Chapter 1

Preliminaries

Contents

In this chapter, we present some preliminary materials which will be used throughout the book. The first section set the stage for the introduction of spectral methods. In Sections 1.2∼1.4, we present some basic properties of orthogonal polynomials, which play an essential role in spectral methods, and introduce the notion of generalized Jacobi polynomials. Since much of the success and popularity of spectral methods can be attributed to the invention of Fast Fourier Transform (FFT), an algorithmic description of the FFT is presented in Section 1.5. In the next two sections, we collect some popular time discretization schemes and iterative schemes which will be frequently used in the book. In the last section, we present a concise error analysis for several projection operators which serves as the basic ingredients for the error analysis of spectral methods.

1.1 Some basic ideas of spectral methods

Comparison with the finite element method
Computational efficiency
Fourier spectral method
Phase error

Finite Difference (FD) methods approximate derivatives of a function by *local* arguments (such as $u'(x) \approx (u(x+h)-u(x-h))/2h$, where h is a small grid spacing) - these methods are typically designed to be exact for polynomials of low orders. This approach is very reasonable: since the derivative is a local property of a function, it makes little sense (and is costly) to invoke many function values far away from the point of interest.

In contrast, spectral methods are *global*. The traditional way to introduce them starts by approximating the function as a sum of very smooth basis functions:

$$u(x) \approx \sum_{k=0}^{N} a_k \Phi_k(x),$$

where the $\Phi_k(x)$ are polynomials or trigonometric functions. In practice, there are many feasible choices of the basis functions, such as:

$\Phi_k(x) = e^{ikx}$ (the Fourier spectral method); $\Phi_k(x) = T_k(x)$ ($T_k(x)$ are the Chebyshev polynomials; the Chebyshev spectral method); $\Phi_k(x) = L_k(x)$ ($L_k(x)$ are the Legendre polynomials; the Legendre spectral method).

In this section, we will describe some basic ideas of spectral methods. For ease of exposition, we consider the Fourier spectral method (i.e. the basis functions are chosen as e^{ikx}). We begin with the periodic heat equation, starting at time 0 from $u_0(x)$:

$$u_t = u_{xx}, \tag{1.1.1}$$

with a periodic boundary condition $u(x,0) = u_0(x) = u_0(x+2\pi)$. Since the exact solution u is periodic, it can be written as an infinite Fourier series. The approximate solution u^N can be expressed as a *finite* series. It is

$$u^N(x,t) = \sum_{k=0}^{N-1} a_k(t) e^{ikx}, \qquad x \in [0, 2\pi),$$

where each $a_k(t)$ is to be determined.

Comparison with the finite element method

We may compare the spectral method (before actually describing it) to the finite element method. One difference is this: the trial functions τ_k in the finite element method are usually 1 at the mesh-point, $x_k = kh$ with $h = 2\pi/N$, and 0 at the other mesh-points, whereas e^{ikx} is nonzero everywhere. That is not such an important distinction. We could produce from the exponentials an interpolating function like τ_k, which is zero at all mesh-points except at $x = x_k$:

$$F_k(x) = \frac{1}{N}\sin\frac{N}{2}(x - x_k)\cot\frac{1}{2}(x - x_k), \qquad N \text{ even}, \tag{1.1.2}$$

$$F_k(x) = \frac{1}{N}\sin\frac{N}{2}(x - x_k)\csc\frac{1}{2}(x - x_k), \qquad N \text{ odd}. \tag{1.1.3}$$

Of course it is not a piecewise polynomial; that distinction is genuine. A consequence of this difference is the following:

> Each function F_k spreads over the whole solution interval, whereas τ_k is zero in all elements not containing x_k. The stiffness matrix is sparse for the finite element method; in the spectral method it is full.

The computational efficiency

Since the matrix associated with the spectral method is full, the spectral method seems more time-consuming than finite differences or finite elements. In fact, the spectral method had not been used widely for a long time. The main reason is the expensive cost in computational time. However, the discovery of the Fast Fourier Transform (FFT) by Cooley and Tukey[33] solves this problem. We will describe the Cooley-Tukey algorithm in Chapter 5. The main idea is the following. Let $w_N = e^{2\pi i/N}$ and

$$(\mathcal{F}_N)_{jk} = w_N^{jk} = \cos\frac{2\pi jk}{N} + i\sin\frac{2\pi jk}{N}, \qquad 0 \leqslant j, \quad k \leqslant N-1.$$

Then for any N-dimensional vector v_N, the usual N^2 operations in computing $\mathcal{F}_N v_N$ are reduced to $N\log_2 N$. The significant improvement can be seen from the following table:

N	N^2	$N\log_2 N$	N	N^2	$N\log_2 N$
16	256	64	256	65536	2048

```
 32     1024       160        512     262144       4608
 64     4096       384       1024    1048576      10240
128    16384       896       2048    4194304      22528
```

The Fourier spectral method

Unlike finite differences or finite elements, which replace the right-hand side u_{xx} by differences at nodes, the spectral method uses u_{xx}^N exactly. In the spectral method, there is no Δx. The derivatives with respect to space variables are computed explicitly and correctly.

The Fourier approximation u^N is a combination of oscillations e^{ikx} up to frequency $N-1$, and we simply differentiate them; hence

$$u_t^N = u_{xx}^N$$

becomes

$$\sum_{k=0}^{N-1} a_k'(t)e^{ikx} = \sum_{k=0}^{N-1} a_k(t)(ik)^2 e^{ikx}.$$

Since frequencies are uncoupled, we have $a_k'(t) = -k^2 a_k(t)$, which gives

$$a_k(t) = e^{-k^2 t} a_k(0),$$

where the values $a_k(0)$ are determined by using the initial function:

$$a_k(0) = \frac{1}{2\pi}\int_0^{2\pi} u_0(x)e^{-ikx}\mathrm{d}x.$$

It is an easy matter to show that

$$|u(x,t) - u^N(x,t)| = \left|\sum_{k=N}^{\infty} a_k(0)e^{ikx}e^{-k^2 t}\right|$$

$$\leqslant \max_k |a_k(0)| \sum_{k=N}^{\infty} e^{-k^2 t}$$

$$\leqslant \max_{0\leqslant x\leqslant 2\pi} |u_0(x)| \int_N^{\infty} e^{-tx^2}\mathrm{d}x.$$

Therefore, the error goes to zero very rapidly as N becomes reasonably large. The

convergence rate is determined by the integral term

$$J(t,N) := \int_N^\infty e^{-tx^2}\,\mathrm{d}x = \sqrt{\frac{\pi}{4t}}\,\mathrm{erfc}(\sqrt{t}N),$$

where `erfc`(x) is the complementary error function (both FORTRAN and MATLAB have this function). The following table lists the value of $J(t,N)$ at several values of t:

N	J(0.1, N)	J(0.5, N)	J(1, N)
1	1.8349e+00	3.9769e-01	1.3940e-01
2	1.0400e+00	5.7026e-02	4.1455e-03
3	5.0364e-01	3.3837e-03	1.9577e-05
4	2.0637e-01	7.9388e-05	1.3663e-08
5	7.1036e-02	7.1853e-07	1.3625e-12
6	2.0431e-02	2.4730e-09	1.9071e-17
7	4.8907e-03	3.2080e-12	3.7078e-23
8	9.7140e-04	1.5594e-15	9.9473e-30

In more general problems, the equation in time will not be solved exactly. It needs a difference method with time step Δt, as Chapter 5 will describe. For derivatives with respect to space variables, there are two ways:

(1) Stay with the harmonics e^{ikx} or $\sin kx$ or $\cos kx$, and use FFT to go between coefficients a_k and mesh values $u^N(x_j,t)$. Only the mesh values enter the difference equation in time.

(2) Use an expansion $U = \sum U_k(t)F_k(x)$, where $F_k(x)$ is given by (1.1.2) and (1.1.3), that works directly with values U_k at mesh points (where $F_k = 1$). There is a *differentiation matrix* D that gives mesh values of the derivatives, $D_{jk} = F_k'(x_j)$. Then the approximate heat equation becomes $U_t = D^2U$.

Phase error

The fact that x-derivatives are exact makes spectral methods free of phase error. Differentiation of the multipliers e^{jkx} give the right factor ik while finite differences lead to the approximate factor iK:

$$\frac{e^{ik(x+h)} - e^{ik(x-h)}}{2h} = iKe^{ikx}, \qquad K = \frac{\sin kh}{h}\,.$$

When kh is small and there are enough mesh points in a wavelength, K is close to k. When kh is large, K is significantly smaller than k. In the case of the heat

equation (1.1.1) it means a slower wave velocity. For details, we refer to Richtmyer and Morton[131] and LeVeque [101]. In contrast, the spectral method can follow even the nonlinear wave interactions that lead to turbulence. In the context of solving high Reynolds number flow, the low physical dissipation will not be overwhelmed by large numerical dissipation.

Exercise 1.1

Problem 1 Consider the linear heat equation (1.1.1) with homogeneous Dirichlet boundary conditions $u(-1,t) = 0$ and $u(1,t) = 0$. If the initial condition is $u(x,0) = \sin(\pi x)$, then the exact solution of this problem is given by $u(x,t) = e^{-\pi^2 t}\sin(\pi x)$. It has the infinite Chebyshev expansion

$$u(x,t) = \sum_{n=0}^{\infty} b_n(t) T_n(x),$$

where

$$b_n(t) = \frac{1}{c_n} J_n(\pi) e^{-\pi^2 t},$$

with $c_0 = 2$ and $c_n = 1$ if $n \geqslant 1$.

a. Calculate

$$J_n(\pi) = \int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} T_n(x)\sin(\pi x)\mathrm{d}x$$

by some numerical method (e.g. Simpson's rule)①;

b. Plot $J_n(\pi)$ against n for $n \leqslant 25$. This will show that the truncation series converges at an exponential rate (a well-designed collocation method will do the same).

1.2 Orthogonal polynomials

Existence
Zeros of orthogonal polynomials
Polynomial interpolations
Quadrature formulas
Discrete inner product and discrete transform

① Hint: (a) Notice that $J_n(\pi) = 0$ when n is even; (b) a coordinate transformation like $x = \cos\theta$ may be used.

Orthogonal polynomials play a fundamental role in the implementation and analysis of spectral methods. It is thus essential to understand some general properties of orthogonal polynomials. Two functions f and g are said to be *orthogonal* in the weighted Sobolev space $L^2_\omega(a,b)$ if

$$\langle f,\, g\rangle := (f,g)_\omega := \int_a^b \omega(x) f(x) g(x) \mathrm{d}x = 0,$$

where ω is a fixed positive *weight function* in (a,b). It can be easily verified that $\langle \cdot,\, \cdot \rangle$ defined above is an inner product in $L^2_\omega(a,b)$.

A sequence of *orthogonal polynomials* is a sequence $\{p_n\}_{n=0}^\infty$ of polynomials with $\deg(p_n) = n$ such that

$$\langle p_i,\, p_j\rangle = 0 \qquad \text{for} \quad i \neq j. \tag{1.2.1}$$

Since orthogonality is not altered by multiplying a nonzero constant, we may normalize the polynomial p_n so that the coefficient of x^n is one, i.e.,

$$p_n(x) = x^n + a_{n-1}^{(n)} x^{n-1} + \cdots + a_0^{(n)}.$$

Such a polynomial is said to be *monic*.

Existence

Our immediate goal is to establish the existence of orthogonal polynomials. Although we could, in principle, determine the coefficients $a_j^{(n)}$ of p_n in the natural basis $\{x^j\}$ by using the orthogonality conditions (1.2.1), it is more convenient, and numerically more stable, to express p_{n+1} in terms of lower-order orthogonal polynomials. To this end, we need the following general result:

Let $\{p_n\}_{n=0}^\infty$ be a sequence of polynomials such that p_n is exactly of degree n. If

$$q(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0, \tag{1.2.2}$$

then q can be written uniquely in the form

$$q(x) = b_n p_n + b_{n-1} p_{n-1} + \cdots + b_0 p_0. \tag{1.2.3}$$

In establishing this result, we may assume that the polynomials $\{p_n\}$ are monic. We shall prove this result by induction. For $n = 0$, we have

$$q(x) = a_0 = a_0 \cdot 1 = a_0 p_0(x).$$

Hence we must have $b_0 = a_0$. Now assume that q has the form (1.2.2). Since p_n is the only polynomial in the sequence $p_n, p_{n-1}, \cdots, p_0$ that contains x^n and since p_n is monic, it follows that we must have $b_n = a_n$. Hence, the polynomial $q - a_n p_n$ is of degree $n-1$. Thus, by the induction hypothesis, it can be expressed uniquely in the form

$$q - a_n p_n = b_{n-1} p_{n-1} + \cdots + b_0 p_0,$$

which establishes the result.

A consequence of this result is the following:

Lemma 1.2.1 *If the sequence of polynomials $\{p_n\}_{n=0}^{\infty}$ is monic and orthogonal, then the polynomial p_{n+1} is orthogonal to any polynomial q of degree n or less.*

We can establish this by the following observation:

$$\langle p_{n+1},\, q\rangle = b_n\langle p_{n+1},\, p_n\rangle + b_{n-1}\langle p_{n+1},\, p_{n-1}\rangle + \cdots + b_0\langle p_{n+1},\, p_0\rangle = 0,$$

where the last equality follows from the orthogonality of the polynomials $\{p_n\}$.

We now prove the existence of orthogonal polynomials①. Since p_0 is monic and of degree zero, we have

$$p_0(x) \equiv 1.$$

Since p_1 is monic and of degree one, it must have the form

$$p_1(x) = x - \alpha_1.$$

To determine α_1, we use orthogonality:

$$0 = \langle p_1,\, p_0\rangle = \int_a^b \omega(x) x \mathrm{d}x - \alpha_1 \int_a^b \omega(x)\mathrm{d}x.$$

Since the weight function is positive in (a, b), it follows that

$$\alpha_1 = \int_a^b \omega(x) x \mathrm{d}x \Big/ \int_a^b \omega(x)\mathrm{d}x.$$

In general we seek p_{n+1} in the form $p_{n+1} = xp_n - \alpha_{n+1}p_n - \beta_{n+1}p_{n-1} - \gamma_{n+1}p_{n-2} - \cdots$. As in the construction of p_1, we use orthogonality to determine the coefficients above. To determine α_{n+1}, write

$$0 = \langle p_{n+1},\, p_n\rangle = \langle xp_n,\, p_n\rangle - \alpha_{n+1}\langle p_n,\, p_n\rangle - \beta_{n+1}\langle p_{n-1},\, p_n\rangle - \cdots .$$

① The procedure described here is known as Gram-Schmidt orthogonalization.

By orthogonality, we have

$$\int_a^b x\omega p_n^2 \mathrm{d}x - \alpha_{n+1}\int_a^b \omega p_n^2 \mathrm{d}x = 0,$$

which yields

$$\alpha_{n+1} = \int_a^b x\omega p_n^2 \mathrm{d}x \Big/ \int_a^b \omega p_n^2 \mathrm{d}x.$$

For β_{n+1}, using the fact $\langle p_{n+1},\, p_{n-1}\rangle = 0$ gives

$$\beta_{n+1} = \int_a^b x\omega p_n p_{n-1} \mathrm{d}x \Big/ \int_a^b \omega p_{n-1}^2 \mathrm{d}x.$$

The formulas for the remaining coefficients are similar to the formula for β_{k+1}; e.g.

$$\gamma_{n+1} = \int_a^b x\omega p_n p_{n-2} \mathrm{d}x \Big/ \int_a^b \omega p_{n-2}^2 \mathrm{d}x.$$

However, there is a surprise here. The numerator $\langle xp_n,\, p_{n-2}\rangle$ can be written in the form $\langle p_n,\, xp_{n-2}\rangle$. Since xp_{n-2} is of degree $n-1$ it is orthogonal to p_n. Hence $\gamma_{n+1} = 0$, and likewise the coefficients of p_{n-3}, p_{n-4}, etc. are all zeros.

To summarize:

The orthogonal polynomials can be generated by the following recurrence:

$$\begin{cases} p_0 = 1, \\ p_1 = x - \alpha_1, \\ \quad \cdots\cdots \\ p_{n+1} = (x - \alpha_{n+1})p_n - \beta_{n+1}p_{n-1}, \quad n \geqslant 1, \end{cases} \tag{1.2.4}$$

where

$$\alpha_{n+1} = \int_a^b x\omega p_n^2 \mathrm{d}x \Big/ \int_a^b \omega p_n^2 \mathrm{d}x \text{ and } \beta_{n+1} = \int_a^b x\omega p_n p_{n-1} \mathrm{d}x \Big/ \int_a^b \omega p_{n-1}^2 \mathrm{d}x.$$

The first two equations in the recurrence merely start things off. The right-hand side of the third equation contains three terms and for that reason is called the *three-term recurrence relation* for the orthogonal polynomials.

Zeros of orthogonal polynomials

The zeros of the orthogonal polynomials play a particularly important role in the implementation of spectral methods.

Lemma 1.2.2 *The zeros of p_{n+1} are real, simple, and lie in the open interval (a, b).*

The proof of this lemma is left as an exercise. Moreover, one can derive from the three term recurrence relation (1.2.4) the following useful result.

Theorem 1.2.1 *The zeros $\{x_j\}_{j=0}^{n}$ of the orthogonal polynomial p_{n+1} are the eigenvalues of the symmetric tridiagonal matrix*

$$A_{n+1} = \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & & & \\ \sqrt{\beta_1} & \alpha_1 & \sqrt{\beta_2} & & \\ & \ddots & \ddots & \ddots & \\ & & \sqrt{\beta_{n-1}} & \alpha_{n-1} & \sqrt{\beta_n} \\ & & & \sqrt{\beta_n} & \alpha_n \end{bmatrix}, \tag{1.2.5}$$

where

$$\alpha_j = \frac{b_j}{a_j}, \ \text{for} \ j \geqslant 0; \quad \beta_j = \frac{c_j}{a_{j-1}a_j}, \ \text{for} \ j \geqslant 1, \tag{1.2.6}$$

with $\{a_k, b_k, c_k\}$ being the coefficients of the three term recurrence relation (cf. (1.2.4)*) written in the form:*

$$p_{k+1} = (a_k x - b_k)p_k - c_k p_{k-1}, \qquad k \geqslant 0. \tag{1.2.7}$$

Proof The proof is based on introducing

$$\tilde{p}_n(x) = \frac{1}{\sqrt{\gamma_n}} p_n(x),$$

where γ_n is defined by

$$\gamma_n = \frac{c_n a_{n-1}}{a_n}\gamma_{n-1}, \ \ n \geqslant 1, \quad \gamma_0 = 1. \tag{1.2.8}$$

We deduce from (1.2.7) that

$$x\tilde{p}_j = \frac{c_j}{a_j}\sqrt{\frac{\gamma_{j-1}}{\gamma_j}}\tilde{p}_{j-1} + \frac{b_j}{a_j}\tilde{p}_j + \frac{1}{a_j}\sqrt{\frac{\gamma_{j+1}}{\gamma_j}}\tilde{p}_{j+1}, \quad j \geqslant 0, \tag{1.2.9}$$

with $\tilde{p}_{-1} = 0$. Owing to (1.2.6) and (1.2.8), it can be rewritten as

$$x\tilde{p}_j(x) = \sqrt{\beta_j}\tilde{p}_{j-1}(x) + \alpha_j\tilde{p}_j(x) + \sqrt{\beta_{j+1}}\tilde{p}_{j+1}(x), \quad j \geqslant 0. \tag{1.2.10}$$

We now take $j = 0, 1, \cdots, n$ to form a system

$$x\widetilde{\mathbf{P}}(x) = A_{n+1}\widetilde{\mathbf{P}}(x) + \sqrt{\beta_{n+1}}\tilde{p}_{n+1}(x)\mathbf{E}_n, \tag{1.2.11}$$

where $\widetilde{\mathbf{P}}(x) = (\tilde{p}_0(x), \tilde{p}_1(x), \cdots, \tilde{p}_n(x))^{\mathrm{T}}$ and $\mathbf{E}_n = (0, 0, \cdots, 0, 1)^{\mathrm{T}}$. Since $\tilde{p}_{n+1}(x_j) = 0,\ 0 \leqslant j \leqslant n$, the equation (1.2.11) at $x = x_j$ becomes

$$x_j\widetilde{\mathbf{P}}(x_j) = A_{n+1}\widetilde{\mathbf{P}}(x_j), \quad 0 \leqslant j \leqslant n. \tag{1.2.12}$$

Hence, the zeros $\{x_j\}_{j=0}^n$ are the eigenvalues of the symmetric tridiagonal matrix A_{n+1}. □

Polynomial interpolations

Let us denote

$$P_N = \{\text{polynomials of degree not exceeding } N\}. \tag{1.2.13}$$

Given a set of points $a = x_0 < x_1 \cdots < x_N = b$ (we usually take $\{x_i\}$ to be zeros of certain orthogonal polynomials), we define the polynomial interpolation operator, $I_N : C(a, b) \to P_N$, associated with $\{x_i\}$, by

$$I_N u(x_j) = u(x_j), \qquad j = 0, 1, \cdots, N. \tag{1.2.14}$$

The following result describes the discrepancy between a function u and its polynomial interpolant $I_N u$. This is a standard result and its proof can be found in most numerical analysis textbook.

