A Derivation and Implementation of Lanczos' Approximation of the Gamma Function

Remus Guderian C. Yap

April 2025

This paper provides a detailed outline of the derivation and computational implementation of Lanczos' approximation for the Gamma function, which approximates $\Gamma(z+1)$ by the expression, letting g be an arbitrary nonnegative parameter,

$$\Gamma(z+1) \approx \sqrt{2\pi}(z+g+1/2)^{z+1/2}e^{-(z+g+1/2)}A_g(z),$$

where $A_g(z)$ is a sum of the form

$$A_g(z) = a_0' + \frac{a_1'}{z+1} + \ldots + \frac{a_k'}{z+k}$$

whose coefficients $a_0', a_1', \ldots, a_k'$ depend on both g and k. The coefficients of $A_g(z)$ are computed using using the approach of Godfrey as detailed in his 2001 article, obtaining them as the result of the product of four matrices. A full Python implementation utilizing NumPy is presented for the case $g = 7$ with $k = 7$. With MATLAB's implementation of the Gamma function serving as a reference, the approximation demonstrates high numerical accuracy—for 120 distinct values within the interval $(0, 6]$, the residual sum of squares is around 3.4×10^{-22}, and the maximum absolute error is around 7.8×10^{-12}, thereby confirming the method's accuracy.

1 Introduction

The Gamma function is a function $\Gamma : \{w \in \mathbb{C} : \Re(w) > 0\} \to \mathbb{C}$ defined by the following improper definite integral (due to Legendre):

$$\Gamma(w) = \int_0^\infty t^{w-1}e^{-t}\,dt. \tag{1.0}$$

The Gamma function has the following key property:

$$\Gamma(w+1) = w\Gamma(w).$$

Therethrough it is related to the factorial function—if we restrict w to only be a positive integer, we have that

$$\Gamma(w+1) = w!.$$

Lanczos, in his 1964 paper, devised an approximation of the gamma function through the following representation:

$$\Gamma(z+1) = \sqrt{2\pi}(z+g+1/2)^{z+1/2}e^{-(z+g+1/2)}A_g(z), \tag{1.1}$$

where g is an arbitrary nonnegative real-valued parameter; and

$$A_g(z) = \frac{1}{2}a_0 + \frac{z}{z+1}a_1 + \frac{z(z-1)}{(z+1)(z+2)}a_2 + \ldots + \frac{z(z-1)\ldots(z-k+1)}{(z+1)\ldots(z+k)}a_k + \ldots, \quad (1.1.1)$$

$$a_k = \frac{\sqrt{2}}{\pi}\sum_{\alpha=0}^{k} C_{2\alpha}^{2k} F(\alpha), \quad (1.1.2)$$

$$F(\alpha) = (\alpha + g + 1/2)^{-(\alpha+1/2)} e^{\alpha+g+1/2}\Gamma(\alpha + 1/2), \quad (1.1.3)$$

where the expression C_v^u denotes the coefficient of the u-th order term of the Chebyshev polynomial of order v.

Lanczos thus proposed that the Gamma function can be approximated by truncating the series $A_g(z)$ as expressed in (1.1.1) at, say, k terms. That is to say, we would instead compute $A_g(z)$ by approximating it by the finite sum

$$A_g(z) = \frac{1}{2}a_0 + \frac{z}{z+1}a_1 + \frac{z(z-1)}{(z+1)(z+2)}a_2 + \ldots + \frac{z(z-1)\ldots(z-k+1)}{(z+1)\ldots(z+k)}a_k. \quad (1.1.4)$$

The more terms are used, the better the accuracy would be. Furthermore, however, it is possible through the use of partial-fraction decomposition to express $A_g(z)$ as seen in (1.1.4) in the following form:

$$A_g(z) = a_0' + \frac{a_1'}{z+1} + \ldots + \frac{a_k'}{z+k}, \quad (1.2)$$

for some constants $a_0', a_1', \ldots, a_k'$. Therefore, the Gamma function evaluated at the complex number $x + 1$ can be computed as

$$\Gamma(x+1) \approx \sqrt{2\pi}(x + g + 1/2)^{x+1/2} e^{-(x+g+1/2)} \left(a_0' + \frac{a_1'}{x+1} + \ldots + \frac{a_k'}{x+k} \right) \quad (1.3)$$

Thus, in practice, the coefficients of expression (1.2) can be precalculated once and then stored in some array of floating-point numbers in order to approximate the Gamma function using formula (1.3).

