Preface

There are a number of texts covering the theory of orthogonal polynomials on the real line and some of their applications (for example, see [4, 11, 37, 43, 46, 52, 60, 61]). A comprehensive account of computational methods for generating orthogonal polynomials numerically has appeared only relatively recently in [24]. All the techniques discussed in [24], including tests and the generation of figures and tables, are supported by MATLAB programs which are collected in the package `OPQ` (*Orthogonal Polynomials and Quadrature*). Some of the more important `OPQ` routines have since been transcribed into symbolic MATLAB programs, collected in the package `SOPQ` (*Symbolic Orthogonal Polynomials and Quadrature*), which can be run in variable-precision arithmetic. For the content of the two packages, see Appendices A and B, respectively, and for their internet location the beginning of Chapter 1.

The use of the `OPQ` routines is illustrated in the article [27] in four sections entitled Orthogonal polynomials; Sobolev orthogonal polynomials; Quadrature; and Approximation. Appended to each of these sections is a collection of exercises, some fairly routine, others more advanced and not without practical interest. The present text takes up these exercises, slightly edited to make them self-contained, and supplements them with many new exercises. Detailed solutions are provided to all of them, complete with relevant MATLAB software. All pieces of software used in this book can be downloaded from

`www.siam.org/books/se26.`

High-precision computation is applied when appropriate, using the package `SOPQ`.

Most of the exercises involve polynomials orthogonal relative to an absolutely continuous measure,

$$\mathrm{d}\lambda(t) = w(t)\mathrm{d}t,$$

involving a weight function w supported on a finite or infinite interval $[a, b]$, $-\infty \le a < b \le \infty$, or on a finite number of such intervals. We generally assume that $w \ge 0$ on $[a, b]$ and that all moments of w,

$$\mu_r = \int_{\mathbb{R}} t^r w(t)\mathrm{d}t, \quad r = 0, 1, 2, \ldots, \quad \mu_0 > 0, \tag{0.1}$$

exist, or at least those that are relevant. For simplicity of notation, we integrate over the real line $\mathbb{R}$, it being understood that outside the support of w the integrand is identically zero. Most of the time we use *monic* orthogonal polynomials (having leading coefficient 1) and denote them by $\pi_k(\cdot\,;\mathrm{d}\lambda)$, that is,

$$\begin{aligned} &\pi_k(t;\mathrm{d}\lambda) = t^k + \text{terms of lower degree},\\ &\int_{\mathbb{R}} \pi_k(t;\mathrm{d}\lambda)\pi_\ell(t;\mathrm{d}\lambda)\mathrm{d}\lambda(t) = 0, \quad k \ne \ell. \end{aligned} \tag{0.2}$$

Occasionally, we also consider *discrete* orthogonal polynomials, which are orthogonal relative to a discrete measure,

$$\mathrm{d}\lambda_N(x) = \sum_{k=1}^{N} w_k \delta(x - x_k)\mathrm{d}x, \quad w_k > 0, \tag{0.3}$$

where δ is the Dirac delta function and x_k mutually distinct numbers (usually ordered increasingly). There exist exactly N orthogonal polynomials in this case,

$$\sum_{k=1}^{N} w_k \pi_m(x_k; \mathrm{d}\lambda_N)\pi_n(x_k; \mathrm{d}\lambda_N) = 0, \quad m \neq n, \tag{0.4}$$

where $m, n = 0, 1, 2, \ldots, N-1$.

Every system $\{\pi_k(\cdot; \mathrm{d}\lambda)\}$ of monic orthogonal polynomials is known to satisfy a *three-term recurrence relation*

$$\begin{aligned} &\pi_{k+1}(t) = (t - \alpha_k)\pi_k(t) - \beta_k \pi_{k-1}(t), \quad k = 0, 1, 2, \ldots, \\ &\pi_0(t) = 1, \quad \pi_{-1}(t) = 0, \end{aligned} \tag{0.5}$$