Lemma 1.2.3 *If $x_0, x_1, \cdots, x_N$ are distinct numbers in the interval $[a, b]$ and $u \in C^{N+1}[a, b]$, then, for each $x \in [a, b]$, there exists a number ζ in (a, b) such that*

$$u(x) - I_N u(x) = \frac{u^{(N+1)}(\zeta)}{(N+1)!}\prod_{k=0}^{N}(x - x_k), \tag{1.2.15}$$

where $I_N u$ is the interpolating polynomial satisfying (1.2.14).

It is well known that for an arbitrary set of $\{x_j\}$, in particular if $\{x_j\}$ are equally spaced in $[a, b]$, the error in the maximum norm, $\max_{x\in[a,b]} |u(x) - I_N(x)|$, may

not converge as $N \to +\infty$ even if $u \in C^{\infty}[a,b]$. A famous example is the Runge function

$$f(x) = \frac{1}{25x^2+1}, \quad x \in [-1,1], \tag{1.2.16}$$

see Figure 1.1.

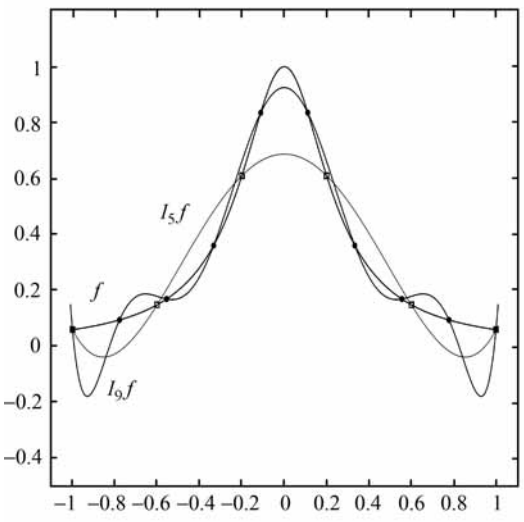

Figure 1.1 Runge function f and the equidistant interpolations $I_5 f$ and $I_9 f$ for (1.2.16)

The approximation gets worse as the number of interpolation points increases.

Hence, it is important to choose a suitable set of points for interpolation. Good candidates are the zeros of certain orthogonal polynomials which are Gauss-type quadrature points, as shown below.

Quadrature formulas

We wish to create quadrature formulas of the type

$$\int_a^b f(x)\omega(x)\mathrm{d}x \approx \sum_{n=0}^{N} A_n f(\gamma_n).$$

If the choice of nodes $\gamma_0, \gamma_1, \cdots, \gamma_n$ is made *a priori*, then in general the above formula is exact for polynomials of degree $\leqslant N$. However, if we are free to choose the nodes γ_n, we can expect quadrature formulas of the above form be exact for polynomials of degree up to $2N+1$.

There are three commonly used quadrature formulas. Each of them is associated

with a set of collocation points which are zeroes of a certain orthogonal polynomial. The first is the well-known Gauss quadrature which can be found in any elementary numerical analysis textbook.

`Gauss Quadrature` Let $x_0, x_1, \cdots, x_N$ be the zeroes of p_{N+1}. Then, the linear system

$$\sum_{j=0}^{N} p_k(x_j)\omega_j = \int_a^b p_k(x)\omega(x)\mathrm{d}x, \qquad 0 \leqslant k \leqslant N, \tag{1.2.17}$$

admits a unique solution $(\omega_0, \omega_1, \cdots, \omega_N)^t$, with $\omega_j > 0$ for $j = 0, 1, \cdots, N$. Furthermore,

$$\sum_{j=0}^{N} p(x_j)\omega_j = \int_a^b p(x)\omega(x)\mathrm{d}x, \quad \text{for all} \quad p \in P_{2N+1}. \tag{1.2.18}$$

The Gauss quadrature is the most accurate in the sense that it is impossible to find x_j, ω_j such that (1.2.18) holds for all polynomials $p \in P_{2N+2}$. However, by Lemma 1.2.1 this set of collocation points $\{x_i\}$ does not include the endpoint a or b, so it may cause difficulties for boundary value problems.

The second is the Gauss-Radau quadrature which is associated with the roots of the polynomial

$$q(x) = p_{N+1}(x) + \alpha p_N(x), \tag{1.2.19}$$

where α is a constant such that $q(a) = 0$. It can be easily verified that $q(x)/(x-a)$ is orthogonal to all polynomials of degree less than or equal to $N-1$ in $L^2_{\tilde{\omega}}(a, b)$ with $\tilde{\omega}(x) = \omega(x)(x-a)$. Hence, the N roots of $q(x)/(x-a)$ are all real, simple and lie in (a, b).

`Gauss-Radau Quadrature` Let $x_0 = a$ and $x_1, \cdots, x_N$ be the zeroes of $q(x)/(x-a)$, where $q(x)$ is defined by (1.2.19). Then, the linear system (1.2.17) admits a unique solution $(\omega_0, \omega_1, \cdots, \omega_N)^t$ with $\omega_j > 0$ for $j = 0, 1, \cdots, N$. Furthermore,

$$\sum_{j=0}^{N} p(x_j)\omega_j = \int_a^b p(x)\omega(x)\mathrm{d}x, \quad \text{for all} \quad p \in P_{2N}. \tag{1.2.20}$$

Similarly, one can construct a Gauss-Radau quadrature by fixing $x_N = b$. Thus, the Gauss-Radau quadrature is suitable for problems with one boundary point.

The third is the Gauss-Lobatto quadrature which is the most commonly used in

spectral approximations since the set of collocation points includes the two endpoints. Here, we consider the polynomial

$$q(x) = p_{N+1}(x) + \alpha p_N(x) + \beta p_{N-1}(x), \tag{1.2.21}$$

where α and β are chosen so that $q(a) = q(b) = 0$. One can verify that $q(x)/((x-a)(x-b))$ is orthogonal to all polynomials of degree less than or equal to $N-2$ in $L^2_{\hat{\omega}}(a,b)$ with $\hat{\omega}(x) = \omega(x)(x-a)(x-b)$. Hence, the $N-1$ zeroes of $q(x)/((x-a)(x-b))$ are all real, simple and lie in (a,b).

`Gauss-Lobatto Quadrature` Let $x_0 = a$, $x_N = b$ and $x_1, \cdots, x_{N-1}$ be the $(N-1)$-roots of $q(x)/((x-a)(x-b))$, where $q(x)$ is defined by (1.2.21). Then, the linear system (1.2.17) admits a unique solution $(\omega_0, \omega_1, \cdots, \omega_N)^t$, with $\omega_j > 0$, for $j = 0, 1, \cdots, N$. Furthermore,

$$\sum_{j=0}^{N} p(x_j)\omega_j = \int_a^b p(x)\omega(x)\mathrm{d}x, \quad \text{for all} \quad p \in P_{2N-1}. \tag{1.2.22}$$

Discrete inner product and discrete transform

For any of the Gauss-type quadratures defined above with the points and weights $\{x_j, \omega_j\}_{j=0}^N$, we can define a discrete inner product in $C[a,b]$ and its associated norm by:

$$(u,v)_{N,\omega} = \sum_{j=0}^{N} u(x_j)v(x_j)\omega_j, \ \|u\|_{N,\omega} = (u,u)_{N,\omega}^{\frac{1}{2}}, \tag{1.2.23}$$

and for $u \in C[a,b]$, we can write

$$u(x_j) = I_N u(x_j) = \sum_{k=0}^{N} \tilde{u}_k p_k(x_j). \tag{1.2.24}$$

One often needs to determine $\{\tilde{u}_k\}$ from $\{u(x_j)\}$ or vice versa. A naive approach is to consider (1.2.24) as a linear system with unknowns $\{\tilde{u}_k\}$ and use a direct method, such as Gaussian elimination, to determine $\{\tilde{u}_k\}$. This approach requires $\mathcal{O}(N^3)$ operations and is not only too expensive but also often unstable due to roundoff errors. We shall now describe a stable $\mathcal{O}(N^2)$-approach using the properties of orthogonal polynomials.

A direct consequence of Gauss-quadrature is the following:

Lemma 1.2.4 *Let $x_0, x_1, \cdots, x_N$ be the zeros of the orthogonal polynomial p_{N+1}, and let $\{\omega_j\}$ be the associated Gauss-quadrature weights. Then*

$$\sum_{n=0}^{N} p_i(x_n)p_j(x_n)\omega_n = 0, \qquad \textit{if} \quad i \neq j \leqslant N. \tag{1.2.25}$$

We derive from (1.2.24) and (1.2.25) that

$$\sum_{j=0}^{N} u(x_j)p_l(x_j)\omega_j = \sum_{j=0}^{N}\sum_{k=0}^{N} \tilde{u}_k p_k(x_j)p_l(x_j)\omega_j = \tilde{u}_l(p_l, p_l)_{N,\omega}. \tag{1.2.26}$$

Hence, assuming the values of $\{p_j(x_k)\}$ are precomputed and stored as an $(N+1)\times(N+1)$ matrix, the forward transform (1.2.24) and the backward transform (1.2.26) can be performed by a simple matrix-vector multiplication which costs $\mathcal{O}(N^2)$ operations. We shall see in later sections that the $\mathcal{O}(N^2)$ operations can be improved to $\mathcal{O}(N\log N)$ if special orthogonal polynomials are used.

Exercise 1.2

Problem 1 Let $\omega(x) \equiv 1$ and $(a, b) = (-1, 1)$. Derive the three-term recurrence relation and compute the zeros of the corresponding orthogonal polynomial $P_7(x)$.

Problem 2 Prove Lemma 1.2.2.

Problem 3 Prove Lemma 1.2.4.

1.3 Chebyshev and Legendre polynomials

Chebyshev polynomials
Discrete norm and discrete Chebyshev transform
Legendre polynomials
Zeros of the Legendre polynomials
Discrete norm and discrete Legendre transform

The two most commonly used sets of orthogonal polynomials are the Chebyshev and Legendre polynomials. In this section, we will collect some of their basic properties.

Chebyshev polynomials

The Chebyshev polynomials $\{T_n(x)\}$ are generated from (1.2.4) with $\omega(x) = (1-x^2)^{-\frac{1}{2}}$, $(a, b) = (-1, 1)$ and normalized with $T_n(1) = 1$. They satisfy the

following three-term recurrence relation

$$\begin{aligned} &T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \qquad n \geqslant 1, \\ &T_0(x) \equiv 1, \quad T_1(x) = x, \end{aligned} \tag{1.3.1}$$

and the orthogonality relation

$$\int_{-1}^{1} T_k(x)T_j(x)(1-x^2)^{-\frac{1}{2}}\mathrm{d}x = \frac{c_k\pi}{2}\delta_{kj}, \tag{1.3.2}$$

where $c_0 = 2$ and $c_k = 1$ for $k \geqslant 1$. A unique feature of the Chebyshev polynomials is their explicit relation with a trigonometric function:

$$T_n(x) = \cos\left(n\cos^{-1}x\right), \quad n = 0, 1, \cdots. \tag{1.3.3}$$

One may derive from the above many special properties, e.g., it follows from (1.3.3) that

$$\begin{aligned} &2T_n(x) = \frac{1}{n+1}T'_{n+1}(x) - \frac{1}{n-1}T'_{n-1}(x), \qquad n \geqslant 2, \\ &T_0(x) = T'_1(x), \quad 2T_1(x) = \frac{1}{2}T'_2(x). \end{aligned} \tag{1.3.4}$$

One can also infer from (1.3.3) that $T_n(x)$ has the same parity as n. Moreover, we can derive from (1.3.4) that

$$T'_n(x) = 2n \sum_{\substack{k=0 \\ k+n \text{ odd}}}^{n-1} \frac{1}{c_k}T_k(x), \qquad T''_n(x) = \sum_{\substack{k=0 \\ k+n \text{ even}}}^{n-2} \frac{1}{c_k}n(n^2-k^2)T_k(x). \tag{1.3.5}$$

By (1.3.3), it can be easily shown that

$$|T_n(x)| \leqslant 1, \qquad |T'_n(x)| \leqslant n^2, \tag{1.3.6a}$$

$$T_n(\pm 1) = (\pm 1)^n, \qquad T'_n(\pm 1) = (\pm 1)^{n-1}n^2, \tag{1.3.6b}$$

$$2T_m(x)T_n(x) = T_{m+n}(x) + T_{m-n}(x), \qquad m \geqslant n. \tag{1.3.6c}$$

The Chebyshev polynomials $\{T_k(x)\}$ can also be defined as the normalized eigenfunctions of the singular Sturm-Liouville problem

$$\left(\sqrt{1-x^2}T'_k(x)\right)' + \frac{k^2}{\sqrt{1-x^2}}T_k(x) = 0, \qquad x \in (-1, 1). \tag{1.3.7}$$

We infer from the above and (1.3.2) that

$$\int_{-1}^{1} T_k'(x)T_j'(x)\sqrt{1-x^2}\mathrm{d}x = \frac{c_k k^2 \pi}{2}\delta_{kj}, \tag{1.3.8}$$

i.e. the polynomials $\{T_k'(x)\}$ are mutually orthogonal with respect to the weight function $w(x) = \sqrt{1-x^2}$.

An important feature of the Chebyshev polynomials is that the Gauss-type quadrature points and weights can be expressed explicitly as follows.①

Chebyshev-Gauss:

$$x_j = \cos\frac{(2j+1)\pi}{2N+2}, \quad \omega_j = \frac{\pi}{N+1}, \quad 0 \leqslant j \leqslant N. \tag{1.3.9}$$

Chebyshev-Gauss-Radau:

$$x_0 = 1, \quad \omega_0 = \frac{\pi}{2N+1}, \quad x_j = \cos\frac{2\pi j}{2N+1}, \quad \omega_j = \frac{2\pi}{2N+1}, \quad 1 \leqslant j \leqslant N. \tag{1.3.10}$$

Chebyshev-Gauss-Lobatto:

$$x_0 = 1, \quad x_N = -1, \quad \omega_0 = \omega_N = \frac{\pi}{2N}, \quad x_j = \cos\frac{\pi j}{N}, \quad \omega_j = \frac{\pi}{N}, \quad 1 \leqslant j \leqslant N-1. \tag{1.3.11}$$

Discrete norm and discrete Chebyshev transform

For the discrete norm $\|\cdot\|_{N,\omega}$ associated with the Gauss or Gauss-Radau quadrature, we have $\|u\|_{N,\omega} = \|u\|_\omega$ for all $u \in P_N$. For the discrete norm $\|\cdot\|_{N,\omega}$ associated with the Chebyshev-Gauss-Lobatto quadrature, the following result holds.

Lemma 1.3.1 *For all $u \in P_N$,*

$$\|u\|_{L^2_\omega} \leqslant \|u\|_{N,\omega} \leqslant \sqrt{2}\|u\|_{L^2_\omega}. \tag{1.3.12}$$

Proof For $u = \sum_{k=0}^{N} \tilde{u}_k T_k$, we have

$$\|u\|^2_{L^2_\omega} = \sum_{k=0}^{N} \tilde{u}_k^2 \frac{c_k \pi}{2}. \tag{1.3.13}$$

On the other hand,

① For historical reasons and for simplicity of notation, the Chebyshev points are often ordered in descending order. We shall keep this convention in this book.

$$\|u\|_{N,\omega}^2 = \sum_{k=0}^{N-1} \tilde{u}_k^2 \frac{c_k \pi}{2} + \tilde{u}_N^2 \langle T_N, T_N \rangle_{N,\omega}. \tag{1.3.14}$$

The inequality (1.3.12) follows from the above results and the identity

$$(T_N, T_N)_{N,\omega} = \sum_{j=0}^{N} \frac{\pi}{\tilde{c}_j N} \cos^2 j\pi = \pi, \tag{1.3.15}$$

where $\tilde{c}_0 = \tilde{c}_N = 2$ and $\tilde{c}_k = 1$ for $1 \leqslant k \leqslant N-1$. □

Let $\{\xi_i\}_{i=0}^N$ be the Chebyshev-Gauss-Lobatto points, i.e. $\xi_i = \cos(i\pi/N)$, and let u be a continuous function on $[-1, 1]$. We write

$$u(\xi_i) = I_N u(\xi_i) = \sum_{k=0}^{N} \tilde{u}_k T_k(\xi_i) = \sum_{k=0}^{N} \tilde{u}_k \cos(ki\pi/N), \quad i = 0, 1, \cdots, N. \tag{1.3.16}$$

One derives immediately from the Chebyshev-Gauss-quadrature that

$$\tilde{u}_k = \frac{2}{\tilde{c}_k N} \sum_{j=0}^{N} \frac{1}{\tilde{c}_j} u(\xi_j) \cos(kj\pi/N). \tag{1.3.17}$$

The main advantage of using Chebyshev polynomials is that the backward and forward discrete Chebyshev transforms (1.3.16) and (1.3.17) can be performed in $\mathcal{O}(N \log_2 N)$ operations, thanks to the Fast Fourier Transform (FFT), see Section 1.5. The main disadvantage is that the Chebyshev polynomials are mutually orthogonal with respect to a singular weight function $(1-x^2)^{-\frac{1}{2}}$ which introduces significant difficulties in the analysis of the Chebyshev spectral method.

