)

$$P_k^T Z P_k = P_k^T \hat{G}^T B \hat{G} P_k = H_k^T P_{k+1}^T B P_{k+1} H_k = H_k^T H_k$$

So we need to solve

$$H_k^T H_k y^{(k)} = H_k^T P_{k+1}^T B c. \tag{8.7}$$

This algorithm is called $GMRES$ (generalized minimum residual), due to Saad and Schultz in 1986, and is probably the most often used Krylov method. If $\hat{G}$ is symmetric and positive definite a more efficent Krylov method is available, namely the CG (*concjugate gradient* method, due to Hestenes and Stiefel in 1952. It is the most often used Krylov method for symmetric problems. The derivation of the CG algorithm is similar to the derivation of the GMRES algorithm. Let $Z = B\hat{G}$. Then

$$P_k^T Z x^* = P_k^T B \hat{G} x^* = P_k^T B c$$

is computable. The left hand side of (8.6) also simplifies

$$P_k^T Z P_k = P_k^T B \hat{G} P_k = P_k^T B P_{k+1} H_k = \bar{H}_k$$

where $\bar{H}_k$ contains the first k rows of H_k. Thus we need to solve

$$\bar{H}_k y^{(k)} = P_k^T B c \tag{8.8}$$

In general, we need to save all of the old vectors in order to accomplish the projection of the residual. For symmetric positive definite matrices, as those arising from the discretization of self-adjoint elliptic PDEs, we only need a few old vectors.

The CG method can also be thought of as a minimization algorithm. If we set $B = I$, then

$$\text{minimize } \|x - x*\|_Z = \text{minimize } \|x - x*\|_A$$

To solve the linear system $Ax = b$ is equivalent to finding the minimum of

$$\|x - x*\|_A = \frac{1}{2} x^T A x - x^T A x^* + \frac{1}{2} x^{*T} A x^* \tag{8.9}$$

For a minimum to exsist, we need A to be positive definite. In addition, the CG algorithm is derived under the assumption that A is symmetric. We leave the constant in (8.9) behind and introduce the function

$$q(x) = \frac{1}{2}x^T Ax - x^T Ax^* = \frac{1}{2}x^T Ax - b^T x$$

which gives the same minimum as (8.9). We wish to find a sequence $x_k, k = 1, 2, ...$, such that

$$q(x_0) > q(x_1) > q(x_2) > ...$$

If we for each x_k choose a direction p_k and let x_{k+1} be at the minimum of $q(x_k + \alpha_k p_k)$ where α_k is the steplength. Then the sequence $q_k, k = 1, 2, ...$ is descending provided p is a *descent direction*. We have that

$$\begin{aligned} q(x_k + \alpha_k p_k) &= \frac{1}{2}(x_k + \alpha p_k)^T A(x_k + \alpha_k p_k) - b^T(x_k + \alpha_k p_k) \\ &= q(x_k) - \alpha_k(b - Ax_k)^T p_k + \frac{1}{2}\alpha_k^2 p_k^T A p_k \end{aligned}$$

The miniumum is found from the condition

$$\frac{dq}{dt} = -r_k^T p_k + \alpha_k p_k^T A p_k = 0$$

where $r_k = b - Ax_k$ is the residual. So the steplength is given by

$$\alpha_k = \frac{r_k^T p_k}{p_k^T A p j_k} \tag{8.10}$$

With this value of α_k vi get

$$q(x_k + \alpha_k p_k) = q(x_k) - \frac{1}{2}\frac{(r_k^T p_k)^2}{p_k^T A p_k} \leq q(x_k) \tag{8.11}$$

since A is spd. But how do we find p_k. We can for example choose the steepest descent direction $p_k = -\text{grad}q(x_k) = r_k$. This leads to the steepest descent algorithm

$[r, x] =$ **steepestDescent**(x_0, b, A)

Given$x_0, r_0 = b - Ax_0$
for$k = 0, 1, 2, ...$
$\alpha_k = r_k^T r_k / r_k^T A r_k$
$x_{k+1} = x_k + \alpha_k r_k$
$r_{k+1} = b - Ax_{k+1} = r_k - \alpha_k A r_k$
end

The steepest descent method converges if A is symmetric and positive definite, but it converges slowly. We try another the conjugate gradient direction instead (why the steepest descent direction converges slowly, and why the conjugate direction is a good choice is explained in [31].) The conjugate direction for a symmetric positive definite problem can be writen as

$$p_k = r_k + \beta_{k-1} p_{k-1}$$

Note that contrary to the Gram-Schmidt procedure performed by the arnoldi algorithm, p_k is found from the r_k and the previous value of p_k. Thus, we dont need to store the all the basis vectors to find a new one.
We now have

$$\begin{aligned} x_{k+1} &= x_k + \alpha_k p_k, \\ r_{k+1} &= r_k - \alpha_k A p_k, \\ p_{k+1} &= r_{k+1} + \beta_k p_k \end{aligned}$$

with α_k given by (8.10). Note that

$$p_k^T r_{k+1} = p_k^T r_k - \alpha_k p_k^T A p_k = 0$$

that is $r_{k+1} \perp p_k$. Similarly

$$p_{k+1}^T r_{k+1} = r_{k+1}^T r_{k+1} + \beta_k p_k^T r_{k+1} = \|r_{k+1}\|_2^2$$