Though the expression for $F(\alpha)$ requires the calculation of $\Gamma(\alpha+1/2)$, the fact that α is a nonnegative integer alow us to evaluate that expression exactly. That can be done by noting that $\Gamma(1/2) = \sqrt{\pi}$ and using the key property of the Gamma function iteratively:

$$\begin{aligned}\Gamma(\alpha + 1/2) &= (\alpha - 1/2)\Gamma(\alpha - 1/2) \\ &= \frac{(2\alpha+1)(2\alpha-1)\ldots 5\cdot 3\cdot 1}{2^\alpha}\Gamma(1/2) \\ &= \frac{\sqrt{\pi}}{2^\alpha}\prod_{j=1}^{\alpha}(2j-1).\end{aligned}$$

As we will see later, the expression (1.1)—and so also expression (1.3)—is actually limited to the domain $\{z \in \mathbb{C} : \Re(z+1/2) > 0\}$. However, if $x+1/2 \leq 0$, then $1-x \geq 3/2$. Thereby, we can use the reflexion identity

$$\Gamma(1+x)\Gamma(1-x) = \frac{\pi x}{\sin \pi x} \implies \Gamma(1+x) = \frac{\pi x}{\Gamma(1-x)\sin \pi x},$$

to evaluate $\Gamma(x+1)$; indeed, $\Gamma(1-x)$ in this case can be evaluated through the approximation provided by the expression (1.3). This process allows us to extend the Gamma function to the entire complex plane.

This paper aims to explain the derivation of the expression (1.1) that Lanczos used to provide this method of approximating the Gamma function. Then, using Python (with the help of the NumPy library), an implementation of this approximation will be provided. To that end, a method proposed by Godfrey (2001) will be employed to precalculate the necessary coefficients $a_0', a_1', a_2', \dots$ of the expression (1.2).

2 Derivation of Lanczos' Approximation

In this section, an outline of Lanczos' derivation of his expression (1.1) will be derived. For a much more rigorous treatment of the derivation, refer to the PhD thesis of Pugh (2004).

We begin with the integral definition of the Gamma function (1.0) but with a shifted argument:

$$\Gamma(z+1) = \int_0^\infty t^z e^{-t}\, dt. \tag{2.0}$$

Thus, we require $\Re(z) > -1$. We can substitute t with pt, where p is some arbitrary positive real constant; doing so yields

$$\Gamma(z+1) = p^{z+1} \int_0^\infty t^z e^{-pt}\, dt.$$

We then set $p = z+g+1$, where g is an arbitrary nonnegative real number. So,

$$\Gamma(z+1) = (z+g+1)^{z+1} \int_0^\infty t^z e^{-(z+g+1)t}\, dt.$$

Here, we make use of a change of variable $t = 1 - \ln v$ so that $dt = (-1/v)dv$. From that subtitution, we see that the point $t = 0$ corresponds to $v = e$, and the point $t = \infty$ corresponds to $v = 0$ (taking the limit as $t \to \infty$). Hence,

$$\begin{aligned}
\Gamma(z+1) &= (z+g+1)^{z+1} \int_e^0 (1-\ln v)^z e^{-(z+g+1)(1-\ln v)} (-1/v)\, dv \\
&= (z+g+1)^{z+1} e^{-(z+g+1)} \int_0^e (1-\ln v)^z e^{(z+g+1)\ln v} v^{-1}\, dv \\
&= (z+g+1)^{z+1} e^{-(z+g+1)} \int_0^e (1-\ln v)^z e^{z\ln v} e^{g\ln v} e^{\ln v} v^{-1}\, dv \\
&= (z+g+1)^{z+1} e^{-(z+g+1)} \int_0^e (1-\ln v)^z v^z v^g\, dv,
\end{aligned}$$

finally arriving at

$$\Gamma(z+1) = (z+g+1)^{z+1} e^{-(z+g+1)} \int_0^e [v(1-\ln v)]^z v^g\, dv. \tag{2.1}$$

Let us now express v as a function that depends on some other variable, say θ. We thus implicitly define $v(\theta)$ by the relation

$$\cos^2\theta = v - v\ln v, \tag{2.2}$$

where the points $\theta = -\pi/2$, $\theta = 0$, $\theta = \pi/2$ correspond to the points $v = 0$, $v = 1$, and $v = e$ respectively.