Legendre polynomials

The Legendre polynomials $\{L_n(x)\}$ are generated from (1.2.4) with $\omega(x) \equiv 1$, $(a, b) = (-1, 1)$ and the normalization $L_n(1) = 1$. The Legendre polynomials satisfy the three-term recurrence relation

$$\begin{aligned} &L_0(x) = 1, \qquad L_1(x) = x, \\ &(n+1)L_{n+1}(x) = (2n+1)xL_n(x) - nL_{n-1}(x), \quad n \geqslant 1, \end{aligned} \tag{1.3.18}$$

and the orthogonality relation

$$\int_{-1}^{1} L_k(x) L_j(x) \mathrm{d}x = \frac{1}{k + \frac{1}{2}} \delta_{kj}. \tag{1.3.19}$$

The Legendre polynomials can also be defined as the normalized eigenfunctions of the singular Sturm-Liouville problem

$$\big((1-x^2)L_n'(x)\big)' + n(n+1)L_n(x) = 0, \qquad x \in (-1,1), \tag{1.3.20}$$

from which and (1.3.19) we infer that

$$\int_{-1}^{1} L_k'(x)L_j'(x)(1-x^2)\mathrm{d}x = \frac{k(k+1)}{k+\frac{1}{2}}\delta_{kj}, \tag{1.3.21}$$

i.e. the polynomials $\{L_k'(x)\}$ are mutually orthogonal with respect to the weight function $\omega(x) = 1-x^2$.

Other useful properties of the Legendre polynomials include:

$$\int_{-1}^{x} L_n(\xi)\mathrm{d}\xi = \frac{1}{2n+1}(L_{n+1}(x) - L_{n-1}(x)), \qquad n \geqslant 1; \tag{1.3.22a}$$

$$L_n(x) = \frac{1}{2n+1}(L_{n+1}'(x) - L_{n-1}'(x)); \tag{1.3.22b}$$

$$L_n(\pm 1) = (\pm 1)^n, \qquad L_n'(\pm 1) = \frac{1}{2}(\pm 1)^{n-1}n(n+1); \tag{1.3.22c}$$

$$L_n'(x) = \sum_{\substack{k=0 \\ k+n \text{ odd}}}^{n-1} (2k+1)L_k(x); \tag{1.3.22d}$$

$$L_n''(x) = \sum_{\substack{k=0 \\ k+n \text{ even}}}^{n-2} \left(k+\frac{1}{2}\right)\left(n(n+1) - k(k+1)\right)L_k(x). \tag{1.3.22e}$$

For the Legendre series, the quadrature points and weights are

Legendre-Gauss: x_j are the zeros of $L_{N+1}(x)$, and

$$\omega_j = \frac{2}{(1-x_j^2)[L_{N+1}'(x_j)]^2}, \qquad 0 \leqslant j \leqslant N. \tag{1.3.23}$$

Legendre-Gauss-Radau: x_j are the $N+1$ zeros of $L_N(x) + L_{N+1}(x)$, and

$$\omega_0 = \frac{2}{(N+1)^2}, \qquad \omega_j = \frac{1}{(N+1)^2}\,\frac{1-x_j}{[L_N(x_j)]^2}, \quad 1 \leqslant j \leqslant N. \tag{1.3.24}$$

Legendre-Gauss-Lobatto: $x_0 = -1, x_N = 1$, $\{x_j\}_{j=1}^{N-1}$ are the zeros of $L_N'(x)$, and

$$\omega_j = \frac{2}{N(N+1)}\,\frac{1}{[L_N(x_j)]^2}, \qquad 0 \leqslant j \leqslant N. \tag{1.3.25}$$

Zeros of Legendre polynomials

We observe from the last subsection that the three types of quadrature points for the Legendre polynomials are related to the zeros of the L_{N+1}, $L_{N+1}+L_N$ and L'_N.

Theorem 1.2.1 provides a simple and efficient way to compute the zeros of orthogonal polynomials, given the three-term recurrence relation. However, this method may suffer from round-off errors as N becomes very large. As a result, we will present an alternative method to compute the zeros of $L_N^{(m)}(x)$ numerically, where $m < N$ is the order of derivative.

We start from the left boundary -1 and try to find the small interval of width H which contains the first zero z_1. The idea for locating the interval is similar to that used by the *bisection method*. In the resulting (small) interval, we use *Newton's method* to find the first zero. The Newton's method for finding a root of $f(x) = 0$ is

$$x_{k+1} = x_k - f(x_k)/f'(x_k). \tag{1.3.26}$$

After finding the first zero, we use the point $z_1 + H$ as the starting point and repeat the previous procedure to get the second zero z_2. This will give us all the zeros of $L_N^{(m)}(x)$. The parameter H, which is related to the smallest gap of the zeros, will be chosen as N^{-2}.

The following pseudo-code generates the zeros of $L_N^{(m)}(x)$.

```
CODE  LGauss.1
Input N, ϵ, m  %ϵ is the accuracy tolerence
H=N^-2; a=-1
For k=1 to N-m do
%The following is to search the small interval containing
a root
   b=a+H
   while L_N^(m)(a)*L_N^(m)(b) > 0
     a=b; b=a+H
   endwhile
%the Newton's method in (a,b)
        x=(a+b)/2;  xright=b
        while |x-xright|⩾ϵ
          xright=x; x=x-L_N^(m)(x)/L_N^(m+1)(x)
        endwhile
        z(k)=x
   a=x+H %move to another interval containing a root
endFor
Output z(1), z(2),···,z(N-m)
```

In the above pseudo-code, the parameter ϵ is used to control the accuracy of the zeros. Also, we need to use the recurrence formulas (1.3.18) and (1.3.22b) to obtain $L_n^{(m)}(x)$ which are used in the above code.

```
CODE LGauss.2
%This code is to evaluate L_n^(m)(x).
function r=Legendre(n,m,x)
For j=0 to m do
   If j=0 then
      s(0,j)=1;  s(1,j)=x
      for k=1 to n-1 do
        s(k+1,j)=((2k+1)*x*s(k,j)-k*s(k-1,j))/(k+1)
      endfor
   else  s(0,j)=0
      if j=1 then  s(1,j)=2
        else s(1,j)=0
      endif
      for k=1 to n-1 do
        s(k+1,j)=(2k+1)*s(k,j-1)+s(k-1,j)
      endfor
   endIf
endFor
r=s(n,m)
```

As an example, by setting $N = 7, m = 0$ and $\epsilon = 10^{-8}$ in `CODE LGauss.1`, we obtain the zeros for $L_7(x)$:

z_1	-0.94910791	z_5	0.40584515
z_2	-0.74153119	z_6	0.74153119
z_3	-0.40584515	z_7	0.94910791
z_4	0.00000000		

By setting $N = 6, m = 1$ and $\epsilon = 10^{-8}$ in `CODE LGauss.1`, we obtain the zeros for $L_6'(x)$. Together with $Z_1 = -1$ and $Z_7 = 1$, they form the Legendre-Gauss-Lobatto points:

Z_1	-1.00000000	Z_5	0.46884879
Z_2	-0.83022390	Z_6	0.83022390
Z_3	-0.46884879	Z_7	1.00000000
Z_4	0.00000000		

Discrete norm and discrete Legendre transform

As opposed to the Chebyshev polynomials, the main advantage of Legendre poly-

nomialsis that they are mutually orthogonal in the standard L^2-inner product, so the analysis of Legendre spectral methods is much easier than that of the Chebyshev spectral method. The main disadvantage is that there is no practical fast discrete Legendre transform available. However, it is possible to take advantage of both the Chebyshev and Legendre polynomials by constructing the so called Chebyshev-Legendre spectral methods; we refer to [41] and [141] for more details.

Lemma 1.3.2 *Let $\|\cdot\|_N$ be the discrete norm relative to the Legendre-Gauss-Lobatto quadrature. Then*

$$\|u\|_{L^2} \leqslant \|u\|_N \leqslant \sqrt{3}\|u\|_{L^2}, \quad \textit{for all} \quad u \in P_N. \tag{1.3.27}$$

Proof Setting $u = \sum_{k=0}^{N} \tilde{u}_k L_k$, we have from (1.3.19) that $\|u\|_{L^2}^2 = \sum_{k=0}^{N} 2\tilde{u}_k^2/(2k+1)$. On the other hand,

$$\|u\|_N^2 = \sum_{k=0}^{N-1} \tilde{u}_k^2 \frac{2}{2k+1} + \tilde{u}_N^2 (L_N, L_N)_N.$$

The desired result (1.3.27) follows from the above results, the identity

$$(L_N, L_N)_N = \sum_{j=0}^{N} L_N(x_j)^2 \omega_j = 2/N, \tag{1.3.28}$$

and the fact that $\frac{2}{2N+1} \leqslant \frac{2}{N} \leqslant 3\frac{2}{2N+1}$. □

Let $\{x_i\}_{0 \leqslant i \leqslant N}$ be the Legendre-Gauss-Lobatto points, and let u be a continuous function on $[-1, 1]$. We may write

$$u(x_j) = I_N u(x_j) = \sum_{k=0}^{N} \tilde{u}_k L_k(x_j). \tag{1.3.29}$$

We then derive from the Legendre-Gauss-Lobatto quadrature points that the discrete Legendre coefficients $\tilde{u}_k$ can be determined by the relation

$$\tilde{u}_k = \frac{1}{N+1} \sum_{j=0}^{N} u(x_j) \frac{L_k(x_j)}{L_N(x_j)}, \qquad k = 0, 1, \cdots, N. \tag{1.3.30}$$

The values $\{L_k(x_j)\}$ can be pre-computed and stored as a $(N+1)\times(N+1)$ matrix by using the three-term recurrence relation (1.3.18). Hence, the backward and forward

discrete Legendre transforms (1.3.30) and (1.3.29) can be performed by a matrix-vector multiplication which costs $\mathcal{O}(N^2)$ operations.

Exercise 1.3

Problem 1 Prove (1.3.22).

Problem 2 Derive the three-term recurrence relation for $\{L_k + L_{k+1}\}$ and use the method in Theorem 1.2.1 to find the Legendre-Gauss-Radau points with $N = 16$.

Problem 3 Prove (1.3.30).

1.4 Jacobi polynomials and generalized Jacobi polynomials

Basic properties of Jacobi polynomials
Generalized Jacobi polynomials

An important class of orthogonal polynomials are the so called Jacobi polynomials, which are denoted by $J_n^{\alpha,\beta}(x)$ and generated from (1.2.4) with

$$\omega(x) = (1-x)^\alpha (1+x)^\beta \quad \text{for } \alpha,\ \beta > -1,\ (a,b) = (-1,1), \tag{1.4.1}$$

and normalized by

$$J_n^{\alpha,\beta}(1) = \frac{\Gamma(n+\alpha+1)}{n!\Gamma(\alpha+1)}, \tag{1.4.2}$$

where $\Gamma(x)$ is the usual Gamma function. In fact, both the Chebyshev and Legendre polynomials are special cases of the Jacobi polynomials, namely, the Chebyshev polynomials $T_n(x)$ correspond to $\alpha = \beta = -\frac{1}{2}$ with the normalization $T_n(1) = 1$, and the Legendre polynomials $L_n(x)$ correspond to $\alpha = \beta = 0$ with the normalization $L_n(1) = 1$.

Basic properties of Jacobi polynomials

We now present some basic properties of the Jacobi polynomials which will be frequently used in the implementation and analysis of spectral methods. We refer to [155] for a complete and authoritative presentation of the Jacobi polynomials.

The three-term recurrence relation for the Jacobi polynomials is:

$$\begin{aligned} &J_{n+1}^{\alpha,\beta}(x) = (a_n^{\alpha,\beta} x - b_n^{\alpha,\beta}) J_n^{\alpha,\beta}(x) - c_n^{\alpha,\beta} J_{n-1}^{\alpha,\beta}(x), \quad n \geqslant 1, \\ &J_0^{\alpha,\beta}(x) = 1, \quad J_1^{\alpha,\beta}(x) = \frac{1}{2}(\alpha+\beta+2)x + \frac{1}{2}(\alpha-\beta), \end{aligned} \tag{1.4.3}$$

where

$$a_n^{\alpha,\beta} = \frac{(2n+\alpha+\beta+1)(2n+\alpha+\beta+2)}{2(n+1)(n+\alpha+\beta+1)}, \tag{1.4.4a}$$

$$b_n^{\alpha,\beta} = \frac{(\beta^2-\alpha^2)(2n+\alpha+\beta+1)}{2(n+1)(n+\alpha+\beta+1)(2n+\alpha+\beta)}, \tag{1.4.4b}$$

$$c_n^{\alpha,\beta} = \frac{(n+\alpha)(n+\beta)(2n+\alpha+\beta+2)}{(n+1)(n+\alpha+\beta+1)(2n+\alpha+\beta)}. \tag{1.4.4c}$$

The Jacobi polynomials satisfy the orthogonality relation

$$\int_{-1}^{1} J_n^{\alpha,\beta}(x) J_m^{\alpha,\beta}(x)(1-x)^{\alpha}(1+x)^{\beta}\mathrm{d}x = 0 \quad \text{for } n \neq m. \tag{1.4.5}$$

A property of fundamental importance is the following:

Theorem 1.4.1 *The Jacobi polynomials satisfy the following singular Sturm-Liouville problem*:

$$(1-x)^{-\alpha}(1+x)^{-\beta}\frac{d}{dx}\left\{(1-x)^{\alpha+1}(1+x)^{\beta+1}\frac{d}{dx}J_n^{\alpha,\beta}(x)\right\} + n(n+1+\alpha+\beta)J_n^{\alpha,\beta}(x) = 0, \quad -1 < x < 1.$$

Proof We denote $\omega(x) = (1-x)^{\alpha}(1+x)^{\beta}$. By applying integration by parts twice, we find that for any $\phi \in P_{n-1}$,

$$\int_{-1}^{1} \frac{\mathrm{d}}{\mathrm{d}x}\left\{(1-x)^{\alpha+1}(1+x)^{\beta+1}\frac{dJ_n^{\alpha,\beta}}{dx}\right\}\phi dx = -\int_{-1}^{1}\omega(1-x^2)\frac{\mathrm{d}J_n^{\alpha,\beta}}{dx}\frac{\mathrm{d}\phi}{\mathrm{d}x}\mathrm{d}x$$

$$= \int_{-1}^{1} J_n^{\alpha,\beta}\left\{[-(\alpha+1)(1+x)+(\beta+1)(1-x)]\frac{\mathrm{d}\phi}{\mathrm{d}x} + (1-x^2)\frac{\mathrm{d}^2\phi}{\mathrm{d}x^2}\right\}\omega\mathrm{d}x = 0.$$

The last equality follows from the fact that $\int_{-1}^{1} J_n^{\alpha,\beta}\psi\omega(x)\mathrm{d}x = 0$ for any $\psi \in P_{n-1}$. An immediate consequence of the above relation is that there exists λ such that

$$-\frac{d}{dx}\left\{(1-x)^{\alpha+1}(1+x)^{\beta+1}\frac{d}{dx}J_n^{\alpha,\beta}(x)\right\} = \lambda J_n^{\alpha,\beta}(x)\omega(x).$$

To determine λ, we take the coefficients of the leading term $x^{n+\alpha+\beta}$ in the above relation. Assuming that $J_n^{\alpha,\beta}(x) = k_n x^n + \{\text{lower order terms}\}$, we get $k_n n(n+1+$

$\alpha+\beta)=k_n\lambda$, which implies that $\lambda=n(n+1+\alpha+\beta)$. □

From Theorem 1.4.1 and (1.4.5), one immediately derives the following result:

Lemma 1.4.1 *For $n\neq m$,*

$$\int_{-1}^{1}(1-x)^{\alpha+1}(1+x)^{\beta+1}\frac{\mathrm{d}J_n^{\alpha,\beta}}{\mathrm{d}x}\frac{\mathrm{d}J_m^{\alpha,\beta}}{\mathrm{d}x}\mathrm{d}x=0. \quad \square \tag{1.4.6}$$

The above relation indicates that $\frac{d}{dx}J_n^{\alpha,\beta}$ forms a sequence of orthogonal polynomials with weight $\omega(x)=(1-x)^{\alpha+1}(1+x)^{\beta+1}$. Hence, by the uniqueness, we find that $\frac{d}{dx}J_n^{\alpha,\beta}$ is proportional to $J_{n-1}^{\alpha+1,\beta+1}$. In fact, we can prove the following important derivative recurrence relation:

Lemma 1.4.2 *For $\alpha,\beta>-1$,*

$$\partial_x J_n^{\alpha,\beta}(x)=\frac{1}{2}(n+\alpha+\beta+1)J_{n-1}^{\alpha+1,\beta+1}(x). \quad \square \tag{1.4.7}$$

Generalized Jacobi polynomials

Since for $\alpha\leqslant -1$ and/or $\beta\leqslant -1$, the function $\omega^{\alpha,\beta}$ is not in $L^1(I)$ so it cannot be used as a usual weight function. Hence, the classical Jacobi polynomials are only defined for $\alpha,\beta>-1$. However, as we shall see later, it is very useful to extend the definition of $J_n^{\alpha,\beta}$ to the cases where α and/or β are negative integers.

We now define the *generalized Jacobi polynomials* (GJPs) with integer indexes (k,l). Let us denote

$$n_0:=n_0(k,l)=\begin{cases}-(k+l) & \text{if } k,l\leqslant -1,\\ -k & \text{if } k\leqslant -1,\ l>-1,\\ -l & \text{if } k>-1,\ l\leqslant -1,\end{cases} \tag{1.4.8}$$

Then, the GJPs are defined as

$$J_n^{k,l}(x)=\begin{cases}(1-x)^{-k}(1+x)^{-l}J_{n-n_0}^{-k,-l}(x) & \text{if } k,l\leqslant -1,\\ (1-x)^{-k}J_{n-n_0}^{-k,l}(x) & \text{if } k\leqslant -1,\ l>-1,\\ (1+x)^{-l}J_{n-n_0}^{k,-l}(x) & \text{if } k>-1,\ l\leqslant -1,\end{cases}\qquad n\geqslant n_0. \tag{1.4.9}$$

It is easy to verify that $J_n^{k,l}\in P_n$.

We now present some important properties of the GJPs. First of all, it is easy to check that the GJPs are orthogonal with the generalized Jacobi weight $\omega^{k,l}$ for all

integers k and l, i.e.,

$$\int_{-1}^{1} J_n^{k,l}(x) J_m^{k,l}(x) \omega^{k,l}(x) \mathrm{d}x = 0, \quad \forall n \neq m. \tag{1.4.10}$$

It can be shown that the GJPs with negative integer indexes can be expressed as compact combinations of Legendre polynomials.