If we choose $p_0 = r_0$, then yields

$$p_k^T r_k = \|r_k\|_2^2 \qquad \text{for } k \geq 0$$

This means that

$$\alpha_k = \frac{\|r_k\|_2^2}{p_k^T A p_k}, \quad q(x_{k+1}) = q(x_k) - \frac{1}{2} \frac{\|r_k\|_2^4}{p_k^T A p_k}$$

We now choose β_k so that $p_k^T A p_k$ gets as small as possible.

$$\begin{aligned} p_k^T A p_k &= (r_k + \beta_{k-1} p_{k-1})^T A (r_k + \beta_{k-1} p_{k-1}) \\ &= r_k^T A r_k + 2\beta_{k-1} r_k^T A p_{k+1} + \beta_{k-1} p_{k-1}^T A p_{k-1} \end{aligned}$$

Thus

$$\beta_{k-1} = -\frac{r_k^T A p_{k-1}}{p_{k-1} A p_{k-1}}, \qquad k \geq 1$$

Now we are done, but we can still simplyfy the expression for β_k. Note that

$$p_{k+1}^T A p_k = r_{k+1}^T A p_k + \beta_k p_{k-1}^T A p_k = 0,$$

$[x, r] = \text{cg}(x, b, A, \epsilon, kmax)$
$r = b - Ax, \rho_0 = \|r\|_2^2, k = 1.$
do while $\sqrt{\rho_{k-1}} \geq \epsilon \|b\|_2$ and $k < kmax$
if $k = 1$ then $p = r$
else
$\beta = \rho_{k-1}/\rho_{k-2}$ and $p = r + \beta p$
$w = Ap$
$\alpha = \rho_{k-1}/p^T w$
$x = x + \alpha p$
$r = r - \alpha w$
$\rho_k = \|r\|_2^2$
$k = k + 1$

So $p_{k+1}^T A p_k = 0, k \geq 0$, in other words p_k and p_{k+1} are A-conjugate. Futher

$$p_{k+1}^T A p_{k+1} + \beta_k p_{k+1}^T A p_k$$

So $p_{k+1}^T A p_{k+1} = r_{k+1}^T A p_{k+1}$, and

$$\begin{aligned} r_{k+1}^T r_k &= r_k^T - \alpha_k p_k^T A r_k \\ &= r_k^T r_k - \alpha_k p_k^T A p_k = 0 \end{aligned}$$

In other words $r_k \perp r_{k+1}, k \geq 0$. At last we have that

$$\begin{aligned} r_{k+1}^T r_{k+1} &= r_{k+1}^T r_k - \alpha_k r_{k+1}^T A p_k \\ &= -\alpha_k r_{k+1}^T A p_k \end{aligned}$$

Using these equations yields the following formula for β_k:

$$\begin{aligned} \beta_k &= -\frac{r_{k+1}^T A p_k}{p_k^T A p_k} = \frac{1}{\alpha_k} \frac{r_{k+1} r_{k+1}}{p_k^T A p_k} \\ &= \frac{\|r_{k+1}\|_2^2}{\|r_k\|_2^2} \end{aligned}$$

The covergence of the CG method depends on the condition number $\kappa_2 = \lambda_1/\lambda_N$ where λ_1 and λ_N corresond to the largest and smallest eigenvalue, respectively. It can be shown i.e in [32] that

$$\frac{\|b\|_2}{\|r_0\|_2} \frac{\|b - Ax_k\|_2}{\|b - Ax_0\|} \leq \sqrt{\kappa_2(A)} \frac{\|x_k - x^*\|_A}{\|x^* - x_0\|_A}$$

$[x, r] = pcg(x, b, A, M, \epsilon, kmax)$
$r = b - Ax, \rho_0 = \|r\|_2^2, k = 1$
Do while$\sqrt{\rho_{k-1}} > \epsilon \|b\|_2andk < kmax$
$z = Mr$
$\tau_{k-1} = z^T r$
if$k = 1$then$\beta = 0$and$p = z$
else
$\beta = \tau_{k-1}/\tau_{k-2}, p = z + \beta p$
$w = Ap$
$\alpha = \tau_{k-1}/p^T w$
$x = x + \alpha p$
$r = r - \alpha w$
$\rho_k = r^T r$
$k = k + 1$

For a ill conditioned problem a preconditioned CG algorithm will perform much beetter.

