Numerical Plank integration

[*]Report 2 on the course "Numerical Analysis".

1st Chen Yihang
Peking University
1700010780

Abstract

In this report, we use Gauss-Laguerre quadrature to calculate Plank integral on the interval $(0, +\infty)$. In addition, we truncate the interval into a finite one $(0, L)$, and use composite (midpoint, trapezoidal, Simpson) quadrature and Romberg quadrature to calculate the integral on the finite interval.

In the Gauss quadrature, despite the fact that we introduce the Golub-Welsch algorithm in Appendix B, we do not adopt the algorithm to compute nodes in the programmes. However, we derive the applicable representations of weights in Appendix A and use it in our programmes.

CONTENTS

I. PROBLEM STATEMENT

In this report, we are going to numerically calculate

$$\int_0^{+\infty} \frac{x^3}{e^x - 1} \mathrm{d}x \tag{I.1}$$

As a matter of fact,

$$\begin{aligned} &\int_0^{+\infty} \frac{x^3}{e^x - 1} \mathrm{d}x = \int_0^{+\infty} \frac{x^3 e^{-x}}{1 - e^{-x}} \mathrm{d}x = \int_0^{+\infty} \sum_{n=1}^{\infty} x^3 e^{-nx} \mathrm{d}x \\ &= \sum_{n=1}^{\infty} \int_0^{+\infty} x^3 e^{-nx} \mathrm{d}x = \sum_{n=1}^{\infty} \frac{6}{n^4} = 6\zeta(4) = \frac{\pi^4}{15} \end{aligned} \tag{I.2}$$

II. Numerical Integration

A. Infinite interval

1) Gauss-Laguerre integration: Since

$$\int_0^{+\infty} \frac{x^3}{e^x-1}\mathrm{d}x = \int_0^{+\infty} \frac{x^3 e^{-x}}{1-e^{-x}}\mathrm{d}x = \int_0^{+\infty} e^{-x} g(x)\mathrm{d}x \tag{II.1}$$

where $g(x) = \frac{x^3}{1-e^{-x}}$. It is most natural to adopt Gauss-Laguerre formula.

$$\int_0^{+\infty} e^{-x} g(x)\mathrm{d}x \approx \sum_{j=1}^{n} A_j g(x_j) \tag{II.2}$$

where x_j $(j = 1, 2, \cdots, n)$ is the root of the n-th Laguerre polynomial. Besides, if we denote $l_j(x)$ as the basis of Lagrange interpolation, A_j can be represented As

$$A_j = \int_0^{\infty} l_j(x) e^{-x}\mathrm{d}x \tag{II.3}$$

The Laguerre polynomials are given by the sum

$$L_n(x) = \sum_{k=0}^{n} \frac{(-1)^k}{k!}\binom{n}{k} x^k. \tag{II.4a}$$

$$L_n'(x) = \sum_{k=0}^{n-1} \frac{(-1)^{k+1}}{k!}\binom{n}{k+1} x^k \tag{II.4b}$$

In practice, we can recursively calculate the results by

$$\begin{aligned} (n+1)L_{n+1}(x) &= (2n+1-x)L_n(x) - nL_{n-1}(x) \\ xL_n'(x) &= n(L_n(x) - L_{n-1}(x)) \end{aligned} \tag{II.5}$$

Hence,

$$\begin{aligned} x_j L_n'(x_j) &= (n+1)L_{n+1}(x_j) \\ (n+1)L_{n+1}(x_j) &= -nL_{n-1}(x_j) \end{aligned} \tag{II.6}$$

A standard result shows that the error can be bound by

$$E(f) = \frac{(n!)^2}{(2n)!} f^{(2n)}(\xi) \tag{II.7}$$

Combined II.6 with A.2 we have

$$A_j = \frac{x_j}{(n+1)^2 (L_{n+1}(x_j))^2} \tag{II.8}$$

B. Finite interval

We truncate the interval $(0, \infty)$ into $(0, L)$. In this subsection, $f(x) = \frac{x^3 e^{-x}}{1-e^{-x}}$. By the L'Hospital's rule, we can naturally define $f(0) = 0$. Then, we perform the numerical integration on the interval $[0, L]$. The grid point of the quadrature is $0 = x_0 < x_1 < \cdots < x_n = L$. In the following, we define $M_k = \max |f^{(k)}|$.

