QUEEN'S UNIVERSITY BELFAST, SCHOOL OF MATHS & PHYSICS – MTH3023

Mini project 2

Jyoutir

October 9, 2024

OBJECTIVE: The objective of this report is to analyse the convergence behaviour of the Taylor series approximation for the error function, erf(z). More specifically, I aim to compare the absolute difference between the Taylor series approximation and the value give by Python's `math.erf()` [1] function, which is a part of Python's numerical computing suite [2]. We compute values using this over different orders of z. This will allow understanding of how quickly series converge depending on the value of z and the number of terms in the series.

SUMMARY OF PROCEDURE: The error function, $\mathrm{erf}(z)$, is defined as shown in Equation 1:

$$\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt \tag{1}$$

We approximated $\mathrm{erf}(z)$ using the Taylor series expansion, provided in Equation 2:

$$P_N(z) = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{N} \frac{(-1)^n z^{2n+1}}{n!(2n+1)} \tag{2}$$

where N is the number of terms used in the approximation.

I implemented a Python function `errfun(z, N)` to calculate the Taylor polynomial $P_N(z)$ up to order N, using series expansion from Equation 2. To verify the function, I compared the results for $z = 1$ and $N = 18$ with the value of erf(z) from Python's `math.erf()` function, as shown in Equation 1.

Next, I created a function that computes absolute difference between the Taylor approximation in Equation 2 and `math.erf(z)` (Equation 1) for increasing N, which stops when the difference falls below a specified tolerance ϵ.

Convergence was tested for four values of z ($z = 0.1, 1.0, 2.0, 3.0$), using the tolerances: $\epsilon = 10^{-15}$ for $z \in \{0.1, 1.0, 2.0\}$, and $\epsilon = 10^{-13}$ for $z = 3.0$. The results are plotted on a semi-logarithmic graph, showing the convergence behaviour for different z values by graphing the absolute difference as a function of the number of terms N in the Taylor series.

RESULTS AND DISCUSSION: Figure 1 shows the convergence of the Taylor series approximation for erf(z) at different values of z. The y-axis shows the absolute difference between the Taylor approximation, and the reference value from `math.erf()`, plotted on a logarithmic scale. The x-axis shows the number of terms, N, in the Taylor series.

From the figure, we can see several trends as the polynomial order and value of z increase:

- For small values of z, such as $z = 0.1$, the series converges rapidly, requiring fewer than 10 terms to achieve a high level of accuracy.
- At $z = 1.0$, the series converges more slowly, with around 15 terms needed to reach the same accuracy as for $z = 0.1$.
- For $z = 2.0$, the convergence slows further, requiring about 30 terms for similar precision, showing increased oscillations in the series terms.
- The slowest convergence is observed for $z = 3.0$, where more than 40 terms are necessary for an equivalent accuracy.

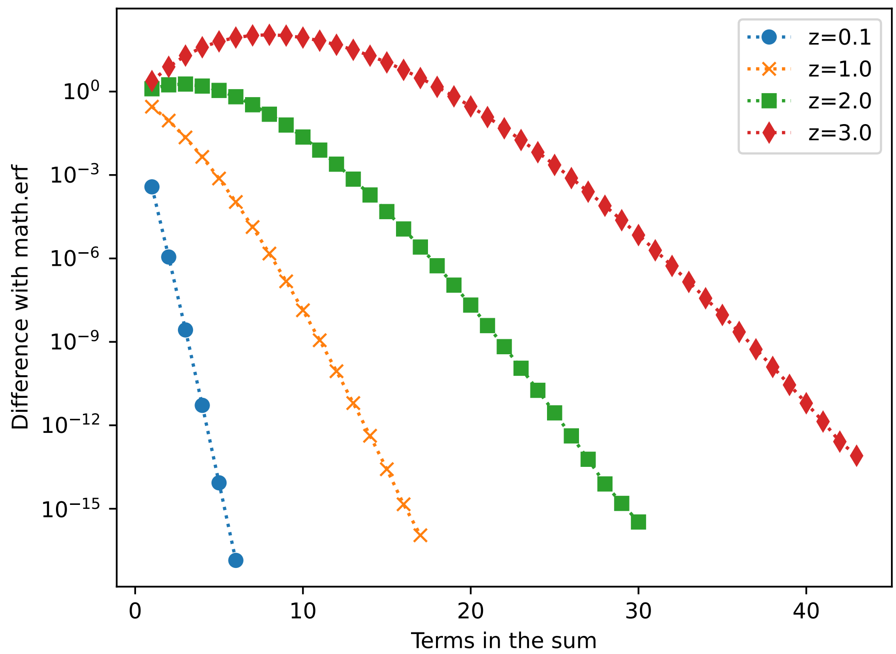

Figure 1: Convergence of Taylor polynomial for $\mathrm{erf}(z)$ for different values of z. The graph shows absolute difference between Taylor approximation and the standard $\mathrm{erf}(z)$ as a function of the number of terms in the Taylor series.

The observations can explained by looking at Taylor's theorem remainder term, $R_N(z)$, which is given by:

$$R_N(z) = \frac{f^{(N+1)}(\xi)}{(N+1)!}(z-a)^{N+1} \tag{3}$$

where ξ is between a and z, and a is the expansion point (here, $a = 0$). As z increases, the remainder term also increases, meaning more terms are needed to stay accurate. This explains the slower convergence for larger z as shown by Equation 3.

Conclusively, I looked at the convergence of the Taylor series for $\mathrm{erf}(z)$, and observed rapid convergence for small z and slower convergence as z increases, thus needs more terms for accuracy. This aligns with Taylor's theorem, where remainder term grows with z. Ultimately these findings show trade-offs between being precise and keeping low computational cost for large values of z.

References

[1] Python Software Foundation. *Python 3.10.8 Documentation - math.erf*. `https://docs.python.org/3/library/math.html#math.erf`.

[2] Travis E. Oliphant. *Python for Scientific Computing*. Computing in Science & Engineering, 9(3), 10-20, 2007. `https://doi.org/10.1109/MCSE.2007.58`.