Internship Report

Aanjishnu Bhattacharyya

March 9, 2025

Contents

1 Introduction

Numerical Analysis is an very important part of mathematics. It provides us with the tools nessesary to tackle real-life math problems which are hard to solve analytically. Functions such as the gamma functions and the normal distribution are extremely difficult to calculate analytically and sometimes impossible. Improper Integrals and integrals of higher dimentional functions may also be solved in a efficient manner utilizing these methods.

We have tried to build a simple yet powerful software utility which is able to integrate functions with vector valued inputs and vector valued outputs. It is possible to have multiple nested integrals with varying limits including improper integrals. Simpsons and Trapezoidal methods are used since they provide a good compromise speed and correctness. The software utility also provids a rudimentary 3-d visualization framework to plot relevant functions utilizing 3d-acceleration hardware present in most modern computers. The software utility also utilizes the feature of runtime link libraries provided by most modern operating system to provide a seameless user experience.

2 Learning Phase

In the learning phase we first develop the mathematical background required for this task.

2.1 Numerical Method of calculating Integrals

To compute the integrals of any function, we must first consider the problem of approximating the function. Approximation in this context reffers to the reconstruction of a function from n sampled points obtained from the function. This problem becomes much simpler if the sample points are spaced equally apart from eachother. Fortunately in our particular case we have direct access the the function itself thus it is trivial to obtain such points. Simpsons method and Trapezoidal method perform the best under these given constraints.

$$
\begin{aligned}
\text{Let,}\quad & f:\mathbb{R}\to\mathbb{R} \\
& f_i = f(x_i) \qquad \forall x_i \in \mathbb{R} \quad \wedge \quad 0 \le i \le n
\end{aligned}
$$

$$
\begin{aligned}
\text{Let,}\quad & L:\mathbb{R}\to\mathbb{R} \\
& L(x) = \sum_{i=0}^{n} \omega_i^n(x) f_i \\
& \omega_i^n(x) = \prod_{\substack{j=0\\ i\neq j}}^{n} \frac{x-x_j}{x_i-x_j} && i \neq j
\end{aligned}
$$

$$
\begin{aligned}
\text{Now,}\quad & I = \int_a^b f(x)\mathrm{d}x \qquad\qquad a = \min\{x_i\} \wedge b = \min\{x_i\} \\
& = \int_a^b L(x)\mathrm{d}x \\
& = \int_a^b \sum_{i=0}^{n} \omega_i^n(x) f_i \mathrm{d}x \\
& = \sum_{i=0}^{n} \int_a^b \omega_i^n(x) f_i \mathrm{d}x \\
& = \sum_{i=0}^{n} f_i \int_a^b \prod_{\substack{j=0\\ i\neq j}}^{n} \frac{x-x_j}{x_i-x_j}\mathrm{d}x
\end{aligned}
$$

$$
\begin{aligned}
\text{Let,}\quad & h = x_i - x_{i-1} \qquad\qquad \because x_i \text{ is equispaced} \\
& u = \frac{x-x_0}{h} \\
& \mathrm{d}u = \frac{1}{h}\mathrm{d}x
\end{aligned}
$$

Substituting u,

$$
\begin{aligned}
& = \sum_{i=0}^{n} f_i \int_0^n h \prod_{\substack{j=0\\ i\neq j}}^{n} \frac{u-j}{i-j}\mathrm{d}u \\
& = \sum_{i=0}^{n} f_i \frac{h\cdot -1^{n-i}}{i!\cdot(n-i)!} \int_0^n \prod_{\substack{j=0\\ i\neq j}}^{n} (u-j)\mathrm{d}u && (1)
\end{aligned}
$$

From this general formulae we can derive equations for both Simpsons ($n = 2$) and Trapezoidal ($n = 1$) rules,

From (1),

$$I = \sum_{i=0}^{n} f_i \frac{h \cdot -1^{n-i}}{i! \cdot (n-i)!} \int_0^n \prod_{\substack{j=0 \\ i \neq j}}^{n} (u-j)\mathrm{d}u$$