1) Newton-Cotes quadrature: The estimate of the integral will be given by

$$\int_0^L f(x)dx = L\sum_{i=0}^{n} w_i f(x_i) \tag{II.9}$$

Assume V is the Vandermonde matrix, whose entry is $V_{ij} = x_{i-1}^{j-1}$. Then by setting $f = x^i, 1 \leq i \leq n$, we have

$$w = (M^\top)^{-1} b \tag{II.10}$$

where $b_i = \frac{n^i}{i+1}$.

In implement the algorithm, scipy package uses the Newton-Raphson iteration since the Vandermonde matrix is ill-conditioned. The following codes are excerpted from function "newton_cotes" in the scipy package.[1]

```
def newton_cotes(rn, equal=0):
    '''
    Parameters
    ----------
    rn : int
        The integer order for equally-spaced data or the relative
        positions of the samples with the first sample at 0 and
        the last at N, where N+1 is the length of 'rn'.
        N is the order of the Newton-Cotes integration.
    equal : int, optional
        Set to 1 to enforce equally spaced data.
    Returns
    -------
    an : ndarray
        1-D array of weights to apply to the function at the provided
        sample positions.

    '''
    yi = rn / float(N)
    ti = 2 * yi - 1
    nvec = np.arange(N+1)
    C = ti ** nvec[:, np.newaxis]
    Cinv = np.linalg.inv(C)
    # improve precision of result
    for i in range(2):
        Cinv = 2*Cinv - Cinv.dot(C).dot(Cinv)
    vec = 2.0 / (nvec[::2]+1)
    ai = Cinv[:, ::2].dot(vec) * (N / 2.)
```

We modify this part of codes in our implementation. We find that it is generally not efficient to use high order Newton_Cotes quadrature.

2) Composite quadrature:

[1]https://github.com/scipy/scipy/blob/v0.18.0/scipy/integrate/quadrature.py#L833-L842

a) Composite midpoint quadrature:

$$\int_0^L f(x)\mathrm{d}x \approx h\sum_{i=0}^{n-1} f(x_{i+\frac{1}{2}}) := M(h) \tag{II.11}$$

whose truncation error is

$$|\int_0^L f(x)\mathrm{d}x - M(h)| \leq \frac{h^2}{24}M_2 L \tag{II.12}$$

b) Composite trapezoidal quadrature:

$$\int_0^L f(x)\mathrm{d}x \approx \frac{h}{2}\sum_{i=0}^{n-1}[f(x_i) + f(x_{i+1})] := T(h) \tag{II.13}$$

whose truncation error is

$$|\int_0^L f(x)\mathrm{d}x - T(h)| \leq \frac{h^2}{12}M_2 L \tag{II.14}$$

c) Composite Simpson quadrature:

$$\int_0^L f(x)\mathrm{d}x \approx \frac{h}{6}\sum_{i=0}^{n-1}[f(x_i) + f(x_{i+\frac{1}{2}}) + f(x_{i+1})] := S(h) \tag{II.15}$$

whose truncation error is

$$|\int_0^L f(x)\mathrm{d}x - S(h)| \leq \frac{h^4}{2880}M_4 L \tag{II.16}$$

d) Estimation of M_2 and M_4: We provide an estimate of M_2 and M_4 for future error estimation. We have (by Wolfram Alpha[2])

$$f''(x) = \frac{x(6 + e^x(-12 + x(6 + x) + e^x(6 + (-6 + x)x)))}{(-1 + e^x)^3} \tag{II.17}$$

$$\begin{aligned}
f^{(4)}(x) &= (-4e^{2x}(2x^3 + (x^3 + 18x - 12)\cosh(x) \\
&+ (6 - 9x^2)\sinh(x) - 18x + 9) \\
&- 2e^{2x}((x^3 + 18x - 12)\sinh(x) + 6x^2 + (6 - 9x^2)\cosh(x) \\
&+ (3x^2 + 18)\cosh(x) - 18x\sinh(x) - 18))/(-1 + e^x)^4 \\
&- (4e^x(-2e^{2x}(2x^3 + (x^3 + 18x - 12)cosh(x) \\
&+ (6 - 9x^2)sinh(x) - 18x + 9) - 6))/(-1 + e^x)^5
\end{aligned} \tag{II.18}$$

Since $\lim_{x\to 0} f''(x) = 2$, and by plotting the figure, we can make $M_2 = 2$. Similarly, by plotting $f^{(4)}(x)$, we can make $M_4 = 2$.