By differentiating both sides of (2.2) with respect to θ through implicit differentiation, we see that $-2\sin\theta\cos\theta = -\ln v\left(\frac{dv}{d\theta}\right)$; hence,

$$2\sin\theta\cos\theta\, d\theta = \ln v\, dv. \tag{2.2.1}$$

Using (2.2.1), the expression (2.1) becomes

$$\Gamma(z+1) = (z+g+1)^{z+1}e^{-(z+g+1)}\int_{-\pi/2}^{\pi/2}\cos^{2z+1}\theta\left(\frac{2v^g\sin\theta}{\ln v}\right)d\theta. \tag{2.3}$$

Let us replace z by $z - 1/2$. By doing so, we now how to require that the real part of z must be strictly greater than $-1/2$. Whence

$$\Gamma(z+1/2) = (z+g+1/2)^{z+1/2}e^{-(z+g+1/2)}\int_{-\pi/2}^{\pi/2}\cos^{2z}\theta\left(\frac{2v^g\sin\theta}{\ln v}\right)d\theta. \tag{2.4}$$

Denote $f(\theta) = 2v^g\sin\theta/(\ln v)$, and denote $F(z)$ by the "integral transform"

$$F(z) = \int_{-\pi/2}^{\pi/2}\cos^{2z}(\theta)f(\theta)\,d\theta,$$

therethrough turning (2.4) into

$$\Gamma(z+1/2) = (z+g+1/2)^{z+1/2}e^{-(z+g+1/2)}F(z). \tag{2.5}$$

The function $f(\theta)$ can be expressed as the sum of its even and odd parts; viz., we can let $f(\theta) = f_E(\theta) + f_O(\theta)$, where $f_E(\theta) = \frac{1}{2}[f(\theta) + f(-\theta)]$ and $f_O(\theta) = \frac{1}{2}[f(\theta) - f(-\theta)]$. By that decomposition of $f(\theta)$ into a sum of an even and odd function, we see that

$$F(z) = \int_{-\pi/2}^{\pi/2}\cos^{2z}(\theta)f_E(\theta)\,d\theta + \int_{-\pi/2}^{\pi/2}\cos^{2z}(\theta)f_O(\theta)\,d\theta = \int_{-\pi/2}^{\pi/2}\cos^{2z}(\theta)f_E(\theta)\,d\theta.$$

Let us now express $f_E(\theta)$ as a uniformly convergent Fourier series valid on the interval $[-\pi/2, \pi/2]$ (see Pugh (2004) for justification), which should only involve cosines since $f_E(\theta)$ is an even function:

$$f_E(\theta) = \frac{1}{2}c_0 + c_1\cos(2\theta) + c_2\cos(4\theta) + \ldots = \frac{1}{2}c_0 + \sum_{k=1}^{\infty}c_k\cos(2k\theta),$$

where $c_k = \frac{2}{\pi}\int_{-\pi/2}^{\pi/2}f_E(\theta)\cos(2k\theta)\,d\theta$. Then, we can express each $\cos(2k\theta)$ in terms of $\cos^{2\alpha}(\theta)$ using Chebyshev polynomials (of the first kind):

$$T_{2k}(\xi) = \sum_{\alpha=0}^{k}C_{2\alpha}^{2k}\xi^{2\alpha},$$

whence, by the characteristic property of Chebyshev polynomials,

$$\cos(2k\theta) = T_{2k}(\cos\theta) = \sum_{\alpha=0}^{k}C_{2\alpha}^{2k}\cos^{2\alpha}\theta.$$

So,

$$\begin{aligned} c_k &= \frac{2}{\pi}\int_{-\pi/2}^{\pi/2}\left(f_E(\theta)\sum_{\alpha=0}^{k} C_{2\alpha}^{2k}\cos^{2\alpha}\theta\right)d\theta \\ &= \frac{2}{\pi}\sum_{\alpha=0}^{k}\left(\int_{-\pi/2}^{\pi/2}\cos^{2\alpha}(\theta)f_E(\theta)\,d\theta\right) \qquad (2.6) \\ &= \frac{2}{\pi}\sum_{\alpha=0}^{k} C_{2\alpha}^{2k}F(\alpha). \end{aligned}$$

