# Function approximation and evaluation methods of mathematical functions

**Seminar Software Engineering**

Winter Semester 2023

Nishant Ingle

University of Würzburg, Germany
nishant.ingle@stud-mail.uni-wuerzburg.de

**Abstract.** This paper explores the computational methods employed by computers to calculate mathematical functions such as logarithms, exponentiation, factorials, nth roots, and trigonometric values. Given the complexity of these mathematical problems, exact solutions are elusive, necessitating the use of approximation techniques. The paper delves into the application of Taylor's series, Newton's method, and other iterative approaches, specifically tailored for efficient implementation on computers, providing swift and accurate results.

## 1 Introduction

While computers can effortlessly perform basic arithmetic operations at the hardware level, complex computations like finding the nth root lack direct hardware instructions. Fortunately, computer-friendly approximation techniques prove effective for evaluating such values efficiently. The discussion begins with an exploration of how computers employ Taylor's series and the inverse hyperbolic tangent function for logarithmic calculations. Subsequently, the paper covers methods for finding nth roots using Newton's method, exponentiation via Taylor's series and trigonometric functions employing Taylor's series. Throughout, the limitations of various approaches are also examined.

## 2 Content

### 2.1 log($x$)

#### Taylor's Series

***Definition*** Let $f$ be a function with derivatives of all orders throughout some interval containing $a$ as an interior point. Then the Taylor series generated by f at x = a is

$$f(x) = \sum_{k=0}^{\infty} f^{(k)}(a)\frac{(x-a)^k}{k!}$$

Taylor series generated by $f$ at x = 0 is called Maclaurin series:

$$f(0) = \sum_{k=0}^{\infty} f^{(k)}(0)\frac{x^k}{k!}$$

The Taylor series for $\log(x)$ can be given as

$$\log(x) = \sum_{k=0}^{\infty} (-1)^{(k+1)} \frac{(x-1)^k}{k}$$

### Pseudo-code

```
function log_using_taylor_series(number, iterations=50):
    """
    Compute the natural logarithm using Taylor series expansion.

    Parameters:

    - number (float): The number for which to calculate the logarithm.
    - iterations (int): The number of terms to include in the Taylor series expansion.

    Returns: float: The natural logarithm of the given number.
    """
    result = 0

    for i in range(1, iterations + 1):
        term_numerator = ((-1) ** (i + 1)) * ((number - 1) ** i)
        term_denominator = i * 1.0
        term = term_numerator / term_denominator
        result += term

    return result
```

### Limitations

The series for $\log(x)$ converges only when $0 < x < 2$.

### Inverse Hyperbolic Tangent

#### *Explanation*

We can leverage the artanh function to calculate logarithms for any real number x less than 0.

$\ln(x) = 2 \cdot \operatorname{arctanh}\left(\frac{x-1}{x+1}\right) = 2\sum_{i=0}^{\infty} \frac{1}{2i+1}\left(\frac{x-1}{x+1}\right)^{2i+1}$

#### Pseudo-code

```
function log_using_inverse_hyperbolic_tangent(number, iterations=50):
    """
    Compute the natural logarithm using the inverse hyperbolic tangent series expansion.

    Parameters:
    - number (float): The number for which to calculate the logarithm.
    - iterations (int): The number of terms to include in the series expansion.

    Returns:
    float: The natural logarithm of the given number.
    """
    result = 0

    for i in range(iterations):
        term_numerator = 1 / (2.0 * i + 1)
        term_denominator = ((number - 1) / (number + 1)) ** (2 * i + 1)
        term = term_numerator * term_denominator
        result += term

    return 2 * result
```

#### Limitations

Accuracy decreases as x becomes larger.

## 2.2 $\sqrt[n]{x}$

### Explanation

Newton Raphson method can be used to find root iteratively as follows

$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

To find the nth root, we can use the following formula

$x_{n+1} = \frac{\frac{\text{number}}{x_n^{(n-1)}} + (n-1) \cdot x_n}{n}$

**Pseudo-code**

```
function nth_root(number, n, max_iterations=1000):
    """
    Calculate the nth root of a given number using the Newton-Raphson method.

    Parameters:
    - number (float): The number for which to calculate the nth root.
    - n (int): The degree of the root.
    - iterations (int): The number of iterations for the Newton-Raphson method.

    Returns:
    float: The nth root of the given number.
    """
    x = 1.0
    for _ in range(max_iterations):
        x = (number / (x ** (n - 1)) + (n - 1) * x) / n
    return x
```

**Limitations**

Execution speed depends on the initial guess.

## 2.3 $a^b$

**Explanation**

We can use Maclaurin series to calculate exponent.

$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$

**Pseudo-code**

```
from math import log, e

function exp_using_ts(base, exponent, iter=50):
    """
    Calculate the exponential function using Taylor series expansion.

    Parameters:
    - base (float): The base of the exponential function.
    - exponent (float): The exponent of the exponential function.
    - iter (int): The number of terms to include in the Taylor series expansion.

    Returns:
    float: The result of the exponential function.
    """
    result = 1
    base = exponent * log(base)
    accumulator = base

    for i in range(1, iterations):
        factorial = 1
        for j in range(1, i + 1):
            factorial *= j

        term = accumulator / (factorial * 1.0)
        result += term
        accumulator *= base

    return result
```

**Limitations**

More iterations are needed as the base and power terms grow larger.