Our next task is to understand what a preconditioned problem is, and why we might need it. To improve the performance one might try to replace $Ax = b$ by another problem with the same solution, but with a condition number that is closer to one. It is important that the modified problem preserve the spd (symmetric positive definite) property of A. This can be done with a two-sided preconditioner. We can express the preconditioned problem in terms of B, where B is spd, $A = B^2$, and by using a two-sided preconditioner, $S \approx B^{-1}$. Then the matrix SAS is spd and ith eigenvalues are clustered near one. Morover the preconditioned system

$$SASy = Sb$$

has the solution $y^* = S^{-1}x^*$, where $Ax* = b$. Hence x^* can be recovered from y^* by multiplication by S. It is however not necessary to compute S. If $\hat{y}^k, r_k, \hat{p}_k$ are the iterate, residual, and search direction applied to SAS and we let

$$x_k = S\hat{y}_k, r_k = S^{-1}\hat{r}_k, p_k = S\hat{p}_k$$

then we obtain the preconditioned CG method (8.1) with $M = S^2$.
For futer analysis of CG, GMRES, and other iterative methods we reffer to

[32]. This reffernce also contains matlab code on its internet page, in fact you can download the whole book if you want. Among the simulation examples is the $2D$ convection diffusion equation.

Bibliography

[1] Arieh Iserles, *A first Course in the Numerical Analysisof Differential Equations*, Cambridge University Press, 2002.

[2] Harvard Lomax, Thomas H.Pulliam David W.Zing, *Fundamentals of Computational Fluid Dynamics*, Springer, 2001.

[3] A.A. Samarskii, P.N. Vabishchevich, *Computational Heat transfer* voume 1, Wiley, 1995

[4] James Bordner, Faisal Saied. *MGLab* 1995

[5] Bernd Flemisch, *Multigrid for Scalar Linear Elliptic PDEs*, 2004

[6] *Creating Graphical User Interfaces* version 7 2002. Matlab

[7] *Partial Differential Equation Toolbox* version 1 2002. Matlab

[8] Peter Knabner, Lutz Angermann. *Numerical Methods for Elliptic and Parabolic Partial Differential Equations*, 2003, Springer

[9] Serge Lang , *Linear Algebra*, Third Edition, 1987, Springer

[10] James W. Demmel, *Applied Numerical Linear Algebra*, 1997, Siam

[11] LIoyd N. Trefthen, *Spectral Methods in Matlab*, 2000, Siam

[12] Stig Larsson, Vidar Thome'e, *Partial Differential Equations with Numerical Methods*, 2003, Springer

[13] Kendall Atkinson, Weimin Han, *Theoretical Numerical Analysis*, 2001, Springer

[14] C. T. Kelly, *Solving Nonlinear Equations with Newton's Method* 2003, Siam

[15] Pablo Pedregal, *Introduction to Optimaization* , 2003, Springer

[16] James F. Epperson *An Introduction to Numerical Methods and Analysis*, 2001, Wiley

[17] W.E.Schiesser, C.A Silebi, *Computational Transport Phenomena*, 1997, Cambridge

[18] Claes Johnson, *Numerical solution of partial differential equations by the finite element method*, 1988, Cambridge

[19] David Kincaid, Ward Cheney, *Numerical Analysis*, 2001, Brooks/Cole

[20] Ulrich Trottenberg, Cornelis Oosterlee, Anton Achuller, *Multigrid*, 2001, Academic Press

[21] Carl D. Meyer, *Matrix Analysis and Applied Linear Algebra*

[22] Robert McOwen, *Partial Differential Equations*, 1995, Prentic-Hall.

[23] Jorge Nocedal, Stephen J. Wright, *Numerical Optimization*, 1999, Springer

[24] H K Versteeg, W Malalasekera, *An introduction to Computational Fluid Dynamics*, 1995, Prentic-Hall.

[25] Anne Kvrn, *lecture notes, Numerical solution of PDE with the Difference method*

[26] V. Arnautu, Pekka Neittanmaki, *Optimal Control from Theory to Computer Programs* , 2003, Springer

[27] LIoyd N. Trefethen, *Numerical Linear Algebra* 1997, Siam

[28] William L.Briggs, Van Emden Henson, Steve F. McCormick, *A Multigrid Tutorial*, Second Edition, 2000, Siam.

[29] Mark S.Gockenbach, *Partial Differential Equations: Analytical and Numerical Methods*, 2002, Siam.

[30] R. Barret, M. Berry, T. F. Chan, J.Demmel, J. Donato, J. Dongara, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst, *Templates for the solution of Linear Systems: Building Blocks for Iterative Methods, 1993* Siam (found at `http://www.netlib.org/templates/index.html`)

[31] Jonathan Richard Shewchuck, *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*, 1994, Thecnical Report from Carnegie Mellon University

[32] C. T. Kelly *Iterative Methods for Linear and Nonlinear Equations*, 1995, Siam.