Notice that, by solving for $F(z)$ in (2.5), we see that

$$F(\alpha) = (\alpha+g+1/2)^{-(\alpha+1/2)}e^{\alpha+g+1/2}\Gamma(\alpha+1/2),$$

which allows us to calculate $F(\alpha)$ exactly. Now, with those coefficients, we can write

$$\begin{aligned} F(z) &= \int_{-\pi/2}^{\pi/2}\cos^{2z}(\theta)\left(\frac{1}{2}c_0+\sum_{k=1}^{\infty}c_k\cos(2k\theta)\right)d\theta \\ &= \frac{1}{2}c_0\int_{-\pi/2}^{\pi/2}cos^{2z}\theta\,d\theta+\int_{-\pi/2}^{\pi/2}\left(\sum_{k=1}^{\infty}c_k\cos^{2z}(\theta)\cos(2k\theta)\right)d\theta \qquad (2.6.1) \\ &= \frac{1}{2}c_0\int_{-\pi/2}^{\pi/2}cos^{2z}\theta\,d\theta+\sum_{k=1}^{\infty}\left(c_k\int_{-\pi/2}^{\pi/2}\cos^{2z}(\theta)\cos(2k\theta)\,d\theta\right). \end{aligned}$$

Lanczos then invoked the following closed-form expression:

$$\int_{-\pi/2}^{\pi/2}\cos^{2z}(\theta)\cos(2k\theta)\,d\theta = \sqrt{\pi}\frac{\Gamma(z+1/2)}{\Gamma(z+k+1)}\frac{\Gamma(z+1)}{\Gamma(z-k+1)},$$

allowing us to write (2.6.1) as

$$F(z) = \frac{1}{2}\sqrt{\pi}\frac{\Gamma(z+1/2)}{\Gamma(z+1)}c_0+\sqrt{\pi}\frac{\Gamma(z+1/2)\Gamma(z+1)}{\Gamma(z+2)\Gamma(z)}c_1+\sqrt{\pi}\frac{\Gamma(z+1/2)\Gamma(z+1)}{\Gamma(z+3)\Gamma(z-1)}c_2+\dots.$$

Now, by the key property of the Gamma function, we have that

$$\frac{\Gamma(z+1)}{\Gamma(z-k+1)} = z(z-1)(z-2)\dots(z-k+1)$$

and

$$\Gamma(z+k+1) = (z+1)(z+2)\dots(z+k)\Gamma(z+1).$$

Therefore, we see that

$$F(z) = \sqrt{\pi}\frac{\Gamma(z+1/2)}{\Gamma(z+1)}\left(\frac{1}{2}c_0+\frac{z}{z+1}c_1+\frac{z(z-1)}{(z+1)(z+2)}c_2+\dots\right).$$

Lanczos then decided to factor out a $\sqrt{2}$ out of the bottommost expression in (2.6), essentially defining a new set of coefficients

$$a_k = \frac{1}{\sqrt{2}}c_k \qquad (2.6.2)$$

for each $k = 0, 1, 2, \ldots$, resulting in the expression

$$F(z) = \sqrt{2\pi}\frac{\Gamma(z+1/2)}{\Gamma(z+1)}\left(\frac{1}{2}a_0 + \frac{z}{z+1}a_1 + \frac{z(z-1)}{(z+1)(z+2)}a_2 + \ldots\right). \tag{2.7}$$

Putting (2.4) and (2.7) together, we get that

$$\Gamma(z+1/2) = (z+g+1/2)^{z+1/2}e^{-(z+g+1/2)} \cdot \sqrt{2\pi}\frac{\Gamma(z+1/2)}{\Gamma(z+1)}\left(\frac{1}{2}a_0 + \frac{z}{z+1}a_1 + \ldots\right).$$

Therefrom, multiplying both sides by $\Gamma(z+1)/\Gamma(z+1/2)$, we finally arrive at

$$\Gamma(z+1) = \sqrt{2\pi}(z+g+1/2)^{z+1/2}e^{-(z+g+1/2)}\left(\frac{1}{2}a_0 + \frac{z}{z+1}a_1 + \frac{z(z-1)}{(z+1)(z+2)}a_2 + \ldots\right);$$

indeed, by all that we had established so far, we recognize that expression to be precisely expression (1.1).

3 Calculating The Coefficients

Note. Following the conventions of programming languages, the counting of vector and matrix indices starts from zero. So, index 0 refers to the first (i.e., topmost) component of a column vector, and index (0, 0) refers to the entry in the upper-left corner of a matrix.