Lemma 1.4.3 *Let $k, l \geqslant 1$ and $k, l \in \mathbb{Z}$. There exists a set of constants $\{a_j\}$ such that*

$$J_n^{-k,-l}(x) = \sum_{j=n-k-l}^{n} a_j L_j(x), \quad n \geqslant k + l. \tag{1.4.11}$$

As some important special cases, one can verify that

$$\begin{aligned}
J_n^{-1,-1} &= \frac{2(n-1)}{2n-1}\Big(L_{n-2} - L_n\Big),\\
J_n^{-2,-1} &= \frac{2(n-2)}{2n-3}\Big(L_{n-3} - \frac{2n-3}{2n-1}L_{n-2} - L_{n-1} + \frac{2n-3}{2n-1}L_n\Big),\\
J_n^{-1,-2} &= \frac{2(n-2)}{2n-3}\Big(L_{n-3} + \frac{2n-3}{2n-1}L_{n-2} - L_{n-1} - \frac{2n-3}{2n-1}L_n\Big),\\
J_n^{-2,-2} &= \frac{4(n-1)(n-2)}{(2n-3)(2n-5)}\Big(L_{n-4} - \frac{2(2n-3)}{2n-1}L_{n-2} + \frac{2n-5}{2n-1}L_n\Big).
\end{aligned} \tag{1.4.12}$$

It can be shown (cf. [75]) that the generalized Jacobi polynomials satisfy the derivative recurrence relation stated in the following lemma.

Lemma 1.4.4 *For $k, l \in \mathbb{Z}$, we have*

$$\partial_x J_n^{k,l}(x) = C_n^{k,l} J_{n-1}^{k,l}(x), \tag{1.4.13}$$

where

$$C_n^{k,l} = \begin{cases} -2(n+k+l+1) & \text{if} \quad k, l \leqslant -1,\\ -n & \text{if} \quad k \leqslant -1,\ l > -1,\\ -n & \text{if} \quad k > -1,\ l \leqslant -1,\\ \dfrac{1}{2}(n+k+l+1) & \text{if} \quad k, l > -1. \end{cases} \tag{1.4.14}$$

Remark 1.4.1 Since $\omega^{\alpha,\beta} \notin L^1(I)$ for $\alpha \leqslant -1$ and $\beta \leqslant -1$, it is necessary that the generalized Jacobi polynomials vanish at one or both end points. In fact, an important feature of the GJPs is that for $k, l \geqslant 1$, we have

$$\begin{aligned} &\partial_x^i J_n^{-k,-l}(1) = 0, \qquad i = 0, 1, \cdots, k-1; \\ &\partial_x^j J_n^{-k,-l}(-1) = 0, \quad j = 0, 1, \cdots, l-1. \end{aligned} \tag{1.4.15}$$

Thus, they can be directly used as basis functions for boundary-value problems with corresponding boundary conditions.

Exercise 1.4

Problem 1 Prove (1.4.12) by the definition (1.4.9).

Problem 2 Prove Lemma 1.4.4.

1.5 Fast Fourier transform

Two basic lemmas
Computational cost
Tree diagram
Fast inverse Fourier transform
Fast Cosine transform
The discrete Fourier transform

Much of this section will be using complex exponentials. We first recall *Euler's formula*: $e^{i\theta} = \cos\theta + i\sin\theta$, where $i = \sqrt{-1}$. It is also known that the functions E_k defined by

$$E_k(x) = e^{ikx}, \qquad k = 0, \pm 1, \cdots \tag{1.5.1}$$

form an orthogonal system of functions in the complex space $L_2[0, 2\pi]$, provided that we define the inner-product to be

$$\langle f, g\rangle = \frac{1}{2\pi}\int_0^{2\pi} f(x)\overline{g(x)}\mathrm{d}x.$$

This means that $\langle E_k, E_m\rangle = 0$ when $k \neq m$, and $\langle E_k, E_k\rangle = 1$. For discrete values, it will be convenient to use the following inner-product notation:

$$\langle f, g\rangle_N = \frac{1}{N}\sum_{j=0}^{N-1} f\left(x_j\right)\overline{g\left(x_j\right)}, \tag{1.5.2}$$

where

$$x_j = 2\pi j/N, \quad 0 \leqslant j \leqslant N-1. \tag{1.5.3}$$

The above is not a true inner-product because the condition $\langle f, f\rangle_N = 0$ does not

imply $f \equiv 0$. It implies that $f(x)$ takes the value 0 at each node x_j.

The following property is important.

Lemma 1.5.1 *For any $N \geqslant 1$, we have*

$$\langle E_k, E_m \rangle_N = \begin{cases} 1 & \text{if } k-m \text{ is divisible by } N, \\ 0 & \text{otherwise.} \end{cases} \tag{1.5.4}$$

A 2π-periodic function $p(x)$ is said to be an *exponential polynomial* of degree at most n if it can be written in the form

$$p(x) = \sum_{k=0}^{n} c_k e^{ikx} = \sum_{k=0}^{n} c_k E_k(x). \tag{1.5.5}$$

The coefficients $\{c_k\}$ can be determined by taking the discrete inner-product of (1.5.5) with E_m. More precisely, it follows from (1.5.4) that the coefficients $c_0, c_1, \cdots, c_{N-1}$ in (1.5.5) can be expressed as:

$$c_k = \frac{1}{N} \sum_{j=0}^{N-1} f(x_j) e^{-ikx_j}, \qquad 0 \leqslant k \leqslant N-1, \tag{1.5.6}$$

where x_j is defined by (1.5.3). In practice, one often needs to determine $\{c_k\}$ from $\{f(x_j)\}$, or vice versa. It is clear that a direct computation using (1.5.6) requires $\mathcal{O}(N^2)$ operations. In 1965, a paper by Cooley and Tukey [33] described a different method of calculating the coefficients c_k, $0 \leqslant k \leqslant N-1$. The method requires only $\mathcal{O}(N \log_2 N)$ multiplications and $\mathcal{O}(N \log_2 N)$ additions, provided N is chosen in an appropriate manner. For a problem with thousands of data points, this reduces the number of calculations to thousands compared to millions for the direct technique.

The method described by Cooley and Tukey has become to be known either as the Cooley-Tukey Algorithm or the Fast Fourier Transform (FFT) Algorithm, and has led to a revolution in the use of interpolating trigonometric polynomials. We follow the exposition of Kincaid and Cheney [92] to introduce the algorithm.

Two basic lemmas

Lemma 1.5.2 *Let p and q be exponential polynomials of degree $N-1$ such that, for the points $y_j = \pi j/N$, we have*

$$p(y_{2j}) = f(y_{2j}), \qquad q(y_{2j}) = f(y_{2j+1}), \qquad 0 \leqslant j \leqslant N-1. \tag{1.5.7}$$

Then the exponential polynomial of degree $\leqslant 2N-1$ *that interpolates* f *at the points* $y_j, 0 \leqslant j \leqslant 2N-1$, *is given by*

$$P(x) = \frac{1}{2}(1+e^{iNx})p(x) + \frac{1}{2}(1-e^{iNx})q(x-\pi/N). \tag{1.5.8}$$

Proof Since p and q have degrees $\leqslant N-1$, whereas e^{jNx} is of degree N, it is clear that P has degree $\leqslant 2N-1$. It remains to show that P interpolates f at the nodes. We have, for $0 \leqslant j \leqslant 2N-1$,

$$P(y_j) = \frac{1}{2}(1+E_N(y_j))p(y_j) + \frac{1}{2}(1-E_N(y_j))q(y_j-\pi/N).$$

Notice that $E_N(y_j) = (-1)^j$. Thus for even j, we infer that $P(y_j) = p(y_j) = f(y_j)$, whereas for odd j, we have

$$P(y_j) = q(y_j - \pi/N) = q(y_{j-1}) = f(y_j).$$

This completes the proof of Lemma 1.5.2. □

Lemma 1.5.3 *Let the coefficients of the polynomials described in Lemma 1.5.2 be as follows*:

$$p = \sum_{j=0}^{N-1} \alpha_j E_j, \qquad q = \sum_{j=0}^{N-1} \beta_j E_j, \qquad P = \sum_{j=0}^{2N-1} \gamma_j E_j.$$

Then, for $0 \leqslant j \leqslant N-1$,

$$\gamma_j = \frac{1}{2}\alpha_j + \frac{1}{2}e^{-ij\pi/N}\beta_j, \quad \gamma_{j+N} = \frac{1}{2}\alpha_j - \frac{1}{2}e^{-ij\pi/N}\beta_j. \tag{1.5.9}$$

Proof To prove (1.5.9), we will be using (1.5.8) and will require a formula for $q(x-\pi/N)$:

$$q(x-\pi/N) = \sum_{j=0}^{N-1} \beta_j E_j(x-\pi/N) = \sum_{j=0}^{N-1} \beta_j e^{ij(x-\pi/N)} = \sum_{j=0}^{N-1} \beta_j e^{-i\pi j/N} E_j(x).$$

Thus, from equation (1.5.8),

$$P = \frac{1}{2}\sum_{j=0}^{N-1} \left\{(1+E_N)\alpha_j E_j + (1-E_N)\beta_j e^{-i\pi j/N} E_j\right\}$$

$$= \frac{1}{2} \sum_{j=0}^{N-1} \Big\{ (\alpha_j + \beta_j e^{-ij\pi/N}) E_j + (\alpha_j - \beta_j e^{-ij\pi/N}) E_{N+j} \Big\}.$$

The formulas for the coefficients γ_j can now be read from this equation. This completes the proof of Lemma 1.5.3. □

Computational cost

It follows from (1.5.6), (1.5.7) and (1.5.8) that

$$\alpha_j = \frac{1}{N} \sum_{j=0}^{N-1} f(x_{2j}) e^{-2\pi ij/N},$$

$$\beta_j = \frac{1}{N} \sum_{j=0}^{N-1} f(x_{2j+1}) e^{-2\pi ij/N},$$

$$\gamma_j = \frac{1}{2N} \sum_{j=0}^{2N-1} f(x_j) e^{-\pi ij/N}.$$

For the further analysis, let $R(N)$ denote the minimum number of multiplications necessary to compute the coefficients in an interpolating exponential polynomial for the set of points $\{2\pi j/N : 0 \leqslant j \leqslant N-1\}$.

First, we can show that

$$R(2N) \leqslant 2R(N) + 2N. \tag{1.5.10}$$

It is seen that $R(2N)$ is the minimum number of multiplications necessary to compute γ_j, and $R(N)$ is the minimum number of multiplications necessary to compute α_j or β_j. By Lemma 1.5.3, the coefficients γ_j can be obtained from α_j and β_j at the cost of $2N$ multiplications. Indeed, we require N multiplications to compute $\frac{1}{2}\alpha_j$ for $0 \leqslant j \leqslant N-1$, and another N multiplications to compute $(\frac{1}{2}e^{-ij\pi/N})\beta_j$ for $0 \leqslant j \leqslant N-1$. (In the latter, we assume that the factors $\frac{1}{2}e^{-ij\pi/N}$ have already been made available.) Since the cost of computing coefficients $\{\alpha_j\}$ is $R(N)$ multiplications, and the same is true for computing $\{\beta_j\}$, the total cost for P is at most $2R(N) + 2N$ multiplications. It follows from (1.5.10) and mathematical induction that $R(2^n) \leqslant m\, 2^m$. As a consequence of the above result, we see that if N is a power of 2, say 2^m, then the cost of computing the interpolating exponential polynomial obeys the inequality

$$R(N) \leqslant N \log_2 N.$$

The algorithm that carries out repeatedly the procedure in Lemma 1.5.2 is the fast Fourier transform.

Tree diagram

The content of Lemma 1.5.2 can be interpreted in terms of two linear operators, L_N and T_h. For any f, let $L_N f$ denote the exponential polynomial of degree $N-1$ that interpolates f at the nodes $2\pi j/N$ for $0 \leqslant j \leqslant N-1$. Let T_h be a *translation operator* defined by $(T_h f)(x) = f(x+h)$. We know from (1.5.4) that

$$L_N f = \sum_{k=0}^{N-1} < f,\ E_k >_N E_k.$$

Furthermore, in Lemma 1.5.2, $P = L_{2N} f, p = L_N f$ and $q = L_N T_{\pi/N} f$. The conclusion of Lemmas 1.5.2 and 1.5.3 is that $L_{2N} f$ can be obtained efficiently from $L_N f$ and $L_N T_{\pi/N} f$.

Our goal now is to establish one version of the fast Fourier transform algorithm for computing $L_N f$, where $N = 2^m$. We define

$$P_k^{(n)} = L_{2^n} T_{2k\pi/N} f, \qquad 0 \leqslant n \leqslant m,\ 0 \leqslant k \leqslant 2^{m-n} - 1. \tag{1.5.11}$$

An alternative description of $P_k^{(n)}$ is as the exponential polynomial of degree $2^n - 1$ that interpolates f in the following way:

$$P_k^{(n)}\left(\frac{2\pi j}{2^n}\right) = f\left(\frac{2\pi k}{N} + \frac{2\pi j}{2^n}\right), \qquad 0 \leqslant j \leqslant 2^n - 1.$$

A straightforward application of Lemma 1.5.2 shows that

$$P_k^{(n+1)}(x) = \frac{1}{2}\left(1 + e^{i2^n x}\right) P_k^{(n)} + \frac{1}{2}(1 - e^{i2^n x}) P_{k+2^{m-n-1}}^{(n)}\left(x - \frac{\pi}{2^n}\right). \tag{1.5.12}$$

We can illustrate in a *tree diagram* how the exponential polynomials $P_k^{(n)}$ are related. Suppose that our objective is to compute

$$P_0^{(3)} = L_8 f = \sum_{k=0}^{7} < f,\ E_k >_N E_k.$$

In accordance with (1.5.12), this function can be easily obtained from $P_0^{(2)}$ and $P_1^{(2)}$. Each of these, in turn, can be easily obtained from four polynomials of lower order,

and so on. Figure 1.2 shows the connections.

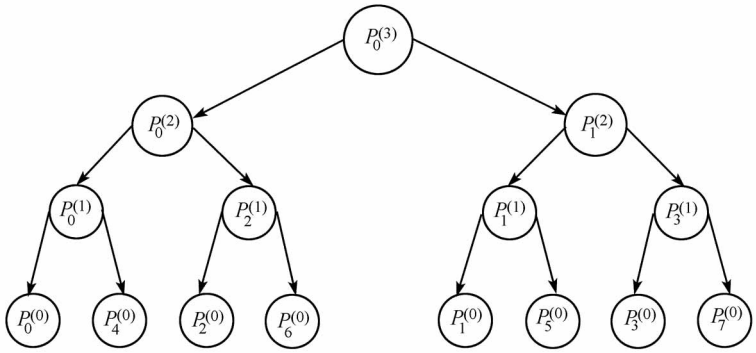

Figure 1.2 An illustration of a tree diagram

Algorithm

Denote the coefficients of $P_k^{(n)}$ by $A_{kj}^{(n)}$. Here $0 \leqslant n \leqslant m, 0 \leqslant k \leqslant 2^{m-n}-1$, and $0 \leqslant j \leqslant 2^n - 1$. We have

$$P_k^{(n)}(x) = \sum_{j=0}^{2^n-1} A_{kj}^{(n)} E_j(x) = \sum_{j=0}^{2^n-1} A_{kj}^{(n)} e^{ijx}.$$

By Lemma 1.5.3, the following equations hold:

$$A_{kj}^{(n+1)} = \frac{1}{2}\left[A_{kj}^{(n)} + e^{-ij\pi/2^n} A_{k+2^{m-n-1},\,j}^{(n)}\right],$$
$$A_{k,j+2^n}^{(n+1)} = \frac{1}{2}\left[A_{kj}^{(n)} - e^{-ij\pi/2^n} A_{k+2^{m-n-1},\,j}^{(n)}\right].$$

For a fixed n, the array $A^{(n)}$ requires $N = 2^m$ storage locations in memory because $0 \leqslant k \leqslant 2^{m-n}-1$ and $0 \leqslant j \leqslant 2^n - 1$. One way to carry out the computations is to use two linear arrays of length N, one to hold $A^{(n)}$ and the other to hold $A^{(n+1)}$. At the next stage, one array will contain $A^{(n+1)}$ and the other $A^{(n+2)}$. Let us call these arrays C and D. The two-dimensional array $A^{(n)}$ is stored in C by the rule

$$C(2^n k + j) = A_{kj}^{(n)}, \qquad 0 \leqslant k \leqslant 2^{m-n}-1, \quad 0 \leqslant j \leqslant 2^n - 1.$$

It is noted that if $0 \leqslant k, k' \leqslant 2^{m-n}-1$ and $0 \leqslant j, j' \leqslant 2^n - 1$ satisfying $2^n k + j =$

$2^n k' + j'$, then $(k, j) = (k', j')$. Similarly, the array $A^{(n+1)}$ is stored in D by the rule

$$D(2^{n+1}k + j) = A_{kj}^{(n+1)}, \qquad 0 \leqslant k \leqslant 2^{m-n-1} - 1, \quad 0 \leqslant j \leqslant 2^{n+1} - 1.$$

The factors $Z(j) = e^{-2\pi ij/N}$ are computed at the beginning and stored. Then we use the fact that

$$e^{-ij\pi/2^n} = Z(j2^{m-n-1}).$$

Below is the fast Fourier transform algorithm:

```
CODE FFT.1
% Cooley-Tukey Algorithm
Input  m
N=2^m, w=e^{-2πi/N}
for k=0 to N-1 do
    Z(k)=w^k, C(k)=f(2πk/N)
endfor
For n=0 to m-1 do
    for k=0 to 2^{m-n-1}-1 do
        for j=0 to 2^n-1 do
            u=C(2^n k+j); v=Z(j2^{m-n-1})*C(2^n k+2^{m-1}+j)
            D(2^{n+1}k+j)=0.5*(u+v); D(2^{n+1}k+j+2^n)=0.5*(u-v)
        endfor
    endfor
    for j=0 to N-1 do
        C(j)=D(j)
    endfor
endFor
Output C(0), C(1), ···, C(N-1).
```

By scrutinizing the pseudocode, we can also verify the bound $N \log_2 N$ for the number of multiplications involved. Notice that in the nested loop of the code, n takes on m values; then k takes on 2^{m-n-1} values, and k takes on 2^n values. In this part of the code, there is really just one command involving a multiplication, namely, the one in which v is computed. This command will be encountered a number of times equal to the product $m \times 2^{m-n-1} \times 2^n = m2^{m-1}$. At an earlier point in the code, the computation of the Z-array involves $2^n - 1$ multiplications. On any binary computer, a multiplication by $1/2$ need not be counted because it is accomplished by subtracting 1 from the exponent of the floating-point number. Therefore, the total number of multiplications used in `CODE FFT.1` is

$$m2^{m-1} + 2^m - 1 \leqslant m2^m = N \log_2 N.$$

Fast inverse Fourier transform

The fast Fourier transform can also be used to evaluate the inverse transform:

$$d_k = \frac{1}{N}\sum_{j=0}^{N-1} g(x_j)e^{ikx_j}, \qquad 0 \leqslant k \leqslant N-1.$$

Let $j = N-1-m$. It is easy to verify that

$$d_k = e^{-ix_k}\frac{1}{N}\sum_{m=0}^{N-1} g(x_{N-1-m})e^{-ikx_m}, \qquad 0 \leqslant k \leqslant N-1.$$

Thus, we apply the FFT algorithm to get $e^{ix_k}d_k$. Then extra N operations give d_k. A pseudocode for computing d_k is given below.

```
CODE FFT.2
% Fast Inverse Fourier Transform
Input m
N=2^m, w=e^{-2πi/N}
for k=0 to N-1 do
    Z(k)=w^k,  C(k)=g(2π(N-1-k)/N)
end
For n=0 to m-1 do
    for k=0 to 2^{m-n-1}-1 do
        for j=0 to 2^n-1 do
            u=C(2^n k+j); v=Z(j2^{m-n-1})*C(2^n k+2^{m-1}+j)
            D(2^{n+1}k+j)=0.5*(u+v); D(2^{n+1}k+j+2^n)=0.5*(u-v)
        endfor
    endfor
    for j=0 to N-1 do
        C(j)=D(j)
    endfor
endFor
for k=0 to N-1 do
    D(k)=Z(k)*C(k)
endfor
Output D(0), D(1), ···, D(N-1).
```

Fast Cosine transform

The fast Fourier transform can also be used to evaluate the cosine transform:

$$a_k = \sum_{j=0}^{N} f(x_j)\cos\left(\pi jk/N\right), \qquad 0 \leqslant k \leqslant N,$$

where the $f(x_j)$ are *real numbers*. Let $v_j = f(x_j)$ for $0 \leqslant j \leqslant N$ and $v_j = 0$ for $N+1 \leqslant j \leqslant 2N-1$. We compute

$$A_k = \frac{1}{2N} \sum_{j=0}^{2N-1} v_j e^{-ikx_j}, \qquad x_j = \frac{\pi j}{N}, \quad 0 \leqslant k \leqslant 2N-1.$$

Since the v_j are real numbers and $v_j = 0$ for $j \geqslant N+1$, it can be shown that the real part of A_k is

$$\mathrm{Re}(A_k) = \frac{1}{2N} \sum_{j=0}^{N} f(x_j) \cos(\pi jk/N), \qquad 0 \leqslant k \leqslant 2N-1.$$

In other words, the following results hold: $a_k = 2N\mathrm{Re}(A_k)$, $0 \leqslant k \leqslant N$. By the definition of the A_k, we know that they can be computed by using the pseudocode FFT.1. When they are multiplied by $2N$, we have the values of a_k.