For n = 1,

$$= \sum_{i=0}^{1} f_i \frac{h \cdot -1^{1-i}}{i! \cdot (1-i)!} \int_0^1 \prod_{\substack{j=0 \\ i \neq j}}^{1} (u-j)\mathrm{d}u$$

$$\begin{aligned}
&= -f_0 \cdot h \cdot \int_0^1 (u-1)\mathrm{d}u + f_1 \cdot h \cdot \int_0^1 u\mathrm{d}u \\
&= -f_0 \cdot h \cdot \left[\frac{u^2}{2} - u\right]_0^1 + f_1 \cdot h \cdot \left[\frac{u^2}{2}\right]_0^1 \\
&= (f_0 + f_1)\frac{h}{2}
\end{aligned}$$

For n = 2,

$$= \sum_{i=0}^{2} f_i \frac{h \cdot -1^{2-i}}{i! \cdot (2-i)!} \int_0^2 \prod_{\substack{j=0 \\ i \neq j}}^{2} (u-j)\mathrm{d}u$$

$$\begin{aligned}
&= f_0 \cdot h \cdot \int_0^1 (u-1)(u-2)\mathrm{d}u - f_1 \cdot h \cdot \int_0^1 u(u-2)\mathrm{d}u \\
&\qquad + f_2 \cdot h \cdot \int_0^1 u(u-1)\mathrm{d}u \\
&= (f_0 + 4f_1 + f_2)\frac{h}{3}
\end{aligned}$$

2.2 Integrals with improper limits

Integrals with improper limits can often be broken down into subpieces which can be computed separately.

e.g,

$$\int_{-1}^{1} \frac{1}{x}\mathrm{d}x = \lim_{z \to 0} \left(\int_{-1}^{z} \frac{1}{x}\mathrm{d}x + \int_{z}^{1} \frac{1}{x}\mathrm{d}x \right)$$

We can also have limits which have infinities as their limits, in these cases we can construct a series with the integrals themselves and try to understand if the series converges. If it does we can also compute for what values.

e.g,

$$\int_0^\infty e^{-x^2}\mathrm{d}x$$

Let us consider the series,

$$S_n = \int_0^n e^{-x^2}\mathrm{d}x$$

We can say S_k converges if at a large enough value of k equals S_{k+1} given a particular precision.

$$S_{n+1} - S_n < |\epsilon| \qquad n \geq k \in \mathbb{R}$$

2.3 Computing Multiple Integrals

Multiple integrals to compute the volumes and other higher dimentional constructs can be computed through the same methods. Computing the limits from the outer most integral and then computing limits for these integrals from the inner most integral. How ever on of the many contrains of this system requires that each limit be defined as a function instead of conditional constraint, and it may be nesessary to convert conditional constraints into limits

e.g,

$$\iint\limits_{x^2+y^2\leq r^2} f(x,y)\mathrm{d}A$$

$$= \int_{-r}^{r}\left(\int_{-\sqrt{r^2-y^2}}^{\sqrt{r^2-y^2}} f(x,y)\mathrm{d}x\right)\mathrm{d}y$$

3 Training

3.1 Building an simple prototype of a function integrator and plotter.

We build a simple prototype* in javascript to understand how we should aproach building the integrator software[1]. During this process we avoid using several built in features and external libraries of javascript as to undestand the internals of such a system and also to keep the program as light weight as possible.

The prototype was capable of integrating any function whose domain was $\mathbb{R}^2$ and range was $\mathbb{R}$. The interface developed in this prototype resembled that of graphing software desmos. The major things that we learned from building this prototype were:

- The computation time increased rapidly with decrease in the magnitude of h so it was imperative that we language which does not have a very high overhead.
- We need to separate out the graphing part of the program from the computation part to make it effective and fast.
- We must allow the user to input data in a known syntax, and allow them to edit the functions at runtime.