Recall that, by (2.6) and (2.6.2), the k-term sum's coefficients a_j ($j \in \{0, ..., k\}$) in (1.1.4) can be obtained through the formula

$$a_j = \sum_{\alpha=0}^{j} C_{2\alpha}^{2j}\Phi_g(\alpha), \tag{3.0}$$

where $\Phi_g(\alpha)$ is given by

$$\Phi_g(\alpha) = \frac{\sqrt{2}}{\pi}\Gamma(\alpha+1/2)(\alpha+g+1/2)^{-(\alpha+1/2)}e^{\alpha+g+1/2}.$$

In an article by Godfrey (2001), it was demonstrated that the coefficients a'_k in (1.2) can be calculated by taking the product of four particular matrices, which he obtained by starting out with expression (3.0). Using the case where $g = 5$ to calculate seven coefficients ($k = 6$), Godfrey provided a concrete demonstration on how to make use of his method in practice. This section aims to illustrate the steps that method.

Godfrey started out by letting $\mathbf{f}$ be a column vector of length $k+1$ where the a-th entry is $\Phi_g(a)$. In the case where $k = 6$ and $g = 5$, following the example of Godfrey in his article, the vector $\mathbf{f}$ would be

$$\mathbf{f} = \begin{pmatrix} 83.24 \\ 16.01 \\ 7.02 \\ 4.10 \\ 2.78 \\ 2.06 \\ 1.63 \end{pmatrix}.$$

The expression $C_{2\alpha}^{2k}$ in (3.0) can be thought of as an entry of a $(k+1) \times (k+1)$ lower-triangular matrix $\mathbf{C}$ that stores the coefficients of even-order terms of Chebyshev polynomials of even orders up to $2k$. For $k = 6$, one has

$$\mathbf{C} = \begin{pmatrix} 1/2 & & & & & & \\ -1 & 2 & & & & & \\ 1 & -8 & 8 & & & & \\ -1 & 18 & -48 & 32 & & & \\ 1 & -32 & 160 & -256 & 128 & & \\ -1 & 50 & -400 & 1120 & -1280 & 512 & \\ 1 & -72 & 840 & -3584 & 6912 & -6144 & 2048 \end{pmatrix}.$$

The entry at index (0, 0) was supposed to be a 1, but it was made into a 1/2 in order to compensate for the fact that a_0 is multiplied by 1/2 in the expression for $A_g(z)$ in (1.2). Now define $\mathbf{a}$ to be a vector of length $k+1$ such that $\mathbf{a} = \begin{pmatrix} a_0 & a_1 & \dots & a_k \end{pmatrix}^\top$. Godfrey then pointed out that, by noticing the form of the summation in the formula that gives the coefficients a_j, we have

$$\mathbf{a} = \mathbf{Cf}.$$

Godfrey then devised a way on how to deal with the partial fraction decomposition in order to compute the coefficients a'_j as seen in expression (1.2). He first defines a $(k+1) \times (k+1)$ upper-triangular matrix $\mathbf{B}$ such that its entries $B_{i,j}$ are given by, following the formulation of Pugh (2004),

$$B_{i,j} = \begin{cases} 0, & \text{if } i < j, \\ 1, & \text{if } i = 0 \text{ and } j \geq 0, \\ (-1)^{j-i} \frac{(j+i-1)!}{(2i-1)!(j-1)!}, & \text{if } i \leq j. \end{cases}$$

In the case where $k = 6$, we have that

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ & 1 & -2 & 3 & -4 & 5 & -6 \\ & & 1 & -4 & 10 & -20 & 35 \\ & & & 1 & -6 & 21 & -56 \\ & & & & 1 & -8 & 36 \\ & & & & & 1 & -10 \\ & & & & & & 1 \end{pmatrix}$$

Godfrey finally defines a $(k+1) \times (k+1)$ diagonal matrix $\mathbf{D}$ such that the (0, 0) entry is 1 and the other entries $D_{i,i}$ $(1 \leq i \leq k)$ are given by

$$D_{i,i} = -\frac{(2i-1)!}{(i-1!)^2}.$$

When $k = 6$, the matrix $\mathbf{D}$ would be

$$\begin{pmatrix} 1 & & & & & & \\ & -1 & & & & & \\ & & -6 & & & & \\ & & & -30 & & & \\ & & & & -140 & & \\ & & & & & -630 & \\ & & & & & & -2772 \end{pmatrix}$$