3) Romberg quadrature: We define $T_1(h)$ is the composite trapezoidal quadrature with grid size h. Recursively, $T_k(h)$ can be written as

$$T_{k+1}(h) = \frac{T_k(h/2) - 4^{-k}T_k(h)}{1 - 4^{-k}}, \quad k \geq 1 \tag{II.19}$$

and the following error estimation can be derived

$$\int_0^L f(x)\mathrm{d}x - T_{k+1}(h) = \mathcal{O}(h^{2(k+1)}), \quad k \geq 1 \tag{II.20}$$

To implement the algorithm, we encode $T_k(h/2^j)$ into a subdiagnoal matrix, and use relation II.19 to calculate its item recursively.

[2]https://www.wolframalpha.com/

4) Gauss-Legendre quadrature: We change the integration interval from $(0, L)$ to $(-1, +1)$ by

$$\begin{aligned}\int_0^L f(x)dx &= \frac{L}{2}\int_{-1}^{1} f(\frac{L\xi + L}{2})d\xi \\ &= \frac{L}{2}\sum_{i=1}^{n} w_i f\left(\frac{Lx_i + L}{2}\right)\end{aligned} \tag{II.21}$$

where x_i are the quadrature nodes and w_i are corrosponding weights.

By definition, Legendre polynomials can be represented by

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2 - 1)^n \tag{II.22}$$

which can be explicitly represented as

$$P_n(x) = \frac{1}{2^n}\sum_{k=0}^{\left[\frac{n}{2}\right]}\binom{n}{k}\binom{2n-2k}{n}x^{n-2k} \tag{II.23}$$

The recursion formula is

$$\begin{aligned}(n+1)P_{n+1}(x) &= (2n+1)xP_n(x) - nP_{n-1}(x) \\ (1-x^2)P_n'(x) &= (n+1)xP_n(x) - (n+1)P_{n+1}(x)\end{aligned} \tag{II.24}$$

which can be proved by induction.

Combine II.24 with A.2, we have

$$w_j = \frac{2(1-x_j^2)}{(n+1)^2[P_{n+1}(x_j)]^2} \tag{II.25}$$

A standard estimation of error shows that

$$E(f) = \frac{2^{2n+1}(n!)^4 L}{2(2n+1)[(2n)!]^3}f^{(2n)}(\xi) \tag{II.26}$$

III. Results

A. List of program files

1) In the file "Infinite_integration.py", we implement the Gauss-Laguerre quadrature.
   a) **Laguerre**: return n-th Laguerre polynomial's value and derivative at a given point.
   b) **Laguerre_coeff**: return quadrature coefficients.
   c) **Laguerre_zeros**: return zeros of Laguerre polynomial.
   d) **Gauss_Laguerre**: Gauss-Laguerre quadrature.
2) In the file "finite_integration.py", we implement quadrature on a finite interval.
   a) **midpoint, simposn**: implement basic corrosponding quadrature.
   b) **composite**: implements composite midpoint, trapezoidal, and Simpson quadrature.
   c) **Romberg**: implements Romberg acceleration method.
   d) **Legendre**: return n-th Legendre polynomial's value and derivative at a given point.
   e) **Legendre_coeff**: return quadrature coefficients.
   f) **Legendre_zeros**: return zeros of Legendre polynomial.
   g) **Gauss_Legendre**: Gauss-Legendre quadrature.
   h) **newton_cotes_coeff**: return quadrature coefficients.
   i) **newton_cotes**: Newton-Cotes quadrature.

To execute the Python scripts, the following packages are required:
numpy, matplotlib, tikzplotlib
where the last two packages are adopted to plot the results.

B. *Infinite interval*

We plot the error by Gauss-Laguerre quadrature under $n = 1, 2, \cdots, 100$ below in Figure III-B.

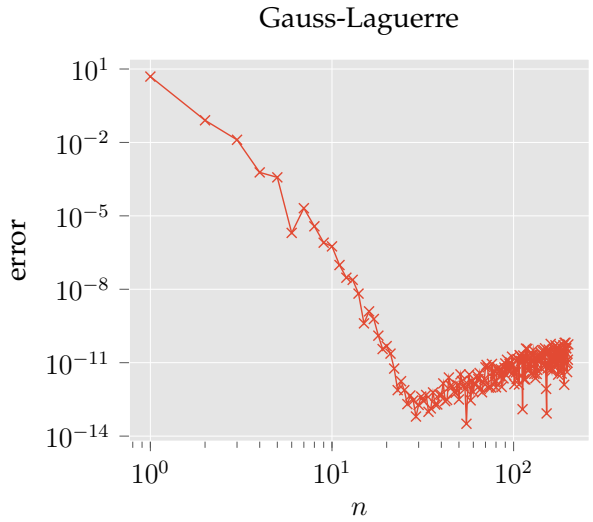

Figure III.1. Gauss-Laguerre quadrature

Since we cannot estimate $f^{(2n)}$ accurately, it is hard to use II.7 to obtain theoretical bound.