where $\alpha_k = \alpha_k(\mathrm{d}\lambda)$ are real, and $\beta_k = \beta_k(\mathrm{d}\lambda)$ positive coefficients, both depending on $\mathrm{d}\lambda$. These are of crucial importance for the constructive theory of orthogonal polynomials. Once known, they provide access to Gaussian quadrature and to other important approximation-theoretic processes involving, for example, continued fractions. Unless they are known explicitly, the central theme in constructive orthogonal polynomials, therefore, is methods for computing these coefficients numerically, given the measure $\mathrm{d}\lambda$. There are two such methods generally applicable. The first is the classical *Chebyshev algorithm*, or the more recent *modified Chebyshev algorithm*, that takes the first $2n$ moments, resp., modified moments, of the measure and produces from them the first n recurrence coefficient α_k, β_k, $k = 0, 1, 2, \ldots, n-1$. Since moments are often explicitly known, the classical Chebyshev algorithm is a convenient way to generate the desired coefficients. Because of severe ill-conditioning, however, the use of high-precision arithmetic is imperative. The second method is based on an appropriate *discretization* of the measure and takes the corresponding discrete orthogonal polynomials to approximate the desired ones.

Another circle of problems, also of practical importance, has to do with modifications of measures. If one knows the recurrence coefficients of the measure $\mathrm{d}\lambda$, the problem is to find the recurrence coefficients of a modified measure, namely the measure $\mathrm{d}\lambda$ multiplied by a given rational function. Algorithms that solve such problems are referred to as *modification algorithms*. When divisors are involved, one faces other instances where high-precision work is essential.

The text has five chapters, the first giving a brief introduction to the software packages `OPQ` and `SOPQ`, and the remaining four being organized according to the four subject matters of [27] subdivided into topical subsections. Each of these chapters will start out with a brief summary of notation and content and is followed by the exercises, each being solved immediately after it has been formulated. Those marked by an asterisk are more advanced. The exercises taken from [27] are so identified. Equations are numbered consecutively in each chapter, Eq. (i, k) being the kth numbered equation in Chapter i, and so are the exercises, examples, and demos.

The majority of the MATLAB scripts in this book were run with MATLAB Release R2011a; a few others used Releases R2012b, R2014a, and R2015a. In general, it does not

matter all that much what release is being used. There are, however, a few instances where it does: The scripts in Exs. 2.34(c) and 5.1(d) work properly in Release R2011a, but not in Release R2014a, because of a problem with the MATLAB routine `eig.m`, which in the latter cases produces eigenvectors of insufficient accuracy (see also Demo 1.17). In Ex. 5.2(b), the MATLAB function `erfc(z)` accepts symbolic arguments `z` in MATLAB Release R2014a, but not in the earlier Release R2011a.

In variable-precision work, a basic tool is loading multiprecision arrays from a source text file into the MATLAB working window. Since the MATLAB Symbolic Toolbox does not currently provide such a loading routine, one named `loadvpa.m` (cf. Appendix B.3) has been developed in collaboration with MathWorks, Inc. It is shown below.

```
function a=loadvpa(filename,nheader_lines,dig,n,m)
%LOADVPA Loading a multiprecision source file.
fid=fopen(filename,'r');
%Skip first nheader_lines of header
for i =  1:nheader_lines
fgetl(fid);
end
C = textscan(fid,'%s');
C = C{:};
fclose(fid);
for k=1:numel(C)
  a(k)=vpa(C{k},dig);
end
a=reshape(a,m,n)';
```

For example, the commands (cf. answer to Ex. 5.1(c))

```
ab32=loadvpa('ab_hrhermite.txt',3,32,200,2);
ab=vpa(ab32(1:5,:),25)
```

display the first five recurrence coefficients to 25 decimal digits

```
ab =

[ 0.56418958354775628694807 95, 0.88622692545275801364908 37]
[ 0.98842539284680028548706 34, 0.18169011381620932846223 25]
[  1.28596761936393996028278 9, 0.34132512895943919856417 18]
[  1.52472084408011530351300 2, 0.50496215298800163193575 12]
[  1.73019227430943925677156 1, 0.67026419463961908567850 84]
```

of the half-range Hermite measure $\mathrm{d}\lambda(t) = e^{-t^2}\mathrm{d}t$ on $\mathbb{R}_+$ as type `sym` by loading the file `ab_hrhermite.txt` containing three comment lines followed by the first 200 recurrence coefficients for $\mathrm{d}\lambda$, accurate to 32 digits. In contrast, the MATLAB commands

```
load -ascii ab_hrhermite.txt;
ab_hrhermite
```

load and display the file `ab_hrhermite.txt` as type `double` to double-precision accuracy.