Let the vector $\mathbf{a}'$ be the vector $\begin{pmatrix} a'_0 & \cdots & a'_k \end{pmatrix}^\top$ containing the coefficients a'_k of the sum (1.2) that we need to make use of formula (1.3) in order to implement Lanczos' approximation. Godfrey eventually arrived at the vector formula

$$\mathbf{a}' = \mathbf{DBCf}.$$

Following Godfrey's example where $g = 5$ and $k = 6$, the value of the matrix **DBC** turns out to be

$$\mathbf{DBC} = \begin{pmatrix} 1/2 & -42 & 560 & -2688 & 5760 & -5632 & 2048 \\ 21 & -882 & 7840 & -28224 & 48384 & -39424 & 12288 \\ -420 & 23520 & -235200 & 903168 & -1612800 & 1351680 & -430080 \\ 2520 & -158760 & 1693440 & -6773760 & 12441600 & -10644480 & 3440640 \\ -6300 & 423360 & -4704000 & 19353600 & -36288000 & 31539200 & -10321920 \\ 6930 & -485100 & 5544000 & -23284800 & 44352000 & -39029760 & 12902400 \\ -2772 & 199584 & -2328480 & 9934848 & -19160064 & 17031168 & -5677056 \end{pmatrix}.$$

4 Implementation in Python

Using the Python programming language with the help of the NumPy library, we provide an implementation of Lanczos' approximation for the Gamma function, i.e., approximating the Gamma function by formula (1.3), truncating the series at 8 terms ($k = 7$) and using $g = 7$. To calculate the coefficients $a'_0, a'_1, ..., a'_7$ required to use (1.3), we will use Godfrey's approach as detailed in the previous section. Thus, we expect to see a vector of length 8 and three 8×8 matrices.

4.1 Setup

We begin by loading the necessary libraries and making a few initializations.

```python
import numpy as np

# For Chebyshev polynomials
from numpy.polynomial import Chebyshev, Polynomial

# For the factorial function
import scipy.special as sp

K = 7 # We need eight terms a'0, ..., a'7.
G = 7

SQ2 = np.sqrt(2)
PI = np.pi
SQPI = np.sqrt(PI)

# This is a cosmetic option to not use scientific notation.
# (Well, most of the time.)
np.set_printoptions(suppress=True)
```

4.2 Generating Matrices

Let us now begin calculating the coefficients $a'_0, ..., a'_7$ using the approach Godfrey used in his 2001 article. Thus, we need to calculate the matrices **f**, **C**, **B**, and **D**.

The following code calculates the components of **f**:

```
def gammaPlusHalf(A):
  result = 1.0
  for q in range(1, int(A) + 1):
    result = result * (2*q-1)

  finalresult = result * (SQPI/np.exp2(A))
  return finalresult

def F(A):
  t0 = SQ2 / PI
  t1 = np.power(A + G + 0.5, -A-0.5)
  t2 = np.exp(A + G + 0.5)
  t3 = gammaPlusHalf(A)
  return t0*t1*t2*t3

f = np.array([[F(i)] for i in range(K + 1)], dtype=np.float128)
```

Then, to calculate the matrix **C**, we can use NumPy's "Chebyshev" and "Polynomial" modules to get the necessary Chebyshev polynomial coefficients.

```
def even_chebyshev_matrix(k):
  max_terms = k + 1
  matrix = np.zeros((k + 1, max_terms), dtype=np.float128)

  for i in range(k + 1):
    degree = 2 * i
    T_cheb = Chebyshev.basis(degree)
    T_poly = T_cheb.convert(kind=Polynomial)  # Convert to monomial basis
    even_coeffs = T_poly.coef[::2]  # Take only even-order terms
    matrix[i, :len(even_coeffs)] = even_coeffs

  return matrix

def Cmat(k):
  CMat = even_chebyshev_matrix(K)
  CMat[0,0] = 0.5
  return CMat

C = Cmat(K)

print_pretty(C)
```

Then, the following code calculates the matrix **B**:

```
def Bmat(k):
  BMat = np.zeros((k+1, k+1), dtype=np.float128)
  for x in range(k+1):
    for y in range(k+1):
      if x > y:
        BMat[x,y] = 0
        continue

      if (x == 0):
        BMat[x,y] = 1
        continue

      if (y>=x):
        v1 = ((-1)**(y-x)) * sp.factorial(y+x-1)
        v2 = sp.factorial(2*x-1) * sp.factorial(y-x)
        BMat[x,y] = v1 / v2
        continue

  return BMat

B = BMat(K)
```