Numerical examples

To test the the efficiency of the FFT algorithm, we compute the coefficients in (1.5.6) using `CODE FFT.1` and the direct method. A subroutine for computing the coefficients directly from the formulas goes as follows:

```
CODE FFT.3
% Direct method for computing the coefficients
Input  m
N=2^m, w=e^{-2πi/N}
for k=0 to N-1 do
    Z(k)=w^k, D(k)=f(2πk/N)
endfor
for n=0 to N-1 do
    u=D(0)+Σ_{k=1}^{N-1} D(k)*Z(n)^k
    C(n)=u/N
endfor
Output  C(0), C(1), ···, C(N-1)
```

The computer programs based on `CODE FFT.1` and `CODE FFT.2` are written in FORTRAN with double precision. We compute the following coefficients:

$$c_k = \frac{1}{N} \sum_{j=0}^{N-1} \cos(5x_j) e^{-ikx_j}, \qquad 0 \leqslant k \leqslant N-1,$$

where $x_j = 2\pi j/N$. The CPU time used are listed in the following table.

m	N	CPU (FFT)	CPU(direct)
9	512	0.02	0.5
10	1024	0.04	2.1
11	2048	0.12	9.0
12	4096	0.28	41.0
13	8192	0.60	180.0

The discrete Fourier transform

Again let f be a 2π-periodic function defined in $[0, 2\pi]$. The Fourier transform of $f(t)$ is defined as

$$H(s) = \mathcal{F}\{f(t)\} = \frac{1}{2\pi}\int_0^{2\pi} f(t)e^{-ist}\mathrm{d}t, \tag{1.5.13}$$

where s is a real parameter and $\mathcal{F}$ is called the Fourier transform operator. The inverse Fourier transform is denoted by $\mathcal{F}^{-1}\{H(s)\}$,

$$f(t) = \mathcal{F}^{-1}\{H(s)\} = \int_{-\infty}^{\infty} e^{ist}H(s)\mathrm{d}s,$$

where $\mathcal{F}^{-1}$ is called the inverse Fourier transform operator. The following result is important: The Fourier transform operator $\mathcal{F}$ is a linear operator satisfying

$$\mathcal{F}\{f^{(n)}(t)\} = (ik)^n\mathcal{F}\{f(t)\}, \tag{1.5.14}$$

where $f^{(n)}(t)$ denotes the n-th order derivative of $f(t)$. Similar to the continuous Fourier transform, we will define the discrete Fourier transform below. Let the solution interval be $[0, 2\pi]$. We first transform $u(x, t)$ into the discrete Fourier space:

$$\hat{u}(k, t) = \frac{1}{N}\sum_{j=0}^{N-1} u(x_j, t)e^{-ikx_j}, \qquad -\frac{N}{2} \leqslant k \leqslant \frac{N}{2} - 1, \tag{1.5.15}$$

where $x_j = 2\pi j/N$. Due to the orthogonality relation (1.5.4),

$$\frac{1}{N}\sum_{j=0}^{N-1} e^{ipx_j} = \begin{cases} 1 & \text{if } p = Nm, m = 0, \pm1, \pm2, \cdots, \\ 0 & \text{otherwise}, \end{cases}$$

we have the inversion formula

$$u(x_j,t)=\sum_{k=-N/2}^{N/2-1}\hat{u}(k,t)e^{ikx_j},\qquad 0\leqslant j\leqslant N-1. \tag{1.5.16}$$

We close this section by pointing out there are many useful developments on fast transforms by following similar spirits of the FFT methods; see e.g. [124], [126], [2], [150], [21], [65], [123], [143].

Exercise 1.5

Problem 1 Prove (1.5.4).

Problem 2 One of the most important uses of the FFT algorithm is that it allows periodic discrete convolutions of vectors of length n to be performed in $\mathcal{O}(n\log n)$ operations.

To keep the notation simple, let us consider $n=4$ (the proof below carries through in just the same way for any size). Use the fact that

$$\begin{bmatrix}1&1&1&1\\1&\omega&\omega^2&\omega^3\\1&\omega^2&\omega^4&\omega^6\\1&\omega^3&\omega^6&\omega^9\end{bmatrix}\begin{bmatrix}\hat{u}_0\\\hat{u}_1\\\hat{u}_2\\\hat{u}_3\end{bmatrix}=\begin{bmatrix}u_0\\u_1\\u_2\\u_3\end{bmatrix},$$

is quivalent to

$$\frac{1}{n}\begin{bmatrix}1&1&1&1\\1&\omega^{-1}&\omega^{-2}&\omega^{-3}\\1&\omega^{-2}&\omega^{-4}&\omega^{-6}\\1&\omega^{-3}&\omega^{-6}&\omega^{-9}\end{bmatrix}\begin{bmatrix}u_0\\u_1\\u_2\\u_3\end{bmatrix}=\begin{bmatrix}\hat{u}_0\\\hat{u}_1\\\hat{u}_2\\\hat{u}_3\end{bmatrix},$$

where $\omega=e^{2\pi i/n}$, prove that the linear system

$$\begin{bmatrix}z_0&z_3&z_2&z_1\\z_1&z_0&z_3&z_2\\z_2&z_1&z_0&z_3\\z_3&z_2&z_1&z_0\end{bmatrix}\begin{bmatrix}x_0\\x_1\\x_2\\x_3\end{bmatrix}=\begin{bmatrix}y_0\\y_1\\y_2\\y_3\end{bmatrix}$$

where $\{z_0,z_1,z_2,z_3\}$ is an arbitrary vector, can be transformed to a simple system of

the form

$$\begin{bmatrix} \hat{z}_0 & & & \\ & \hat{z}_1 & & \\ & & \hat{z}_2 & \\ & & & \hat{z}_3 \end{bmatrix} \begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \hat{x}_2 \\ \hat{x}_3 \end{bmatrix} = \frac{1}{n} \begin{bmatrix} \hat{y}_0 \\ \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \end{bmatrix}.$$

1.6 Several popular time discretization methods

General Runge-Kutta methods
Stability of Runge-Kutta methods
Multistep methods
Backward difference methods (BDF)
Operator splitting methods

We present in this section several popular time discretization methods, which will be repeatedly used in this book, for a system of ordinary differential equations

$$\frac{dU}{dt} = F(U,t), \tag{1.6.1}$$

where $U \in \mathbb{R}^d, F \in \mathbb{R}^d$. An initial condition is also given to the above problem:

$$U(t_0) = U_0. \tag{1.6.2}$$

The simplest method is to approximate dU/dt by the finite difference quotient $U'(t)$ $\approx [U(t+\Delta t) - U(t)]/\Delta t$. Since the starting data is known from the initial condition $U^0 = U_0$, we can obtain an approximation to the solution at $t_1 = t_0 + \Delta t$: $U^1 = U^0 + \Delta t\, F(U^0, t_0)$. The process can be continued. Let $t_k = t_0 + k\Delta t$, $k \geqslant 1$. Then the approximation U_{k+1} to the solution $U(t_{k+1})$ is given by

$$U^{n+1} = U^n + \Delta t\, F(U^n, t_n), \tag{1.6.3}$$

where $U^n \approx U(\cdot,\ t_n)$. The above algorithm is called the *Euler method*. It is known that if the function F has a bounded partial derivative with respect to its second variable and if the solution U has a bounded second derivative, then the Euler method converges to the exact solution with first order of convergence, namely,

$$\max_{1 \leqslant n \leqslant N} |U^n - U(t_n)| \leqslant C\Delta t,$$

where C is independent of N and Δt.

The conceptually simplest approach to higher-order methods is to use more terms in the Taylor expansion. Compared with the Euler method, one more term is taken so that

$$U(t_{n+1}) \approx U(t_n) + \Delta t\, U'(t_n) + \frac{\Delta t^2}{2} U''(t_n), \tag{1.6.4}$$

where the remainder of $\mathcal{O}(\Delta t^3)$ has been dropped. It follows from (1.6.1) that $U'(t_n)$ can be replaced by $F(U^n, t_n)$. Moreover,

$$U''(t) = \frac{d}{dt} F(U(t), t) = F_U(U, t) U'(t) + F_t(U, t)\,,$$

which yields

$$U''(t_n) \approx F_U(U^n, t_n) F(U^n, t_n) + F_t(U^n, t_n).$$

Using this to replace $U''(t_n)$ in (1.6.4) leads to the method

$$U^{n+1} = U^n + \Delta t F(U^n, t_n) + \frac{\Delta t^2}{2} [F_t(U^n, t_n) + F_U(U^n, t_n) F(U^n, t_n)]. \tag{1.6.5}$$

It can be shown the above scheme has second-order order accuracy provided that F and the underlying solution U are smooth.

General Runge-Kutta methods

Instead of computing the partial derivatives of F, we could also obtain higher-order methods by making more evaluations of the function values of F at each step. A class of such schemes is known as Runge-Kutta methods. The second-order Runge-Kutta method is of the form:

$$\begin{cases} U = U^n, & G = F(U, t_n), \\ U = U + \alpha \Delta t G, & G = (-1 + 2\alpha - 2\alpha^2) G + F(U, t_n + \alpha \Delta t), \\ U^{n+1} = U + \dfrac{\Delta t}{2\alpha} G. \end{cases} \tag{1.6.6}$$

Only two levels of storage (U and G) are required for the above algorithm. The choice $\alpha = 1/2$ produces the modified Euler method, and $\alpha = 1$ corresponds to the Heun method.

The third-order Runge-Kutta method is given by:

$$\begin{cases} U = U^n, & G = F(U, t_n), \\ U = U + \dfrac{1}{3}\Delta t G, & G = -\dfrac{5}{9}G + F\left(U, t_n + \dfrac{1}{3}\Delta t\right), \\ U = U + \dfrac{15}{16}\Delta t G, & G = -\dfrac{153}{128}G + F\left(U, t_n + \dfrac{3}{4}\Delta t\right), \\ U^{n+1} = U + \dfrac{8}{15}G. \end{cases} \tag{1.6.7}$$

Only two levels of storage (U and G) are required for the above algorithm.

The classical fourth-order Runge-Kutta (RK4) method is

$$\begin{cases} K_1 = F(U^n, t_n), & K_2 = F\left(U^n + \dfrac{\Delta t}{2}K_1, t_n + \dfrac{1}{2}\Delta t\right), \\ K_3 = F\left(U^n + \dfrac{\Delta t}{2}K_2, t_n + \dfrac{1}{2}\Delta t\right), & K_4 = F(U^n + \Delta t K_3, t_{n+1}), \\ U^{n+1} = U^n + \dfrac{\Delta t}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right). \end{cases} \tag{1.6.8}$$

The above formula requires four levels of storage, i.e. K_1, K_2, K_3 and K_4. An equivalent formulation is

$$\begin{cases} U = U^n, & G = U, \quad P = F(U, t_n), \\ U = U + \dfrac{1}{2}\Delta t P, & G = P, \quad P = F\left(U, t_n + \dfrac{1}{2}\Delta t\right), \\ U = U + \dfrac{1}{2}\Delta t(P - G), & G = \dfrac{1}{6}G, \quad P = F\left(U, t_n + \Delta t/2\right) - P/2, \\ U = U + \Delta t P, & G = G - P, \quad P = F(U, t_{n+1}) + 2P, \\ U^{n+1} = U + \Delta t\left(G + P/6\right). \end{cases} \tag{1.6.9}$$

This version of the RK4 method requires only three levels (U, G and P) of storage.

As we saw in the derivation of the Runge-Kutta method of order 2, a number of parameters must be selected. A similar process occurs in establishing higher-order Runge-Kutta methods. Consequently, there is not just one Runge-Kutta method for each order, but a family of methods. As shown in the following table, the number of required *function evaluations* increases more rapidly than the order of the Runge-Kutta methods:

Number of function evaluations	1	2	3	4	5	6	7	8
Maximum order of RK method	1	2	3	4	4	5	6	6

Unfortunately, this makes the higher-order Runge-Kutta methods less attractive than the classical fourth-order method, since they are more expensive to use.

The Runge-Kutta procedure for systems of first-order equations is most easily written down in the case when the system is *autonomous*; that is, it has the form

$$\frac{dU}{dt} = F(U). \tag{1.6.10}$$

The classical RK4 formulas, in vector form, are

$$U^{n+1} = U^n + \frac{\Delta t}{6}\Big(K_1 + 2K_2 + 2K_3 + K_4\Big), \tag{1.6.11}$$

where

$$\begin{cases} K_1 = F(U^n), \quad K_2 = F\left(U^n + \dfrac{\Delta t}{2}K_1\right), \\ K_3 = F\left(U^n + \dfrac{\Delta t}{2}K_2\right), \quad K_4 = F\left(U^n + \Delta t K_3\right). \end{cases}$$

For problems without source terms such as Examples 5.3.1 and 5.3.2, we will end up with an autonomous system. The above RK4 method, or its equivalent form similar to (1.6.9), can be used.

Stability of Runge-Kutta methods

The general s-stage explicit Runge-Kutta method of maximum order s has stability function

$$r(z) = 1 + z + \frac{z^2}{2} + \cdots + \frac{z^s}{s!}, \quad s = 1, 2, 3, 4. \tag{1.6.12}$$

There are a few stability concepts for the Runge-Kutta methods:

a. The region of absolute stability $\mathcal{R}$ of an s-order Runge-Kutta method is the set of points $z = \lambda\Delta t \in \mathbb{C}$ such that if $z \in \mathcal{R}$, $(\mathcal{R}e(\lambda) < 0)$. Then the numerical method applied to

$$\frac{du}{dt} = \lambda u \tag{1.6.13}$$

gives $u^n \to 0$ as $n \to \infty$. It can be shown that the region of absolute stability of a

Runge-Kutta method is given by

$$\mathcal{R} = \{z \in \mathbb{C} \mid |r(z)| < 1\}. \tag{1.6.14}$$

b. A Runge-Kutta method is said to be A-stable if its stability region contains the left-half of the complex plane, i.e. the non-positive half-plane, $\mathbb{C}^-$.

c. A Runge-Kutta method is said to be L-stable if it is A-stable, and if its stability function $r(z)$ satisfies

$$\lim_{|z|\to\infty} |r(z)| = 0. \tag{1.6.15}$$

In Figure 1.3, we can see that the stability domains for these explicit Runge-Kutta methods consist of the interior of closed regions in the left-half of the complex plane. The algorithm for plotting the absolute stability regions above can be found in the book by Butcher [27]. Notice that all Runge-Kutta methods of a given order have the same stability properties. The stability regions expand as the order increases.

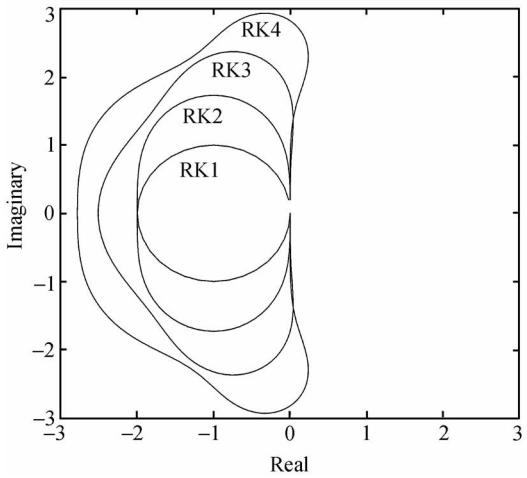

Figure 1.3 Absolute stability regions of Runge-Kutta methods

Multistep methods

Another approach to higher-order methods utilizes information already computed and does not require additional evaluations of $F(U,t)$. One of the simplest such methods is

$$U^{n+1} = U_n + \frac{\Delta t}{2}[3F(U^n, t_n) - F(U^{n-1}, t_{n-1})], \tag{1.6.16}$$

for which the maximum pointwise error is $\mathcal{O}(\Delta t^2)$, and is known as the second-order

Adams-Bashforth method, or AB2 for short. Note that the method requires only the evaluation of $F(U^n, t_n)$ at each step, the value $F(U^{n-1}, t_{n-1})$ being known from the previous step.

We now consider the general construction of Adams-Bashforth methods. Let $U^n, U^{n-1}, \cdots, U^{n-s}$ be the computed approximations to the solution at t_n, t_{n-1}, $\cdots, t_{n-s}$. Let $F^i = F(U^i, t_i)$ and let $p(t)$ be the interpolating polynomial of degree s that satisfies

$$p(t_i) = F^i, \qquad i = n, n-1, \cdots, n-s.$$

We may then consider $p(t)$ to be an approximation to $F(U(t), t)$. Since the solution $U(t)$ satisfies

$$U(t_{n+1}) - U(t_n) = \int_{t_n}^{t_{n+1}} U'(t)\mathrm{d}t = \int_{t_n}^{t_{n+1}} F(U(t), t)\mathrm{d}t \approx \int_{t_n}^{t_{n+1}} p(t)\mathrm{d}t,$$

we obtain the so-called Adams-Bashforth (AB) methods as follows:

$$U^{n+1} = U^n + \int_{t_n}^{t_{n+1}} p(t)\mathrm{d}t. \tag{1.6.17}$$

Below we provide a few special cases of the Adams-Bashforth methods:

- $s = 0$: $p(t) = F_n$ for $t \in [t_n, t_{n+1})$, gives Euler method.
- $s = 1$:

$$p(t) = p_1(t) = U^n + \frac{t - t_n}{\Delta t}(F^n - F^{n-1}),$$

which leads to the second-order Adams-Bashforth method (1.6.16).