C. *Finite interval*

For the composite quadrature, the stopping criterion is $|T(h) - T(h/2)| \leq 10^{-10}$, and initially, we use 10 grid point. We list the results in Table III.1. We find that $L = 40 \sim 80$ is an appropriate choice. Smaller L makes the $\int_L^\infty$ part innegligible, larger L render grid points on the large value useless, which leads to more grid points, compare figure III.3 and III.4 for example.

Setting $L = 50$, for the Romberg quadrature, assume $n = 2^0, 2^1, 2^2, \cdots, 2^{19}$, we plot the results in Figure III.2. As before, the increase in the estimation error is induced by numerical error, and the convergence rate can be clearly observed when $\log_2(n)$ is in $5 \sim 10$.

We take $L = 40$, and calculate our theoretical error bound. For midpoint, trapezoidal, and Simpson rule, we have

$$\begin{aligned} e_1 &= \frac{2}{24} L(L/10/2^9)^2 = 2 \times 10^{-4} \\ e_2 &= \frac{2}{12} L(L/10/2^9)^2 = 4 \times 10^{-4} \\ e_3 &= \frac{2}{2880} L(L/10/2^9)^4 = 1 \times 10^{-10} \end{aligned} \tag{III.1}$$

However, despite major difference between theoretical bound, where $e_3 \ll e_1, e_2$, the numerical error is quite similar ($\sim 10^{-13}$) for these three composite rule.

If we directly apply Newton_Cotes quadrature on the interval, the performace is generally very poor due to numerical instability. We still take $L = 50$, and the error is presented in III.2. Since composite trapezoidal quadrature is accurate enough, there is no need to apply higher order composite rule.

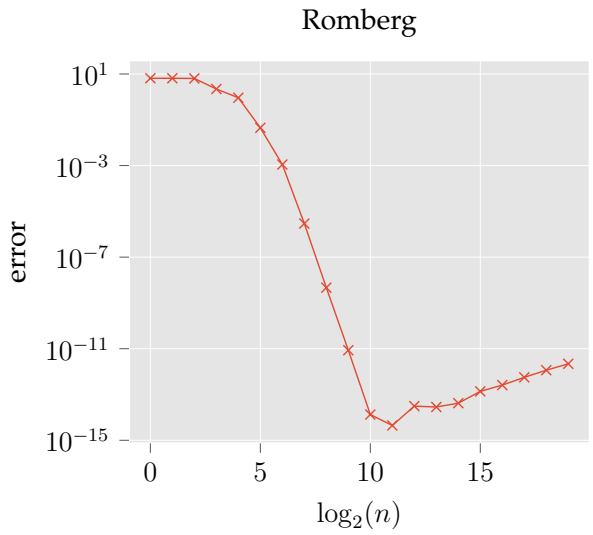

Figure III.2. Romberg quadrature

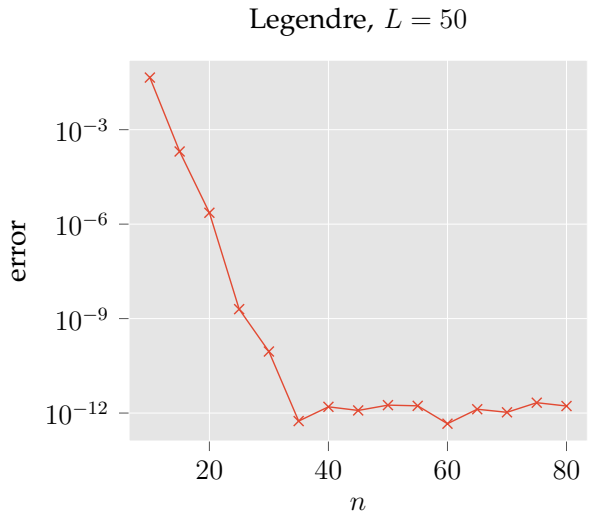

Figure III.3. Legendre quadrature, $L = 50$

Appendix A
Gaussian quadrature weights

Let p_n be a nontrivial polynomial of degree n such that

$$\int_a^b \omega(x)\, x^k p_n(x)\, dx = 0, \quad \text{for all } k = 0, 1, \ldots, n-1. \tag{A.1}$$