*It is possible to access the prototype from any computer via the internet at: `https://nimcompoo-04.github.io/numlysis`

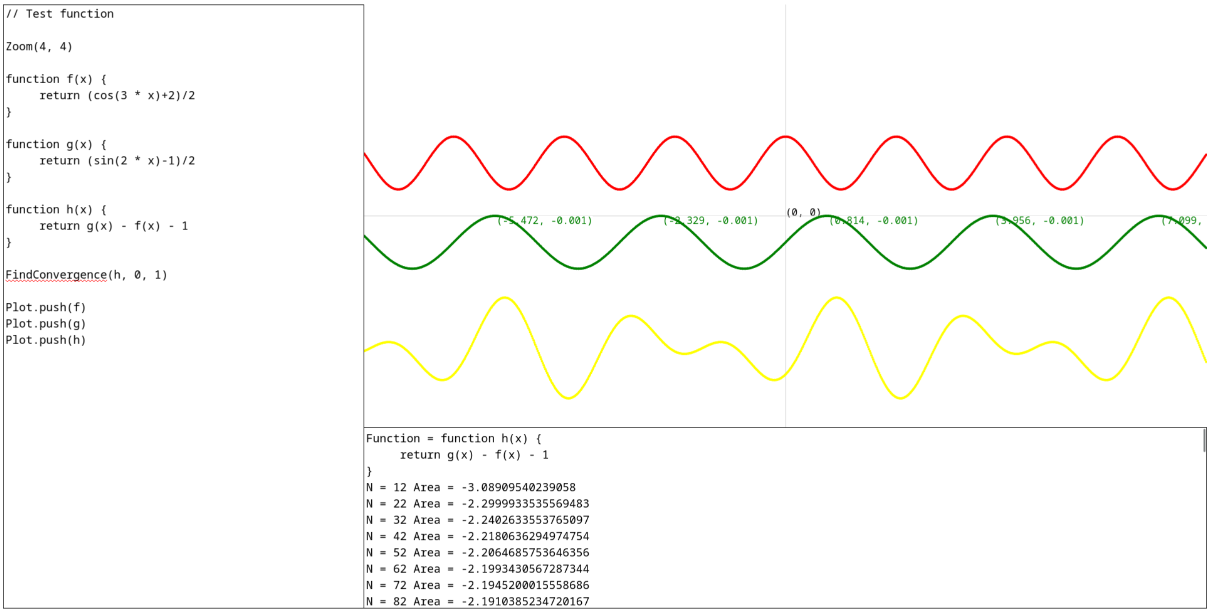

Figure 1: Integrating and plotting a trigonometric function

4 Project

We describe the different parts of the software that we built to implement these numerical methods.

4.1 Algorithmic approach to numerically solving integrals

Computing a single integrals is relatively simple. Iteratively we apply trapezoidal or simpson's rule over multiple samples taken on the function.

Algorithm 1 Computing an integral of the function f.

procedure TRAPEZOIDAL(f_0, f_1) ▷ $f_i = f(x_i)$
  return $(f_0 + f_1)/2$
end procedure

procedure INTEGRATE(s: start, e: end, f: function, n: samples)
  $h \leftarrow (e - s)/n$
  $a \leftarrow 0$ ▷ area under the curve
  for $x \leftarrow s$ **to** $e - h$ **step** h **do**
    $a \leftarrow a + h \cdot Trapezoidal(x, x + h)$
  end for
  return a
end procedure

This is however not very extensible since the function itself is hardcoded to operate on only a single variable. We try to formulate an extention by which we can reduce the number of inputs the function is operating on.

$$g(x) = \int_a^b f(x, y) dy$$

We utilize the expression to extend this algorithm to multiple integrals we devise a simple concept of 'context.' 'context' reffers to an associative array which holds the value of all the variables required to compute the function. The sidesteps the need for positional arguments.

Algorithm 2 Computing multiple integrals of multivariate function f.

```
procedure MULTI-INTEGRAL(c : context, f : function, l : limits)
    if | c |= 0 then
    else
    end if
end procedure
```

4.2 Console Interface

The graphical user interface of the prototype was discarded in favour of a very simple console interface to improve portability of the software. To provide quality of life features we utilize a console input library Crossline[†]

A part of the user input also involved accepting a C source file which contained all the function expressions. This file is used to allow runtime expression changing which is further discussed in the later sections.