- $s = 2$:

$$p_2(t) = p_1(t) + \frac{(t - t_n)(t - t_{n-1})}{2\Delta t^2}(F^n - 2F^{n-1} + F^{n-2}),$$

which leads to the third-order Adams-Bashforth method

$$U^{n+1} = U^n + \frac{\Delta t}{12}(23F^n - 16F^{n-1} + 5F^{n-2}). \tag{1.6.18}$$

- $s = 3$:

$$p_3(t) = p_2(t) - \frac{(t - t_n)(t - t_{n-1})(t - t_{n-2})}{3!\Delta t^3}(F^n - 3F^{n-1} + 3F^{n-2} - F^{n-3}),$$

which leads to the fourth-order Adams-Bashforth method

$$U^{n+1} = U^n + \frac{\Delta t}{24}(55F^n - 59F^{n-1} + 37F^{n-2} - 9F^{n-3}). \tag{1.6.19}$$

In principle, we can continue the preceding process to obtain Adams-Bashforth methods of arbitrarily high-order, but the formulas become increasingly complex as d increases. The Adams-Bashforth methods are multistep methods since two or more levels of prior data are used. This is in contrast to the Runge-Kutta methods which use no prior data and are called one-step methods. We will compute the numerical solutions of the KdV equation using a multistep method (see Sect. 5.4).

Multistep methods cannot start by themselves. For example, consider the fourth-order Adams-Bashforth method. The initial value U^0 is given, but for $k = 0$, the information is needed at t_{-1}, t_{-2}, t_{-3}, which is not available. The method needs "help" getting started. We cannot use the fourth-order multistep method until $k \geqslant 3$. A common policy is to use a one-step method, such as a Runge-Kutta method of the same order of accuracy at some starting steps.

Since the Adams-Bashforth methods of arbitrary order require only one evaluation of $F(U, t)$ at each step, the "cost" is lower than that of Runge-Kutta methods. On the other hand, in Runge-Kutta methods it is much easier to change step-size; hence they are more suitable for use in an adaptive algorithm.

Backward difference methods (BDF)

The Adams-Bashforth methods can be unstable due to the fact they are obtained by integrating the interpolating polynomial outside the interval of the data that defines the polynomial. This can be remedied by using multilevel implicit methods:

- Second-order backward difference method (BD2):

$$\frac{1}{2\Delta t}(3U^{n+1} - 4U^n + U^{n-1}) = F(U^{n+1}, t_{n+1}). \tag{1.6.20}$$

- Third-order backward difference method (BD3):

$$\frac{1}{6\Delta t}(11U^{n+1} - 18U^n + 9U^{n-1} - 2U^{n-2}) = F(U^{n+1}, t_{n+1}). \tag{1.6.21}$$

In some practical applications, $F(u, t)$ is often the sum of linear and nonlinear terms. In this case, some combination of the backward difference method and extrapolation method can be used. To fix the idea, let us consider

$$u_t = \mathcal{L}(u) + \mathcal{N}(u), \tag{1.6.22}$$

where $\mathcal{L}$ is a linear operator and $\mathcal{N}$ is a nonlinear operator. By combining a second-order backward differentiation (BD2) for the time derivative term and a second-order extrapolation (EP2) for the explicit treatment of the nonlinear term, we arrive at a second-order scheme (BD2/EP2) for (1.6.22):

$$\frac{1}{2\Delta t}(3U^{n+1} - 4U^n + U^{n-1}) = \mathcal{L}(U^{n+1}) + \mathcal{N}(2U^n - U^{n-1}). \tag{1.6.23}$$

A third-order scheme for solving (1.6.22) can be constructed in a similar manner, which leads to the so-called BD3/EP3 scheme:

$$\frac{1}{6\Delta t}(11U^{n+1} - 18U^n + 9U^{n-1} - 2U^{n-2}) = \mathcal{L}(U^{n+1}) + \mathcal{N}(3U^n - 3U^{n-1} + U^{n-2}). \tag{1.6.24}$$

Operator splitting methods

In many practical situations, $F(u,t)$ is often the sum of several terms with different properties. Then it is often advisable to use an operator splitting method (also called fractional step method)[171, 119, 57, 154]. To fix the idea, let us consider

$$u_t = f(u) = Au + Bu, \quad u(t_0) = u_0, \tag{1.6.25}$$

where $f(u)$ is a nonlinear operator and the splitting $f(u) = Au + Bu$ can be quite arbitrary; in particular, A and B do not need to commute.

`Strang's operator splitting method` For a given time step $\Delta t > 0$, let $t_n = n\ \Delta t$, $n = 0, 1, 2, \cdots$ and u^n be the approximation of $u(t_n)$. Let us formally write the solution $u(x,t)$ of (1.6.25) as

$$u(t) = e^{t(A+B)}u_0 =: S(t)u_0. \tag{1.6.26}$$

Similarly, denote by $S_1(t) := e^{tA}$ the solution operator for $u_t = Au$, and by $S_2(t) := e^{tB}$ the solution operator for $u_t = Bu$. Then the first-order operator splitting is based on the approximation

$$u^{n+1} \approx S_2(\Delta t)S_1(\Delta t)u^n, \tag{1.6.27}$$

or on the one with the roles of S_2 and S_1 reversed. To maintain second-order accuracy, the Strang splitting[154] can be used, in which the solution $S(t_n)u_0$ is approximated by

$$u^{n+1} \approx S_2(\Delta t/2)S_1(\Delta t)S_2(\Delta t/2)u^n, \tag{1.6.28}$$

or by the one with the roles of S_2 and S_1 reversed. It should be pointed out that

first-order accuracy and second-order accuracy are based on the truncation errors for smooth solutions. For discontinuous solutions, it is not difficult to show that both approximations (1.6.27) and (1.6.28) are at most first-order accurate, see e.g. [35], [159].

`Fourth-order time-splitting method` A fourth-order symplectic time integrator (cf. [172], [99]) for (1.6.25) is as follows:

$$\begin{aligned} &u^{(1)} = e^{2w_1 A\Delta t}\, u^n, \quad u^{(2)} = e^{2w_2 B\Delta t}\, u^{(1)}, \quad u^{(3)} = e^{2w_3 A\Delta t}\, u^{(2)}, \\ &u^{(4)} = e^{2w_4 B\Delta t}\, u^{(3)}, \quad u^{(5)} = e^{2w_3 A\Delta t}\, u^{(4)}, \quad u^{(6)} = e^{2w_2 B\Delta t}\, u^{(5)}, \\ &u^{n+1} = e^{2w_1 A\Delta t}\, u^{(6)}; \end{aligned} \tag{1.6.29}$$

or, equivalently,

$$\begin{aligned} u^{n+1} \approx & S_1(2w_1\Delta t)S_2(2w_2\Delta t)S_1(2w_3\Delta t)S_2(2w_4\Delta t) \\ & S_1(2w_3\Delta t)S_2(2w_2\Delta t)S_1(2w_1\Delta t)u^n, \end{aligned}$$

where

$$\begin{aligned} &w_1 = 0.33780\ 17979\ 89914\ 40851,\ w_2 = 0.67560\ 35959\ 79828\ 81702, \\ &w_3 = -0.08780\ 17979\ 89914\ 40851,\ w_4 = -0.85120\ 71979\ 59657\ 63405. \end{aligned} \tag{1.6.30}$$

Numerical tests

To test the Runge-Kutta algorithms discussed above, we consider Example 5.3.1 in Section 5.3. Let $U = (U_1, \cdots, U_{N-1})^{\mathrm{T}}$, namely the vector of approximation values at the interior Chebyshev points. Using the definition of the differentiation matrix to be provided in the next chapter, the Chebyshev pesudospectral method for the heat equation (1.1.1) with homogeneous boundary condition leads to the system

$$\frac{dU}{dt} = AU,$$

where A is a constant matrix with $(A)_{ij} = (D^2)_{ij}$. The matrix $D^2 = D^1 * D^1$, where D^1 is given by `CODE DM.3` in Sect 2.1. The following pseudo-code implements the RK2 (1.6.6).

```
CODE RK.1
Input N, u_0(x), Δt, Tmax, α
%Form the matrix A
```

```
call CODE DM.3 in Sect 2.1 to get D1(i,j), 0⩽i,j⩽N
D2=D1*D1;
A(i,j)=D2(i,j), 1⩽i,j⩽N-1
Set starting time:  time=0
Set the initial data:  U0=u_0(x)
While time⩽Tmax do
   %Using RK2 (1.6.6)
   U=U0; G=A*U
   U=U+α*Δt*G; G=(-1+2α-2α^2)G+A*U
   U0=U+Δt*G/(2*α)
   Set new time level:  time=time+Δt
endWhile
Output U0(1),U(2), ···, U(N-1)
```

Codes using (1.6.11), i.e., RK4 for autonomous system, can be written in a similar way. Numerical results for Example 5.3.1 using RK2 with $\alpha = 1$ (i.e., the Heun method) and RK4 are given in the following table. `Tmax` in the above code is set to be 0.5. It is seen that these results are more accurate than the forward Euler solutions obtained in Section 5.3.

N	Heun method ($\Delta t=10^{-3}$)	RK4 ($\Delta t=10^{-3}$)
3	1.11e-02	1.11e-02
4	3.75e-03	3.75e-03
6	3.99e-05	4.05e-05
8	1.23e-06	1.77e-06
10	5.92e-07	3.37e-08
11	5.59e-07	1.43e-09
12	5.80e-07	4.32e-10

The numerical errors for $\Delta t = 10^{-3}$, Tmax=0.5 and different values of s (the order of accuracy) can be seen from the following table:

N	s=2	s=3	s=4
3	1.11e-02	1.11e-02	1.11e-02
4	3.75e-03	3.75e-03	3.75e-03
6	3.99e-05	4.05e-05	4.05e-05
8	1.23e-06	1.77e-06	1.77e-06
10	5.92e-07	3.23e-08	3.37e-08
11	5.59e-07	2.82e-09	1.43e-09
12	5.80e-07	1.70e-09	4.32e-10

Exercise 1.6

Problem 1 Solve the problem in Example 5.3.1 by using a pseudo-spectral ap-

proach (i.e. using the differential matrix to solve the problem in the physical space). Take $3 \leqslant N \leqslant 20$, and use RK4.

1.7 Iterative methods and preconditioning

BiCG algorithm
CGS algorithm
BiCGSTAB algorithm
GMRES method
Preconditioning techniques
Preconditioned GMRES

Among the iterative methods developed for solving large sparse problems, we will mainly discuss two methods: the conjugate gradient (CG) method and the generalized minimal residual (GMRES) method. The CG method proposed by Hestenes and Stiefel in 1952 [82] is the *method of choice* for solving large *symmetric positive definite* linear systems, while the GMRES method was proposed by Saad and Schultz in 1986 for solving non-symmetric linear systems [135].

Let the matrix $A \in \mathbb{R}^{n\times n}$ be a symmetric positive definite matrix and $b \in \mathbb{R}^n$ a given vector. It can be verified that $\hat{x}$ is the solution of $Ax = b$ if and only if $\hat{x}$ minimizes the quadratic functional

$$J(x) = \frac{1}{2}x^{\mathrm{T}}Ax - x^{\mathrm{T}}b. \tag{1.7.1}$$

Let us consider the minimization procedure. Suppose x_k has been obtained. Then x_{k+1} can be found by

$$x_{k+1} = x_k + \alpha_k p_k, \tag{1.7.2}$$

where the scalar α_k is called the step size factor and the vector p_k is called the search direction. The coefficient α_k in (1.7.2) is selected such that $J(x_k + \alpha_k p_k) = \min_\alpha J(x_k + \alpha p_k)$. A simple calculation shows that

$$\alpha_k = (r_k, p_k)/(Ap_k, p_k) = p_k^{\mathrm{T}} r_k / p_k^{\mathrm{T}} A p_k.$$

The residual at this step is given by

$$\begin{aligned} r_{k+1} &= b - Ax_{k+1} = b - A(x_k + \alpha_k p_k) \\ &= b - Ax_k - \alpha_k Ap_k = r_k - \alpha_k Ap_k. \end{aligned}$$

Select the *next search direction* p_{k+1} such that $(p_{k+1}, Ap_k) = 0$, i.e,

$$p_{k+1} = r_{k+1} + \beta_k p_k, \tag{1.7.3}$$

where

$$\beta_k = -\frac{(Ap_k, r_{k+1})}{(Ap_k, p_k)} = -\frac{r_{k+1}^{\mathrm{T}} A p_k}{p_k^{\mathrm{T}} A p_k}.$$

It can be verified that

$$r_i^{\mathrm{T}} r_j = 0, \quad p_i^{\mathrm{T}} A p_j = 0, \quad i \neq j. \tag{1.7.4}$$

Consequently, it can be shown that if A is a real $n \times n$ symmetric positive definite matrix, then the iteration converges in at most n steps, i.e. $x_m = \hat{x}$ for some $m \leqslant n$.

The above derivations lead to the following conjugate gradient (CG) algorithm:

Choose x_0, compute $r_0 = b - Ax_0$ and set $p_0 = r_0$.
For $k = 0, 1, \cdots$ *do*
Compute $\alpha_k = (r_k, r_k)/(Ap_k, p_k)$
  Set $x_{k+1} = x_k + \alpha_k p_k$
  Compute $r_{k+1} = r_k - \alpha_k A p_k$
  If $\|r_{k+1}\|_2 \geqslant \epsilon$, continue,
  Compute $\beta_k = (r_{k+1}, r_{k+1})/(r_k, r_k)$
  Set $p_{k+1} = r_{k+1} + \beta_k p_k$
endFor

It is left as an exercise for the reader to prove that these coefficient formulas in the CG algorithm are equivalent to the obvious expressions in the above derivations.

The rate of convergence of the conjugate gradient method is given by the following theorem:

Theorem 1.7.1 *If A is a symmetric positive definite matrix, then the error of the conjugate gradient method satisfies*

$$\|\hat{x} - x_k\|_A \leqslant 2\gamma^k \|\hat{x} - x_0\|_A, \tag{1.7.5}$$

where

$$\|x\|_A = (Ax, x) = x^{\mathrm{T}} A x, \quad \gamma = (\sqrt{\kappa} - 1)/(\sqrt{\kappa} + 1), \tag{1.7.6}$$

and $\kappa = \|A\|_2 \|A^{-1}\|_2$ is the condition number of A. □

For a symmetric positive definite matrix, $\|A\|_2 = \lambda_n$, $\|A^{-1}\|_2 = \lambda_1^{-1}$, where λ_n and λ_1 are the largest and smallest eigenvalues of A. It follows from Theorem 1.7.1

that a 2-norm error bound can be obtained:

$$\|\hat{x} - x_k\|_2 \leqslant 2\sqrt{\kappa}\gamma^k \|x - x_0\|_2. \tag{1.7.7}$$

We remark that

• we only have matrix-vector multiplications in the CG algorithm. In case that the matrix is sparse or has a special structure, these multiplications can be done efficiently.

• unlike the traditional successive over-relaxation (SOR) type method, there is no free parameter to choose in the CG algorithm.

BiCG algorithms

When the matrix A is non-symmetric, an direct extension of the CG algorithm is the so called biconjugate gradient (BiCG) method.

The BiCG method aims to solve $Ax = b$ and $A^{\mathrm{T}}x^* = b^*$ simultaneously. The iterative solutions are updated by

$$x_{j+1} = x_j + \alpha_j p_j, \qquad x^*_{j+1} = x^*_j + \alpha_j p^*_j \tag{1.7.8}$$

and so

$$r_{j+1} = r_j - \alpha_j A p_j, \qquad r^*_{j+1} = r^*_j - \alpha_j A^{\mathrm{T}} p^*_j. \tag{1.7.9}$$

We require that $(r_{j+1}, r^*_j) = 0$ and $(r_j, r^*_{j+1}) = 0$ for all j. This leads to

$$\alpha_j = (r_j, r^*_j)/(Ap_j, p^*_j). \tag{1.7.10}$$

The search directions are updated by

$$p_{j+1} = r_{j+1} + \beta_j p_j, \qquad p^*_{j+1} = r^*_{j+1} + \beta_j p^*_j. \tag{1.7.11}$$

By requiring that $(Ap_{j+1}, p^*_j) = 0$ and $(Ap_j, p^*_{j+1}) = 0$, we obtain

$$\beta_j = (r_{j+1}, r^*_{j+1})/(r_j, r^*_j). \tag{1.7.12}$$

The above derivations lead to the following BiCG algorithm:

Choose x_0, compute $r_0 = b - Ax_0$ and set $p_0 = r_0$.
Choose r^*_0 such that $(r_0, r^*_0) \neq 0$.
For $j = 0, 1, \cdots$ do

Compute $\alpha_j = \frac{(r_j, r_j^*)}{(Ap_j, p_j^*)}$.
Set $x_{j+1} = x_j + \alpha_j p_j$.
Compute $r_{j+1} = r_j - \alpha_j A p_j$ and $r_{j+1}^* = r_j^* - \alpha_j A^{\mathrm{T}} p_j^*$.
If $\|r_{k+1}\|_2 \geqslant \epsilon$, continue,
Compute $\beta_j = \frac{(r_{j+1}, r_{j+1}^*)}{(r_j, r_j^*)}$.
Set $p_{j+1} = r_{j+1} + \beta_j p_j$ and $p_{j+1}^* = r_{j+1}^* + \beta_j p_j^*$
endFor

We remark that

• The BiCG algorithm is particularly suitable for matrices which are positive definite, i.e., $(Ax, x) > 0$ for all $x \neq 0$, but not symmetric.

• the algorithm breaks down if $(Ap_j, p_j^*) = 0$. Otherwise, the amount of work and storage is of the same order as n the CG algorithm.

• if A is symmetric and $r_0^* = r_0$, then the BiCG algorithm reduces to the CG algorithm.

CGS algorithm

The BiCG algorithm requires multiplication by both A and A^{T} at each step. Obviously, this means extra work, and, additionally, it is sometimes cumbersome to multiply by A^{T} than it is to multiply by A. For example, there may be a special formula for the product of A with a given vector when A represents, say, a Jacobian, but a corresponding formula for the product of A^{T} with a given vector may not be available. In other cases, data may be stored on a parallel machine in such a way that multiplication by A is efficient but multiplication by A^{T} involves extra communication between processors. For these reasons it is desirable to have an iterative method that requires multiplication only by A and that generates good approximate solutions. A method that attempts to do this is the conjugate gradient squared (CGS) method.