The weights can be expressed as

$$w_i = \frac{a_n}{a_{n-1}} \frac{\int_a^b \omega(x) p_{n-1}(x)^2\, dx}{p_n'(x_i) p_{n-1}(x_i)} \tag{A.2}$$

where a_k is the coefficient of x^k in $p_k(x)$. To prove this, note that using Lagrange interpolation one can express $r(x)$ in terms of $r(x_i)$ as

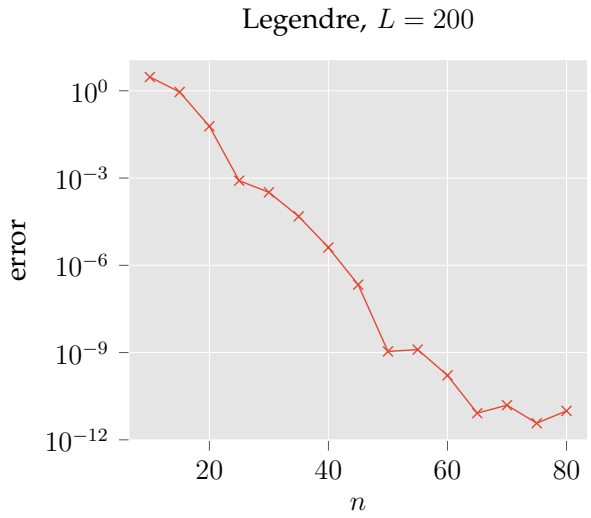

Figure III.4. Legendre quadrature, $L = 200$

Method \ Results	L	error	iteration
Midpoint	10	0.0620	12
	20	1.92×10^{-5}	8
	40	5.09×10^{-13}	9
	80	7.24×10^{-13}	10
	160	7.24×10^{-13}	11
Trapezoidal	10	0.0620	13
	20	1.92×10^{-5}	8
	40	1.24×10^{-12}	9
	80	1.00×10^{-12}	10
	160	1.00×10^{-12}	11
Simpson	10	0.0620	12
	20	1.92×10^{-5}	8
	40	5.08×10^{-13}	9
	80	7.23×10^{-13}	10
	160	7.23×10^{-13}	11

Table III.1
COMPOSITE QUADRATURE

N	5	10	20	30	40	50
error	5.90	0.87	0.20	0.02	2898.16	79908.88

Table III.2
DIRECT NEWTON COTES QUADRATURE

$$r(x) = \sum_{i=1}^{n} r(x_i) \prod_{\substack{1 \le j \le n \\ j \neq i}} \frac{x - x_j}{x_i - x_j} \tag{A.3}$$

because $r(x)$ has degree less than n and is thus fixed by the values it attains at n different points. Multiplying both sides by $r(x_i)$ and integrating from a to b yields

$$\int_a^b \omega(x) r(x) dx = \sum_{i=1}^{n} r(x_i) \int_a^b \omega(x) \prod_{\substack{1 \le j \le n \\ j \neq i}} \frac{x - x_j}{x_i - x_j} dx \tag{A.4}$$

The weights w_i are thus given by

$$w_i = \int_a^b \omega(x) \prod_{\substack{1 \le j \le n \\ j \neq i}} \frac{x - x_j}{x_i - x_j} dx \tag{A.5}$$

This integral expression for w_i can be expressed in terms of the orthogonal polynomials $p_n(x)$ and $p_{n-1}(x)$ as follows.

We can write

$$\prod_{\substack{1 \le j \le n \\ j \neq i}} (x - x_j) = \frac{\prod_{1 \le j \le n} (x - x_j)}{x - x_i} = \frac{p_n(x)}{a_n (x - x_i)} \tag{A.6}$$

where a_n is the coefficient of x^n in $p_n(x)$. Taking the limit of x to x_i yields using L'Hôpital's rule

$$\prod_{\substack{1 \le j \le n \\ j \neq i}} (x_i - x_j) = \frac{p_n'(x_i)}{a_n} \tag{A.7}$$

We can thus write the integral expression for the weights as

$$w_i = \frac{1}{p_n'(x_i)} \int_a^b \omega(x) \frac{p_n(x)}{x - x_i} dx \tag{A.8}$$