Lastly, the following code computes the matrix **D**.

```
def Dmat(k):
  DMat = np.zeros((k+1, k+1), dtype=np.float128)
  for x in range(k+1):
    for y in range(k+1):
    if x != y:
      DMat[x,y] = 0
      continue

    if (x==0):
      DMat[x,y] = 1
      continue

    x1 = x-1
    v1 = -1 * sp.factorial(2*x1+1)
    v2 = sp.factorial(x1)**2
    DMat[x,y] = v1/v2

  return DMat

D = Dmat(K)
```

4.3 Matrix Outputs

Printing **f** yields the following output:

```
[[526.7663739621027616522042]
 [79.1197257892642369370151]
 [28.7402068636878951224389]
 [14.4834107275807379977550]
 [8.7147641503631039228139]
 [5.8600030751234886139400]
 [4.2500162603734059629801]
 [3.2546771031878045654651]]
```

Then, printing **C**, **B**, and **D** yield the following outputs respectively:

```
[[     0.5       0.        0.        0.        0.        0.        0.        0. ]
 [    -1.        2.        0.        0.        0.        0.        0.        0. ]
 [     1.       -8.        8.        0.        0.        0.        0.        0. ]
 [    -1.       18.      -48.       32.        0.        0.        0.        0. ]
 [     1.      -32.      160.     -256.      128.        0.        0.        0. ]
 [    -1.       50.     -400.     1120.    -1280.      512.        0.        0. ]
 [     1.      -72.      840.    -3584.     6912.    -6144.     2048.        0. ]
 [    -1.       98.    -1568.     9408.   -26880.    39424.   -28672.     8192. ]]
```

```
[[   1.     1.     1.     1.     1.     1.     1.     1.]
 [   0.     1.    -2.     3.    -4.     5.    -6.     7.]
 [   0.     0.     1.    -4.    10.   -20.    35.   -56.]
 [   0.     0.     0.     1.    -6.    21.   -56.   126.]
 [    0.     0.     0.     0.     1.    -8.    36.  -120.]
 [    0.     0.     0.     0.     0.     1.   -10.    55.]
 [    0.     0.     0.     0.     0.     0.     1.   -12.]
 [    0.     0.     0.     0.     0.     0.     0.     1.]]
```

```
[[      1.       0.       0.       0.       0.       0.       0.       0.]
 [      0.      -1.       0.       0.       0.       0.       0.       0.]
 [      0.       0.      -6.       0.       0.       0.       0.       0.]
 [      0.       0.       0.     -30.       0.       0.       0.       0.]
 [      0.       0.       0.       0.    -140.       0.       0.       0.]
 [      0.       0.       0.       0.       0.    -630.       0.       0.]
 [      0.       0.       0.       0.       0.       0.   -2772.       0.]
 [      0.       0.       0.       0.       0.       0.       0.  -12012.]]
```

Therewith we can calculate their product **DBCf**. First, we calculate the product **DBC** as those matrices mostly involve integers:

```
np.set_printoptions(suppress=True)
DBC = np.matmul(np.matmul(D, B, dtype=np.float128), C, dtype=np.float128)
```

Printing the product **DBC** yields the following output:

```
[[-5.000000000e-01  5.600000000e+01 -1.008000000e+03  6.720000000e+03
  -2.112000000e+04  3.379200000e+04 -2.662400000e+04  8.192000000e+03]
 [ 2.800000000e+01 -1.568000000e+03  1.881600000e+04 -9.408000000e+04
   2.365440000e+05 -3.153920000e+05  2.129920000e+05 -5.734400000e+04]
 [-7.560000000e+02  5.644800000e+04 -7.620480000e+05  4.064256000e+06
  -1.064448000e+07  1.459814400e+07 -1.006387200e+07  2.752512000e+06]
 [ 6.300000000e+03 -5.292000000e+05  7.620480000e+06 -4.233600000e+07
   1.140480000e+08 -1.596672000e+08  1.118208000e+08 -3.096576000e+07]
 [-2.310000000e+04  2.069760000e+06 -3.104640000e+07  1.774080000e+08
  -4.878720000e+08  6.938624000e+08 -4.920115200e+08  1.376256000e+08]
 [ 4.158000000e+04 -3.880800000e+06  5.987520000e+07 -3.492720000e+08
   9.757440000e+08 -1.405071360e+09  1.006387200e+09 -2.838528000e+08]
 [-3.603600000e+04  3.459456000e+06 -5.448643200e+07  3.228825600e+08
  -9.132963840e+08  1.328431104e+09 -9.594224640e+08  2.724986880e+08]
 [ 1.201200000e+04 -1.177176000e+06  1.883481600e+07 -1.130088960e+08
   3.228825600e+08 -4.735610880e+08  3.444080640e+08 -9.840230400e+07]]
```