For the recurrence relations of BiCG algorithms, we see that

$$r_j = \Phi_j^1(A) r_0 + \Phi_j^2(A) p_0,$$

where $\Phi_j^1(A)$ and $\Phi_j^2(A)$ are j-th order polynomials of the matrix A. Choosing $p_0 = r_0$ gives

$$r_j = \Phi_j(A) r_0 \qquad (\Phi_j = \Phi_j^1 + \Phi_j^2),$$

with $\Phi_0 \equiv 1$. Similarly,

$$p_j = \pi_j(A)r_0,$$

where π_j is a polynomial of degree j. As r_j^* and p_j^* are updated, using the same recurrence relation as for r_j and p_j, we have

$$r_j^* = \Phi_j(A^{\mathrm{T}})r_0^*, \qquad p_j^* = \pi_j(A^{\mathrm{T}})r_0^*. \tag{1.7.13}$$

Hence,

$$\alpha_j = \frac{(\Phi_j(A)r_0, \Phi_j(A^{\mathrm{T}})r_0^*)}{(A\pi_j(A)r_0, \pi_j(A^{\mathrm{T}})r_0^*)} = \frac{(\Phi_j^2(A)r_0, r_0^*)}{(A\pi_j^2(A)r_0, r_0^*)}\,. \tag{1.7.14}$$

From the BiCG algorithm:

$$\Phi_{j+1}(t) = \Phi_j(t) - \alpha_j t\pi_j(t), \quad \pi_{j+1}(t) = \Phi_{j+1}(t) + \beta_j\pi_j(t). \tag{1.7.15}$$

Observe that

$$\Phi_j\pi_j = \Phi_j(\Phi_j + \beta_{j-1}\pi_{j-1}) = \Phi_j^2 + \beta_{j-1}\Phi_j\pi_{j-1}. \tag{1.7.16}$$

It follows from the above results that

$$\Phi_{j+1}^2 = \Phi_j^2 - 2\alpha_j t(\Phi_j^2 + \beta_{j-1}\Phi_j\pi_{j-1}) + \alpha_j^2 t^2\pi_j^2,$$

$$\Phi_{j+1}\pi_j = \Phi_j\pi_j - \alpha_j t\pi_j^2 = \Phi_j^2 + \beta_{j-1}\Phi_j\pi_{j-1} - \alpha_j t\pi_j^2,$$

$$\pi_{j+1}^2 = \Phi_{j+1}^2 + 2\beta_j\Phi_{j+1}\pi_j + \beta_j^2\pi_j^2.$$

Define

$$\begin{aligned} r_j &= \Phi_j^2(A)r_0, \quad p_j = \pi_j^2(A)r_0,\\ q_j &= \Phi_{j+1}(A)\pi_j(A)r_0,\\ d_j &= 2r_j + 2\beta_{j-1}q_{j-1} - \alpha_j Ap_j. \end{aligned}$$

It can be verified that

$$\begin{aligned} r_j &= r_{j-1} - \alpha_j Ad_j,\\ q_j &= r_j + \beta_{j-1}q_{j-1} - \alpha_j Ap_j\\ p_{j+1} &= r_{j+1} + 2\beta_j q_j + \beta_j^2 p_j,\\ d_j &= 2r_j + 2\beta_{j-1}q_{j-1} - \alpha_j Ap_j. \end{aligned}$$

Correspondingly,

$$x_{j+1} = x_j + \alpha_j d_j. \tag{1.7.17}$$

This gives the CGS algorithm. It is true that x_j may not be the same as that produced by the BiCG.

The above derivations lead to the following the CGS algorithm:

Choose x_0, compute $r_0 = b - Ax_0$ and set $p_0 = r_0, u_0 = r_0,\ q_0 = 0$.
Choose r_0^* such that $(r_0, r_0^*) \neq 0$.
For $j = 0, 1, \cdots$ do
  Compute $\alpha_j = \frac{(r_j, r_0^*)}{(Ap_j, r_0^*)}$; Compute $q_{j+1} = u_j - \alpha_j A p_j$
  Set $x_{j+1} = x_j + \alpha_j(u_j + q_{j+1})$
  Compute $r_{j+1} = r_j - \alpha_j A(u_j + q_{j+1})$
  If $\|r_{k+1}\|_2 \geqslant \epsilon$, continue,
  Compute $\beta_j = \frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)}$; Compute $u_{j+1} = r_{j+1} + \beta_j q_{j+1}$
  Set $p_{j+1} = u_{j+1} + \beta_j(q_{j+1} + \beta_j p_j)$
endFor

The CGS method requires two matrix-vector multiplications at each step but no multiplications by the transpose. For problems where the BiCG method converges well, the CGS typically requires only about half as many steps and, therefore, half the work of BiCG (assuming that multiplication by A or A^{T} requires the same amount of work). When the norm of the BiCG residual increases at a step, however, that of the CGS residual usually increases by approximately the square of the increase of the BiCG residual norm. The CGS convergence curve may therefore show wild oscillations that can sometimes lead to numerical instabilities.

BiCGSTAB algorithm

To avoid the large oscillations in the CGS convergence curve, one might try to produce a residual of the form

$$r_j = \Psi_j(A)\Phi_j(A)r_0, \tag{1.7.18}$$

where Φ_j is again the BiCG polynomial but Ψ_j is chosen to keep the residual norm small at each step while retaining the rapid overall convergence of the CGS method.

For example, $\Psi_j(t)$ is of the form

$$\Psi_{j+1}(t) = (1 - w_j t)\Psi_j(t). \tag{1.7.19}$$

In the BiCGSTAB algorithm, the solution is updated in such a way that r_j is of the form (1.7.18), where $\Psi_j(A)$ is a polynomial of degree j which satisfies (1.7.19). It can be shown that

$$\begin{aligned} \Psi_{j+1}\Phi_{j+1} &= (1 - w_j t)\Psi_j(\Phi_j - \alpha_j t\pi_j) \\ &= (1 - w_j t)(\Psi_j\Phi_j - \alpha_j t\Psi_j\pi_j), \end{aligned} \tag{1.7.20}$$

$$\begin{aligned} \Psi_j\pi_j &= \Psi_j(\Phi_j + \beta_{j-1}\pi_{j-1}) \\ &= \Psi_j\Phi_j + \beta_{j-1}(1 - w_{j-1}t)\Psi_{j-1}\pi_{j-1}. \end{aligned} \tag{1.7.21}$$

Let $r_j = \Phi_j(A)\Psi_j(A)r_0$ and $p_j = \Psi_j(A)\pi_j(A)r_0$. It can be verified that

$$\begin{aligned} r_{j+1} &= (I - w_j A)(r_j - \alpha_j A p_j), \\ p_{j+1} &= r_{j+1} + \beta_j(I - w_j A)p_j. \end{aligned} \tag{1.7.22}$$

By letting $s_j = r_j - \alpha_j A p_j$, we obtain

$$r_{j+1} = (I - w_j A)s_j. \tag{1.7.23}$$

The parameter w_j is chosen to minimize the 2-norm of r_{j+1}, i.e.,

$$w_j = \frac{(As_j, s_j)}{(As_j, As_j)}. \tag{1.7.24}$$

We also need to find an updating formula for α_j and β_j, only using r_k, p_k and s_k; this is rather complicated and the calculations for deriving them are omitted here.

The BiCGSTAB algorithm is given by

Choose x_0, compute $r_0 = b - Ax_0$ and set $p_0 = r_0$.
Choose r_0^* such that $(r_0, r_0^*) \neq 0$.
For $j = 0, 1, \cdots$ do
Compute $\alpha_j = \frac{(r_j, r_0^*)}{(Ap_j, r_0^*)}$
Set $s_j = r_j - \alpha_j A p_j$; Compute $w_j = \frac{(As_j, s_j)}{(As_j, As_j)}$

Set $x_{j+1} = x_j + \alpha_j p_j + w_j s_j$; $r_{j+1} = s_j - w_j A s_j$
If $\|r_{k+1}\|_2 \geqslant \epsilon$, continue,
Compute $\beta_j = \frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)} \cdot \frac{\alpha_j}{w_j}$
Set $p_{j+1} = r_{j+1} + \beta_j (p_j - w_j A p_j)$
endFor

GMRES method

The GMRES method proposed by Saad and Schultz in 1986 is one of the most important tools for a general *non-symmetric* system

$$Ax = b, \quad \text{with } A \text{ non-symmetric}. \tag{1.7.25}$$

In the k-th iteration of the GMRES method, we need to find a solution of the least-squares problem

$$\min_{x \in x_0 + \|(A, r_0, k)} \|b - Ax\|_2 , \tag{1.7.26}$$

where $r_0 = b - Ax_0$ and $\|(A, r_0, k) := \{r_0, Ar_0, \cdots, A^{k-1} r_0\}$. Let $x \in x_0 + \|(A, r_0, k)$. We have

$$x = x_0 + \sum_{j=0}^{k-1} \gamma_j A^j r_0. \tag{1.7.27}$$

Moreover, it can be shown that

$$r = b - Ax = r_0 - \sum_{j=1}^{k} \gamma_{j-1} A^j r_0. \tag{1.7.28}$$

Like the CG method, the GMRES method will obtain the *exact* solution of $Ax = b$ within n iterations. Moreover, if b is a linear combination of k eigenvectors of A, say $b = \sum_{p=1}^{k} \gamma_p u_{i_p}$, then the GMRES method will terminate in at most k iterations.

Suppose that we have a matrix $V_k = [v_1^k, v_2^k, \cdots, v_k^k]$ whose columns form an orthogonal basis of $\|(A, r_0, k)$. Then any $z \in \|(A, r_0, k)$ can be expressed as

$$z = \sum_{p=1}^{k} u_p v_p^k = V_k u, \tag{1.7.29}$$

where $u \in \mathbb{R}^k$. Thus, once we have found V_k, we can convert the original least-squares problem (1.7.26) into a least-squares problem in $\mathbb{R}^k$, as to be described below. Let x_k be the solution after the k-th iteration. We then have $x_k = x_0 + V_k y_k$, where the vector y_k minimizes

$$\min_{y\in\mathbb{R}^k} \|b - A(x_0 + V_k y)\|_2 = \min_{y\in\mathbb{R}^k} \|r_0 - AV_k y\|_2. \tag{1.7.30}$$

This is a standard linear least-squares problem that can be solved by a QR decomposition.

One can use the modified Gram-Schmidt orthogonalization to find an orthonormal basis of $\|(A, r_0, k)$. The algorithm is given as follows:

Choose x_0, set $r_0 = b - Ax_0, v_1 = r_0/\|r_0\|_2$.
For $i = 1, 2, \cdots, k-1$, do:
  Compute $v_{i+1} = \frac{Av_i - \sum_{j=1}^{i}((Av_i)^{\mathrm{T}} v_j)v_j}{\left\|Av_i - \sum_{j=1}^{i}((Av_i)^{\mathrm{T}} v_j)v_j\right\|_2}$,
endFor

This algorithm produces the columns of the matrix V_k which also form an orthonormal basis for $\|(A, r_0, k)$. Note that the algorithm breaks down when a division by zero occurs.

If the modified Gram-Schmidt process does not break down, we can use it to carry out the GMRES method in the following efficient way. Let $h_{ij} = (Av_j)^{\mathrm{T}} v_i$. By the modified Gram-Schmidt algorithm, we have a $(k+1) \times k$ matrix H_k which is upper Hessenberg, i.e., its entries satisfy $h_{ij} = 0$ if $i > j + 1$. This process produces a sequence of matrices $\{V_k\}$ with orthonormal columns such that $AV_k = V_{k+1}H_k$. Therefore, we have

$$\begin{aligned} r_k &= b - Ax_k = r_0 - A(x_k - x_0) \\ &= \beta V_{k+1}e_1 - AV_k y_k = V_{k+1}(\beta e_1 - H_k y_k), \end{aligned} \tag{1.7.31}$$

where e_1 is the first unit k-vector $(1, 0, \cdots, 0)^{\mathrm{T}}$, and y_k is the solution of

$$\min_{y\in\mathbb{R}^k} \|\beta e_1 - H_k y\|_2. \tag{1.7.32}$$

Hence, $x_k = x_0 + V_k y_k$. To find a minimizer for (1.7.32), we need to look at the

linear algebraic system $\bar{H}_k y = \beta e_1$, namely,

$$\begin{pmatrix} h_{11} & h_{21} & \cdots & h_{k1} \\ h_{12} & h_{22} & \cdots & h_{k2} \\ & h_{23} & \cdots & h_{k3} \\ & & & \vdots \\ & & & h_{kk} \\ & & & h_{k+1,k} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_k \end{pmatrix} = \begin{pmatrix} \beta \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}.$$

This problem can be solved by using rotation matrices to do Gauss-elimination for $\bar{H}_k$ (see e.g. [134]), which yields $\overline{H}_k{}^{(k)} y = \overline{g}_k$, where

$$\overline{H}_k{}^{(k)} = \begin{pmatrix} h_{11}^{(k)} & h_{12}^{(k)} & \cdots & h_{1k}^{(k)} \\ & h_{22}^{(k)} & \cdots & h_{2k}^{(k)} \\ & & \ddots & \vdots \\ & & & h_{kk}^{(k)} \\ & & & h_{k+1,k} \end{pmatrix}, \quad \overline{g}_k = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_k \\ r_{k+1} \end{pmatrix}.$$

Moreover,

$$\min_{y \in \mathbb{R}^k} \|H_k y - \beta e_1\|_2 = \min_{y \in \mathbb{R}^k} \|\overline{H}_k{}^{(k)} y - \overline{g}_k\|_2. \tag{1.7.33}$$

Define $H_k^{(k)}$ to be the matrix containing the first m rows of $\overline{H}_k{}^{(k)}$. It is easy to see that the minimizer of (1.7.33) is the solution of $H_k^{(k)} y_k = \overline{g}_k$.

Below we give the GMRES algorithm for solving $Ax = b$ with A non-symmetric:

Choose x_0, set $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$ and $v_1 = r_0/\beta$.
 For $j = 1, 2, \cdots, k, \cdots,$ do
  Compute $w_j = Av_j$
  for $i = 1, 2, \cdots, j$ do
   Compute $h_{ij} = w_j^{\mathrm{T}} v_i$.
   Set $w_j = w_j - h_{ij} v_i$.
  endfor
  Compute $h_{j+1,j} = \|w_j\|_2$
  Set $v_{j+1} = w_j / h_{j+1,j}$

endFor
Compute $H_k^{(k)}$ and $\overline{g}_k$
Solve $H_k^{(k)} y_k = \overline{g}_k$
Set $x_k = x_0 + V_k y_k$

Preconditioning techniques

It is seen from Theorem 1.7.1 that the rate of convergence of the conjugate gradient method depends on the condition number of A: the larger κ is, the closer γ will be to 1 and the slower will be the rate of convergence. A good preconditioner is a matrix M that is (i) easy to invert, and (ii) the condition number of $M^{-1}A$ is small, or the preconditioned system $M^{-1}Ax = M^{-1}b$ can be solved efficiently by an iterative method. This idea leads to the so-called **preconditioned conjugate gradient** (PCG) method:

Choose x_0, compute $r_0 = b - Ax_0$ and solve $M\tilde{r}_0 = r_0$
Set $p_0 = \tilde{r}_0$
For $k = 0, 1, \cdots$ do
 Compute $\alpha_k = -(\tilde{r}_k, r_k)/(p_k, Ap_k)$
 Set $x_{k+1} = x_k + \alpha_k p_k$; Set $r_{k+1} = r_k - \alpha_k A p_k$
 If $\|r_{k+1}\|_2 \geqslant \epsilon$, continue,
 Solve $M\tilde{r}_{k+1} = r_{k+1}$
 Compute $\beta_k = (\tilde{r}_{k+1}, r_{k+1})/(\tilde{r}_k, r_k)$
 Set $p_{k+1} = \tilde{r}_{k+1} + \beta_k p_k$
endFor

In the above algorithm, we need to solve the system $M\tilde{r} = r$ which may be as complicated as the original system. The idea for reducing the condition number of $M^{-1}A$ is to choose M such that M^{-1} is close to A^{-1}, while the system $M\tilde{r} = r$ is easy to solve. The following theorem describes a way to choose M.

Theorem 1.7.2 *Let A be an $n \times n$ nonsingular matrix and $A = P - Q$ a splitting of A such that P is nonsingular. If $H = P^{-1}Q$ and $\rho(H) < 1$, then*

$$A^{-1} = \left(\sum_{k=0}^{\infty} H^k \right) P^{-1}. \qquad \square$$

Based on this theorem, we can consider the matrices

$$M = P(I + H + \cdots + H^{m-1})^{-1},$$
$$M^{-1} = (I + H + \cdots + H^{m-1})P^{-1}$$

to be approximations of A and A^{-1}, respectively. Thus the solution of the system $M\tilde{r} = r$ becomes

$$\tilde{r} = M^{-1}r = (I + H + \cdots + H^{m-1})P^{-1}r.$$

Equivalently, the solution $\tilde{r} = r_m$ is the the result of applying m steps of the iterative method

$$Pr_{i+1} = Qr_i + r, \;\; i = 0, 1, \cdots, m-1, \;\; r_0 = 0.$$

If $P = D$, $Q = L + U$, the above iteration is the standard Jacobi iteration. Then in the PCG method we replace the system $M\tilde{r}_{k+1} = r_{k+1}$ with `do` m `Jacobi iterations on` $Ar = r_{k+1}$ `to obtain` $\tilde{r}_{k+1}$. The resulting method is called the m-**step Jacobi PCG Method**.

In practice, we may just use the one-step Jacobi PCG Method: in this case $M = D$. Similarly, the symmetric Gauss-Seidel and symmetric successive over-relaxation (SSOR) methods can also be used as preconditioners:

- Symmetric Gauss-Seidel preconditioner:

$$M = (D - L)D^{-1}(D - U), \quad M^{-1} = (D - U)^{-1}D(D - L)^{-1};$$

- SSOR preconditioner:

$$M = \frac{\omega}{2 - \omega}(\omega^{-1}D - L)D^{-1}(\omega^{-1}D - U),$$
$$M^{-1} = (2 - \omega)\omega(D - \omega U)^{-1}D(D - \omega L)^{-1}.$$

Preconditioned GMRES

If we use M as a left preconditioner for the GMRES method, then we are trying to minimize the residual in the space:

$$K_m(A, r_0) = \text{span}\{r_0, M^{-1}Ar_0, \cdots (M^{-1}A)^{m-1}r_0\}. \tag{1.7.34}$$

The resulting algorithm is exactly the same as the original GMRES, except that the matrix A is replaced by $M^{-1}A$.

Below is the preconditioned version of the **GMRES method with left-**

preconditioning:

Compute $r_0 = M^{-1}(b - Ax_0)$ and set $\beta = \|r_0\|_2,\ v_1 = r_0/\beta$.
For $j = 1, 2, \cdots, k, \cdots$ do:
  Compute $w_j = M^{-1}Av_j$.
  for $i = 1, 2, \cdots, j,$ do:
    Compute $h_{ij} = (w_j, v_i)$; Set $w_j = w_j - h_{ij}v_i$
  endfor
  Compute $h_{j+1,j} = \|w_j\|$.
  Set $v_{j+1} = w_j/h_{j+1,j}$.
endFor
Compute $H_k^{(k)}$ and $\overline{g}_k$
Solve $H_k^{(k)} y_k = \overline{g}_k$
Set $x_k = x_0 + V_k y_k$

If M is used as a right preconditioner, we just need to replace A in the original GMRES by AM^{-1}. Also, in the last step, we need to update x_k by

$$x_k = x_0 + M^{-1}V_k y_k. \tag{1.7.35}$$

In practice, for the GMRES method, however, the Gauss-Seidel and SOR methods can also be used as preconditioners:

- Gauss-Seidel preconditioner: $M = D - L,\ M^{-1} = (D - L)^{-1}$;
- SOR preconditioner: $M = \omega^{-1}D - L,\ M^{-1} = \omega(D - \omega L)^{-1}$.

The preconditioned CGS or BiCGSTAB algorithms can be constructed similarly. In general, to use preconditioners for the CGS or BiCGSTAB, we just need to replace the matrix A in the original algorithms by $M^{-1}A$ or AM^{-1}.