In the integrand, writing

$$\frac{1}{x - x_i} = \frac{1 - \left(\frac{x}{x_i}\right)^k}{x - x_i} + \left(\frac{x}{x_i}\right)^k \frac{1}{x - x_i} \tag{A.9}$$

yields

$$\int_a^b \omega(x) \frac{x^k p_n(x)}{x - x_i} dx = x_i^k \int_a^b \omega(x) \frac{p_n(x)}{x - x_i} dx \tag{A.10}$$

provided $k \le n$, because $\frac{1 - \left(\frac{x}{x_i}\right)^k}{x - x_i}$ is a polynomial of degree $k - 1$ which is then orthogonal to $p_n(x)$. So, if $q(x)$ is a polynomial of at most n-th degree we have

$$\int_a^b \omega(x) \frac{p_n(x)}{x - x_i} dx = \frac{1}{q(x_i)} \int_a^b \omega(x) \frac{q(x) p_n(x)}{x - x_i} dx \tag{A.11}$$

We can evaluate the integral on the right hand side for $q(x) = p_{n-1}(x)$ as follows. Because $\frac{p_n(x)}{x-x_i}$ is a polynomial of degree $n-1$, we have

$$\frac{p_n(x)}{x - x_i} = a_n x^{n-1} + s(x) \tag{A.12}$$

where $s(x)$ is a polynomial of degree $n-2$. Since $s(x)$ is orthogonal to $p_{n-1}(x)$ we have

$$\int_a^b \omega(x) \frac{p_n(x)}{x - x_i} dx = \frac{a_n}{p_{n-1}(x_i)} \int_a^b \omega(x) p_{n-1}(x) x^{n-1} dx \tag{A.13}$$

We can then write

$$x^{n-1} = \left(x^{n-1} - \frac{p_{n-1}(x)}{a_{n-1}} \right) + \frac{p_{n-1}(x)}{a_{n-1}} \tag{A.14}$$

The term in the brackets is a polynomial of degree $n-2$, which is therefore orthogonal to $p_{n-1}(x)$. The integral can thus be written as

$$\int_a^b \omega(x) \frac{p_n(x)}{x - x_i} dx = \frac{a_n}{a_{n-1} p_{n-1}(x_i)} \int_a^b \omega(x) p_{n-1}(x)^2 dx \tag{A.15}$$

According to equation A.5, the weights are obtained by dividing this by $p'_n(x_i)$ and that yields the expression in equation A.2.

Appendix B
The Golub-Welsch algorithm

Orthogonal polynomials with leading coefficient one satisfy the recurrence relation

$$p_{r+1}(x) = (x - a_r) p_r(x) - b_r p_{r-1}(x) \tag{B.1}$$

Writing B.1 into matrix form, we have that $\mathbf{J}\tilde{P} = x\tilde{P} - p_n(x)\mathbf{e}_n$, where $\tilde{P} = [p_0(x), \cdots, p_{n-1}(x)]^\top$, $\mathbf{e}_n = [0, 0, \cdots, 1]^\top$, and

$$\mathbf{J} = \begin{pmatrix} a_0 & 1 & 0 & \dots & \dots & \dots \\ b_1 & a_1 & 1 & 0 & \dots & \dots \\ 0 & b_2 & a_2 & 1 & 0 & \dots \\ 0 & \dots & \dots & \dots & \dots & 0 \\ \dots & \dots & 0 & b_{n-2} & a_{n-2} & 1 \\ \dots & \dots & \dots & 0 & b_{n-1} & a_{n-1} \end{pmatrix} \tag{B.2}$$

The zeros x_j of the polynomials up to degree n, which are used as nodes for the Gaussian quadrature can be found by computing the eigenvalues of this tridiagonal matrix. However, for computing the weights and nodes, it is preferable to consider the symmetric tridiagonal matrix $\mathcal{J}$ defined as

$$\mathcal{J} = \begin{pmatrix} a_0 & \sqrt{b_1} & 0 & \dots & \dots & \dots \\ \sqrt{b_1} & a_1 & \sqrt{b_2} & 0 & \dots & \dots \\ 0 & \sqrt{b_2} & a_2 & \sqrt{b_3} & 0 & \dots \\ 0 & \dots & \dots & \dots & \dots & 0 \\ \dots & \dots & 0 & \sqrt{b_{n-2}} & a_{n-2} & \sqrt{b_{n-1}} \\ \dots & \dots & \dots & 0 & \sqrt{b_{n-1}} & a_{n-1} \end{pmatrix} \tag{B.3}$$