### 2.4 *n*!

**Explanation**

Stirling's approximation, also known as Stirling's formula, serves as an estimate for factorials. Renowned for its accuracy, this approximation yields precise results and bears the name of its originator, James Stirling.

$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

$\sqrt{2\pi n}\left(\frac{n}{e^{(12n+1)}}\right)^n < \text{n!} < \sqrt{2\pi n}\left(\frac{n}{e^{(12n)}}\right)^n$

**Pseudo-code**

In this implementation, we calculate the arithmetic mean of the lower and upper bounds.

```
from math import e

def fact_stirling(num):
    """
    Approximate the factorial of a number using Stirling's approximation.

    Parameters:
    - num (float): The number for which to calculate the factorial.

    Returns:
    float: The approximate factorial value using Stirling's approximation.
    """
    tl = e ** (1.0 / (12.0*num + 1))
    tu = e ** (1.0 / (12.0*num))
    lower = sqrt(2 * pi * num) * ((num / e) ** num) * tl
    upper = sqrt(2 * pi * num) * ((num / e) ** num) * tu
    return (lower + upper) / 2.0
```

**Limitations**

For smaller numbers the error % is high but it reduces as the number gets larger.

## 2.5 Trigonometric functions

### Maclaurin Series

***Definition*** Maclaurin series for sin(x), cos(x) and tan(x) can be given as follows:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$$

$$\tan(x) = x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315} + \ldots = \sum_{n=1}^{\infty} \frac{B_{2n}(2n)!}{(2n)!} \frac{x^{2n-1}}{(2n-1)!}$$

$B_{2n}$ represents the $2n$-th Bernoulli number.

### Pseudo-code

```
def sin(r, iterations=50):
"""
Calculate the sine of an angle using the Maclaurin series expansion.

Parameters:
- r (float): The angle in radians for which to calculate the sine.
- iterations (int): Half the number of terms to include in the Maclaurin series expansion.

Returns:
float: The sine of the given angle.
"""
res = r
num = r * r
den = 1.0

for i in range(3, iterations, 2):
    sign = (-1) ** ((i - 1) / 2.0)
    num *= (r ** 2)
    den *= i * (i - 1)
    res += sign * (num / den)

return res
```

### Limitations

Taylor/Maclaurin series converges only within a certain radius around the expansion point. Beyond this radius, the series may diverge, leading to inaccurate results especially in the case of trigonometric functions which are periodic in nature.

## 3 Conclusions

In conclusion, this review paper provides a comprehensive exploration of the intricate computational methods employed by computers for the approximate calculation of mathematical functions. The paper elucidates the mathematical foundations of each method, accompanied by detailed discussions on their corresponding pseudo-code implementations. By meticulously examining the usage and limitations of these approaches, we gain valuable insights into their applicability and efficiency in the realm of computer-based calculations. The paper addresses the challenges posed by the inherent complexity of mathematical problems, emphasizing the necessity of approximation techniques in the absence of exact solutions. Key methods discussed include Taylor's series, Newton's method, Stirling's approximation and other iterative approaches, each meticulously examined in the context of logarithmic calculations, nth root determinations, exponentiation, and trigonometric function evaluations.

This review paper is a valuable asset for computational mathematicians, providing theoretical insights and practical considerations for the approximate calculation of mathematical functions tailored to the limitations of computer architecture. It caters to researchers, practitioners, and enthusiasts in the field.

## References

1. George B. Thomas and Ross L. Finney, *Calculus and Analytical Geometry*, 9th edition, Addison-Wesley, Reading, MA, 1995, pages 672-708, ISBN 0201531747.
2. M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions*, Dover Publications, New York, 1972, Page 69, ISBN 0486612724.
3. Johan Verbeke and Ronald Cools, *The Newton-Raphson method*, *International Journal of Mathematical Education in Science and Technology*, 2006, volume 26, number 2, pages 177-193, DOI: 10.1080/0020739950260202.
4. Jacques Dutka, *The early history of the factorial function*, *Archive for History of Exact Sciences*, 1991, volume 43, number 3, pages 225–249, DOI: 10.1007/BF00389433, URL: `https://link.springer.com/article/10.1007/BF00389433`.

I certify that I have completed the above thesis independently and without outside help and that I have not used any aids other than those listed in the attached lists. **Furthermore, I expressly declare that, with the exception of pure translation and grammar tools, no text- and media-generating AIs (such as ChatGPT) or similar electronic programs were used for this work.** All text passages taken verbatim or in spirit from third-party publications are identified as such. All sources taken from the World Wide Web or used in digital form are appended to the work.

No other persons were involved in the intellectual output of this work. In particular, I have not used the help of a ghostwriter or a ghostwriting agency. Third parties have neither directly nor indirectly received money or monetary benefits from me for work related to the content of the submitted work.

I hereby consent to the performance of an electronic plagiarism check. The submitted electronic version of the thesis is complete. I am aware that subsequent additions are excluded.

The report has not yet been submitted to any other examination authority and has not been published. I am aware that an untrue declaration regarding the assurance of independent performance may have legal consequences.

**Würzburg, 13. December 2023**

.....
(Nishant Ingle)