Finally, we calculate the product $\mathbf{a}' = \mathbf{DBCf}$:

```
a = np.matmul(DBC,f, dtype=np.float128)
```

Printing $\mathbf{a}'$ gives

```
[[1.000000000139834810398]
 [676.52036812190885939344 30]
 [-1259.13921672262222273275 26]
 [771.32342877778864931315 18]
 [-176.61502917751204222440 72]
 [12.50734340702183544635 77]
 [-0.13857144169742241501 81]
 [0.00001036771573126316 07]]
```

4.4 Implementing the Lanczos Approximation

With all the necessary coefficients precomputed, we can finally implement the Lanczos approximation, following formula (1.3):

```
def Gamma(z, dType=np.float128):
  z = dType(z)

  if z <= -0.5:
    # Apply reflexion formula
    return np.divide(PI, (np.sin(PI*z) * Gamma(1-z, dType)))
  z -= 1 # Account for shifted argument
  x = a[0,0]
  for i in range(1, len(a)):
    x += a[i,0] / (z + i)
  t = z + G + 0.5
  y1 = np.sqrt(2*PI) * np.power(t, z+0.5)
  y2 = y1 * np.exp(-t)
  y3 = y2 * x
  return y3
```

4.5 Accuracy

Inputting `z=0.5` into our implementation of the Gamma function gives

```
>>> Gamma(0.5)
>>> 1.7724538509055401472
```

And if we take the difference between $\sqrt{\pi}$ (which we saved as `SQPI`) and the value we got for `Gamma(0.5)`, we see that

```
>>> SQPI - Gamma(0.5)
>>> -2.4265311981963577637e-14
```

It appears that our approximation is quite accurate. To actually test this, we can try to calculate the residual sum of squares (RSS) between the values given by our approximation and some reference values. For the reference, values generated using MATLAB's built-in implementation of the Gamma function were used:

```
x = 0.05:0.05:6;
y_true = gamma(x);
writematrix([y_true'], 'reference.txt');
```

These values represent the Gamma function evaluated at steps of 0.05, starting from $\Gamma(0.05)$ and continuing up to $\Gamma(6.0)$. The values that appeared in the outputted text file were then copied and pasted inside a Python list called `REFERENCE`. Test values were then generated in a similar manner using Python:

```
x_values = np.arange(0.05, 6.05, 0.05)  # [0.05, 0.1, 0.15, ..., 6.0]

# Initialize an empty list
APPROX_VALUES = []

for x in x_values:
  approx = Gamma(x)
  APPROX_VALUES.append(approx)
```

Therewith we can calculate the RSS:

```
true = np.array(REFERENCE)
approx = np.array(APPROX_VALUES)

# Compute Residual Sum of Squares (RSS)
rss = np.sum((true - approx)**2)
```

```
print("Residual Sum of Squares (RSS):", rss)
```

And the print statement outputs:

```
Residual Sum of Squares (RSS): 3.3928884819896239193e-22
```

We can see that our approximation turns out to be superb given that our RSS is tiny. We can also compute the maximum absolute error:

```
max_abs_error = max(abs(t - a) for t, a in zip(REFERENCE, APPROX_VALUES))
print("Maximum Absolute Error:", max_abs_error)
```

And the print statement outputs:

```
Maximum Absolute Error: 7.7704856438209901626e-12
```

Even when we only used 8 terms of expression (1.2) to make use of the approximation formula (1.3), we already have a excellent estimate for the Gamma function.

Bibliography

[1] Abramowitz, M., & Stegun, I. A. (Eds.). (1964). *Handbook of mathematical functions* (9th ed.). Dover Publications.

[2] Godfrey, P. (2001). A note on the computation of the convergent Lanczos complex gamma approximation. Retrieved from https://www.numericana.com/answer/info/godfrey.htm

[3] Lanczos, C. (1964). A precision approximation of the gamma function. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis, 1*(1), 86–96. https://doi.org/10.1137/0701008

[4] Online Encyclopedia of Integer Sequences. (n.d.). Sequence A002457. Retrieved from https://oeis.org/A002457

[5] Pugh, G. (2004). *An analysis of the Lanczos gamma approximation* (PhD thesis).