Exercise 1.7

Problem 1 Prove (1.7.5) and (1.7.7).

Problem 2 Prove Theorem 1.7.2.

1.8 Error estimates of polynomial approximations

Orthogonal projection in $L^2_{\omega^{\alpha,\beta}}(I)$
Orthogonal projection in $H^1_{0,\omega^{\alpha,\beta}}(I)$
Interpolation error

The numerical analysis of spectral approximations relies on the polynomial approximation results in various norms. In this section, we present some of the basic approximation results for the Jacobi polynomials which include the Legendre and Chebyshev polynomials as special cases. Some basic properties of the Jacobi polynomials are introduced in Section 1.4.

We first introduce some notations. Let $I = (-1, 1)$ and $\omega(x) > 0$ be a weight function (ω is not necessarily in $L^1(I)$). We define the "usual" weighted Sobolev spaces:

$$\begin{aligned} L^2_\omega(I) &= \left\{ u : \int_I u^2 \omega \mathrm{d}x < +\infty \right\}, \\ H^l_\omega(I) &= \left\{ u \in L^2_\omega(I) : \partial_x u, \cdots, \partial^l_x u \in L^2_\omega(I) \right\}, \\ H^l_{0,\omega}(I) &= \left\{ u \in H^l_\omega(I) : u(\pm 1) = \partial_x u(\pm 1) = \cdots = \partial^{l-1}_x u(\pm 1) = 0 \right\}. \end{aligned} \tag{1.8.1}$$

The norms in $L^2_\omega(I)$ and $H^l_\omega(I)$ will be denoted by $\|\cdot\|_\omega$ and $\|\cdot\|_{l,\omega}$, respectively. Furthermore, we shall use $|u|_{l,\omega} = \|\partial^l_x u\|_\omega$ to denote the semi-norm in $H^l_\omega(I)$. When $\omega(x) \equiv 1$, the subscript ω will often be omitted from the notations. Hereafter, we denote the Jacobi weight function of index (α, β) by

$$\omega^{\alpha,\beta}(x) = (1-x)^\alpha (1+x)^\beta.$$

It turns out that the "uniformly" weighted Sobolev spaces in (1.8.1) are not the most appropriate ones to describe the approximation error. Hence, we introduce the following non-uniformly weighted Sobolev spaces:

$$H^m_{\omega^{\alpha,\beta},*}(I) := \left\{ u : \partial^k_x u \in L^2_{\omega^{\alpha+k,\beta+k}}(I), \ \ 0 \leqslant k \leqslant m \right\}, \tag{1.8.2}$$

equipped with the inner product and norm

$$\big(u, v\big)_{m,\omega^{\alpha,\beta},*} = \sum_{k=0}^{m} (\partial^k_x u, \partial^k_x v)_{\omega^{\alpha+k,\beta+k}}, \quad \|u\|_{m,\omega^{\alpha,\beta},*} = \big(u, u\big)^{\frac{1}{2}}_{m,\omega^{\alpha,\beta},*}. \tag{1.8.3}$$

Hereafter, we shall use the expression $A_N \lesssim B_N$ to mean that there exists a positive constant C, independent of N, such that $A_N \leqslant CB_N$.

Orthogonal projection in $L^2_{\omega^{\alpha,\beta}}(I)$

Since $\{J_n^{\alpha,\beta}\}$ forms a complete orthogonal system in $L^2_{\omega^{\alpha,\beta}}(I)$, we can write

$$u(x) = \sum_{n=0}^{\infty} \hat{u}_n^{\alpha,\beta} J_n^{\alpha,\beta}(x), \quad \text{with} \quad \hat{u}_n^{\alpha,\beta} = \frac{(u, J_n^{\alpha,\beta})_{\omega^{\alpha,\beta}}}{\gamma_n^{\alpha,\beta}}, \tag{1.8.4}$$

where $\gamma_n^{\alpha,\beta} = \|J_n^{\alpha,\beta}\|^2_{\omega^{\alpha,\beta}}$. It is clear that

$$P_N = \text{span}\left\{J_0^{\alpha,\beta},\ J_1^{\alpha,\beta},\ \cdots,\ J_N^{\alpha,\beta}\right\}. \tag{1.8.5}$$

We start by establishing some fundamental approximation results on the $L^2_{\omega^{\alpha,\beta}}-$ orthogonal projection $\pi_{N,\omega^{\alpha,\beta}}:\ L^2_{\omega^{\alpha,\beta}}(I) \to P_N$, defined by

$$(\pi_{N,\omega^{\alpha,\beta}} u - u,\ v)_{\omega^{\alpha,\beta}} = 0, \quad \forall v \in P_N. \tag{1.8.6}$$

It is clear that $\pi_{N,\omega^{\alpha,\beta}} u$ is the best $L^2_{\omega^{\alpha,\beta}}-$approximate polynomial of u, and can be expressed as

$$(\pi_{N,\omega^{\alpha,\beta}} u)(x) = \sum_{n=0}^{N} \hat{u}_n^{\alpha,\beta} J_n^{\alpha,\beta}(x). \tag{1.8.7}$$

First of all, we derive inductively from (1.4.7) that

$$\partial_x^k J_n^{\alpha,\beta}(x) = d_{n,k}^{\alpha,\beta} J_{n-k}^{\alpha+k,\beta+k}(x), \quad n \geqslant k, \tag{1.8.8}$$

where

$$d_{n,k}^{\alpha,\beta} = \frac{\Gamma(n+k+\alpha+\beta+1)}{2^k \Gamma(n+\alpha+\beta+1)}. \tag{1.8.9}$$

As an immediate consequence of this formula and the orthogonality (1.4.5), we have

$$\int_{-1}^{1} \partial_x^k J_n^{\alpha,\beta}(x) \partial_x^k J_l^{\alpha,\beta}(x) \omega^{\alpha+k,\beta+k}(x) \mathrm{d}x = h_{n,k}^{\alpha,\beta} \delta_{n,l}, \tag{1.8.10}$$

where

$$h_{n,k}^{\alpha,\beta} = (d_{n,k}^{\alpha,\beta})^2 \gamma_{n-k}^{\alpha+k,\beta+k}. \tag{1.8.11}$$

Let us recall first Stirling's formula,

$$\Gamma(x)=\sqrt{2\pi}x^{x-1/2}e^{-x}\Big\{1+\frac{1}{12x}+\frac{1}{288x^2}+\mathcal{O}(x^{-3})\Big\}. \tag{1.8.12}$$

In particular, we have

$$\Gamma(n+1)=n!\cong\sqrt{2\pi}n^{n+1/2}e^{-n}, \tag{1.8.13}$$

which can be used to obtain the following asymptotic behaviors for $n\gg 1$:

$$\gamma_n^{\alpha,\beta}\sim n^{-1},\quad d_{n,k}^{\alpha,\beta}\sim n^k,\quad h_{n,k}^{\alpha,\beta}\sim n^{2k-1}. \tag{1.8.14}$$

Here, we have adopted the conventional assumption that α,β and k are small constants when compared with large n.

Below is the main result on the Jacobi projection error:

Theorem 1.8.1 *Let $\alpha,\beta>-1$. For any $u\in H^m_{\omega^{\alpha,\beta},*}(I)$ and $m\in\mathbb{N}$,*

$$\|\partial_x^l(\pi_{N,\omega^{\alpha,\beta}}u-u)\|_{\omega^{\alpha+l,\beta+l}}\lesssim N^{l-m}\|\partial_x^m u\|_{\omega^{\alpha+m,\beta+m}},\ \ 0\leqslant l\leqslant m. \tag{1.8.15}$$

Proof Owing to (1.8.10)∼(1.8.11), we have

$$\|\partial_x^k u\|^2_{\omega^{\alpha+k,\beta+k}}=\sum_{n=k}^{\infty}\big(\hat{u}_n^{\alpha,\beta}\big)^2\|\partial_x^k J_n^{\alpha,\beta}\|^2_{\omega^{\alpha+k,\beta+k}}, \tag{1.8.16}$$

$$\begin{aligned}\|\partial_x^l(\pi_{N,\omega^{\alpha,\beta}}u-u)\|^2_{\omega^{\alpha+l,\beta+l}}&=\sum_{n=N+1}^{\infty}\big(\hat{u}_n^{\alpha,\beta}\big)^2\|\partial_x^l J_n^{\alpha,\beta}\|^2_{\omega^{\alpha+l,\beta+l}}\\&=\sum_{n=N+1}^{\infty}\frac{h_{n,l}^{\alpha,\beta}}{h_{n,m}^{\alpha,\beta}}\big(\hat{u}_n^{\alpha,\beta}\big)^2\|\partial_x^m J_n^{\alpha,\beta}\|^2_{\omega^{\alpha+m,\beta+m}}.\end{aligned} \tag{1.8.17}$$

Using the the asymptotic estimate (1.8.14) gives

$$h_{n,l}^{\alpha,\beta}/h_{n,m}^{\alpha,\beta}\lesssim n^{2(l-m)},\quad n\gg 1,\ l,m\in\mathbb{N},$$

which, together with (1.8.17), leads to

$$\begin{aligned}\|\partial_x^l(\pi_{N,\omega^{\alpha,\beta}}u-u)\|^2_{\omega^{\alpha+l,\beta+l}}&\lesssim(N+1)^{2(l-m)}\sum_{n=N+1}^{\infty}\big(\hat{u}_n^{\alpha,\beta}\big)^2\|\partial_x^m J_n^{\alpha,\beta}\|^2_{\omega^{\alpha+m,\beta+m}}\\&\lesssim N^{2(l-m)}\|\partial_x^m u\|^2_{\omega^{\alpha+m,\beta+m}}.\end{aligned}$$

This ends the proof. □

We shall now extend the above result to the cases where α and/or β are negative integers, using the properties of the generalized Jacobi polynomials. We point out that like the classical Jacobi polynomials, the GJPs with negative integer indexes form a complete orthogonal system in $L^2_{\omega^{k,l}}(I)$.

Hence, we define the polynomial space

$$Q_N^{k,l} := \text{span}\{J_{n_0}^{k,l}, J_{n_0+1}^{k,l}, \cdots, J_N^{k,l}\}, \quad k \leqslant -1 \ \text{ and/or } \ l \leqslant -1, \tag{1.8.18}$$

where n_0 is defined in (1.4.8). According to Remark 1.4.1, we have that for $k < -1$ and/or $l \leqslant -1$,

$$Q_N^{k,l} = \{\phi \in P_N \ : \ \partial_x^i \phi(-1) = \partial_x^j \phi(1) = 0, \ \ 0 \leqslant i \leqslant -k-1, \ 0 \leqslant j \leqslant -l-1\}.$$

We now define the orthogonal projection $\pi_{N,\omega^{k,l}} : \ L^2_{\omega^{k,l}}(I) \to Q_N^{k,l}$ by

$$(u - \pi_{N,\omega^{k,l}} u, v_N)_{\omega^{k,l}} = 0, \quad \forall v_N \in Q_N^{k,l}. \tag{1.8.19}$$

Owing to the orthogonality (1.4.10) and the derivative relation (1.4.13), the following theorem is a direct extension of Theorem 1.8.1.

Theorem 1.8.2 *For any $k, l \in \mathbb{Z}$, and $u \in H^m_{\omega^{k,l},*}(I)$,*

$$\|\partial_x^\mu(\pi_{N,\omega^{k,l}} u - u)\|_{\omega^{k+\mu,l+\mu}} \lesssim N^{\mu-m} \|\partial_x^m u\|_{\omega^{k+m,l+m}}, \ \ 0 \leqslant \mu \leqslant m. \tag{1.8.20}$$

Orthogonal projection in $H^1_{0,\omega^{\alpha,\beta}}(I)$

In order to carry out the error analysis of spectral methods for second-order elliptic equations with Dirichlet boundary conditions, we need to study the orthogonal projection error in the space $H^1_{0,\omega^{\alpha,\beta}}(I)$. We define

$$P_N^0 = \{u \in P_N : \ u(\pm 1) = 0\}. \tag{1.8.21}$$

Definition 1.8.1 The orthogonal projector $\pi^{1,0}_{N,\omega^{\alpha,\beta}} : \ H^1_{0,\omega^{\alpha,\beta}}(I) \to P_N^0$ is defined by

$$((u - \pi^{1,0}_{N,\omega^{\alpha,\beta}} u)', v')_{\omega^{\alpha,\beta}} = 0, \quad \forall \ v \in P_N^0. \tag{1.8.22}$$

Theorem 1.8.3 *Let $-1 < \alpha, \beta < 1$. Then for any $u \in H^1_{0,\omega^{\alpha,\beta}}(I) \cap H^m_{\omega^{\alpha-1,\beta-1},*}(I)$,*

$$\|\partial_x(u - \pi^{1,0}_{N,\omega^{\alpha,\beta}} u)\|_{\omega^{\alpha,\beta}} \lesssim N^{1-m} \|\partial_x^m u\|_{\omega^{\alpha+m-1,\beta+m-1}}, \qquad m \geqslant 1.$$

Proof For any $u \in H^1_{0,\omega^{\alpha,\beta}}(I)$, we set

$$u_N = \int_{-1}^{x} \Big\{ \pi_{N-1,\omega^{\alpha,\beta}} u' - \frac{1}{2} \int_{-1}^{1} \pi_{N-1,\omega^{\alpha,\beta}} u' \mathrm{d}\eta \Big\} \mathrm{d}\xi. \tag{1.8.23}$$

Therefore,

$$u_N \in P_N^0 \ \text{ and } u'_N = \pi_{N-1,\omega^{\alpha,\beta}} u' - \frac{1}{2} \int_{-1}^{1} \pi_{N-1,\omega^{\alpha,\beta}} u' \mathrm{d}\eta.$$

Hence,

$$\|u' - u'_N\|_{L^2_{\omega^{\alpha,\beta}}} \leqslant \|u' - \pi_{N-1,\omega^{\alpha,\beta}} u'\|_{L^2_{\omega^{\alpha,\beta}}} + \Big| \frac{1}{2} \int_{-1}^{1} \pi_{N-1,\omega^{\alpha,\beta}} u' \mathrm{d}\eta \Big|. \tag{1.8.24}$$

On the other hand, since $u(\pm 1) = 0$, we derive by using the Cauchy-Schwarz inequality that

$$\begin{aligned} &\Big| \int_{-1}^{1} \pi_{N-1,\omega^{\alpha,\beta}} u' \mathrm{d}x \Big| = \Big| \int_{-1}^{1} (\pi_{N-1,\omega^{\alpha,\beta}} u' - u') \mathrm{d}x \Big| \\ \leqslant &\Big(\int_{-1}^{1} (\omega^{\alpha,\beta})^{-1} \mathrm{d}x \Big)^{\frac{1}{2}} \|\pi_{N-1,\omega^{\alpha,\beta}} u' - u'\|_{L^2_{\omega^{\alpha,\beta}}} \lesssim \|\pi_{N-1,\omega^{\alpha,\beta}} u' - u'\|_{L^2_{\omega^{\alpha,\beta}}}, \end{aligned} \tag{1.8.25}$$

for $\alpha, \beta < 1$. We then conclude from (1.8.24), (1.8.25) and Theorem 1.8.1 that

$$\begin{aligned} \|\partial_x (u - \pi^{1,0}_{N,\omega^{\alpha,\beta}} u)\|_{\omega^{\alpha,\beta}} &= \inf_{\phi_N \in P_N^0} \|u' - \phi'_N\|_{\omega^{\alpha,\beta}} \leqslant \|u' - u'_N\|_{\omega^{\alpha,\beta}} \\ &\lesssim \|u' - \pi_{N-1,\omega^{\alpha,\beta}} u'\|_{\omega^{\alpha,\beta}} \lesssim N^{1-m} \|\partial_x^m u\|_{\omega^{\alpha+m-1,\beta+m-1}}. \end{aligned}$$

This completes the proof of Theorem 1.8.3. □

Interpolation error

We present below an optimal error estimate for the interpolation polynomials based on the Gauss-Lobatto points.

Theorem 1.8.4 *Let $\{x_j\}_{j=0}^N$ be the roots of $(1-x^2)\partial_x J_N^{\alpha,\beta}(x)$ with $-1 < \alpha, \beta < 1$. Let $I_{N,\omega^{\alpha,\beta}} : C[-1,1] \to P_N$ be the interpolation operator with respect to $\{x_j\}_{j=0}^N$. Then, we have*

$$\|\partial_x^l (I_N^{\alpha,\beta} u - u)\|_{\omega^{\alpha+l,\beta+l}} \lesssim N^{l-m} \|\partial_x^m u\|_{\omega^{\alpha+m,\beta+m}}, \ \ 0 \leqslant l \leqslant m. \tag{1.8.26}$$

The proof of the above lemma is rather technical. We refer to [3] for a complete proof (see also [11] for a similar result for the special case $\alpha = \beta$).

Theorem 1.8.4 indicates that error estimates for the interpolation polynomial based on the Gauss-Lobatto points are optimal in suitable weighted Sobolev spaces. One should note that an interpolation polynomial based on uniformly spaced points is usually a very poor approximation unless the function is periodic in the concerned interval.

As we can see from the estimates presented in this section, the convergence rates of spectral projection/interpolation increase with the smoothness of the function, as opposed to a fixed convergence rate for the finite difference or finite element approximations. Moreover, it can be shown that the convergence rates of spectral projection/interpolation are exponential for analytical functions. We now provide a direct proof of this statement in the Chebyshev case.

Let $\{x_j\}$ be the set of Chebyshev-Gauss-Lobatto points, i.e. $x_0 = 1$, $x_N = -1$ and $T'_N(x_j) = 0, 1 \leqslant j \leqslant N-1$. This suggests that

$$T'_N(x) = \alpha_N \prod_{j=1}^{N-1} (x - x_j).$$

Since $T_N(x) = 2^{N-1}\hat{T}_N(x)$, where $\hat{T}_N(x)$ is monic, we have

$$T_N(x) = 2^{N-1}x^N + \text{lower order terms}.$$

Combining the above two equations gives $\alpha_N = N2^{N-1}$. Notice also that $x_0 = 1$ and $x_N = -1$, we obtain

$$\prod_{k=0}^{N} (x - x_k) = \frac{2^{1-N}}{N} (x^2 - 1) T'_N(x).$$

The above result, together with (1.3.6a), yields

$$\Big| \prod_{k=0}^{N} (x - x_k) \Big| \leqslant N2^{1-N}. \tag{1.8.27}$$

Let u be a smooth function in $C^{N+1}(-1, 1)$. Using Lemma 1.2.3, (1.8.27) and Stir-

ling's formula (1.8.13), we obtain

$$\max_{x\in\bar{I}}|u(x)-I_{N,\omega^{\alpha,\beta}}u(x)|\leqslant C\|u^{(N+1)}\|_\infty\left(\frac{e}{2N}\right)^N, \tag{1.8.28}$$

for large N, where C is a constant independent of N. This result implies that if u is smooth, then the interpolations using the Chebyshev-Gauss-Lobatto points may lead to *exponential order* of convergence.

Exercise 1.8

Problem 1 Prove Theorem 1.8.2.

Problem 2 Show that $\pi_{N,\omega^{-1,-1}}=\pi^{1,0}_{N,\omega^{0,0}}$.