The users interact with internal integrator using simple commands like 'integrate' and 'plot'. Commands like 'list' and 'pointer' were added to aid with debugging perposes.

A special set of macros were developed to allow the 'list' function to work and to allow a simple way to return vector values.

4.3 Runtime expression change

A major problem with our integrator was the slow execution times of a interpreted language. Since each function had to be executed multiple times the cost of interpreting added up extremely quickly. We noticed this problem in the prototype, the website would sometimes stop responding due to the high integrator execution times.

To address this problem it was nessesary to compile the expression down to a native binaries which would run orders of magnitude faster. To achieve this we utilized the very small c99 complient 'tiny c compiler'[‡]. We spawned a process with the tcc binaries and compiled the expression C file into a Dynamic link library (or Shared Object). The generated executable is a Position Independent (reffered to as PIE from here). This allowed use to load this executable at runtime and execute the native binary version of the expressions. An important factor which allowed this was that all the expressions did not have any side effects i.e. they did not affect memory of the main process. This meant that we did not have to deal with a situation where the structure of the code changed but the memory held in the process did not leading to fatal memory corruptions or undefined accesses of memory locations.

We also considered developing a simple scripting language, but that idea was found to be largly out of the scope of this project.

[†]Crossline-1.0 Source Code and License: `https://github.com/JunchuanWang80/crossline`.
[‡]Tiny C Compiler Source and License: `https://bellard.org/tcc/`

Some of the hurdles we faced while using the tcc compiler included the lack of support for complex data types which is standard for any c11 compilent compiler like gcc. We had to implement our own version of complex data type.

4.4 Multiple Integration

Functionally Multiple Integrals operate similarly to single integrals as demostrated in previous sections. The way we deal with this in our software is we have a C structure which represents any given integrals with its limits and the variable it is operating on. We can construct a linked list of such integrals resolving the limits of each integral from outer outer most to inner most. Then integrating the function from inner to to outer integrals. We also incorportate a special integral computation module which would allow us to compute integrals of common functions which are known not to behave well under simpsons and trapezoidal rule of integration.

4.5 Plotting of Functions

Each vector valued function is assumed to be of the form $f : \mathbb{R}^2 \to \mathbb{R}$ This assumption allows us to plot any given 2d or 1d function.

Values for multiple points are sampled from the function. A group of 3 adjescent points are selected a triangle is formed. In this manner a contiguous mesh of triangles is generated. each triangle is shaded on the basis of their normal values. The shading gives a sense of depth to the plot. Each point is recomputed each turn of the plot. To achieve higher resolution plot the number of points sampled is increased.

The triangles are displayed on the screen using the cross platform graphics library Raylib[§]. Raylib allows a streamlined interfacing with accelerated graphics hardware using the OpenGL API and a simple way to accept user keyboard input.

4.6 Special Functions

Some functions like the gamma function do not behave well with generalized integral computation methods, thus special methods have been developed to specifically deal with these functions. Currently the only function that has been implemented under this module is the Gamma function. We utilize the Lanczos Aproximation[1] for the gamma function to compute their values at a resonably high speed and acuracy.

[§] Raylib Website: `httsp://www.raylib.com`

4.7 Examples of Function Integration and Plotting

4.7.1 Integrating $sin(x)$ function from 0 to π[2]

Expression C file to integrate sin function from 0 to π,

```
#include "function.h"

FUNCTION ( X )
vec_t function(vec_t v)
{
        RETURN_VEC ( sin(v.it[0]) );
}

FUNCTION ( )
vec_t limit_start(vec_t v)
{
        RETURN_VEC ( 0 );
}

FUNCTION ( )
vec_t limit_end(vec_t v)
{
        RETURN_VEC( PI );
}
```

Integrating on variable X from 0 to 3.141593.

Step	Variable	h	Start	End	Result
1	x	0.031416	0.000000	3.141593	1.999507
2	x	0.030800	0.000000	3.141593	1.999526
3	x	0.030208	0.000000	3.141593	1.999544
4	x	0.029638	0.000000	3.141593	1.999561
5	x	0.029089	0.000000	3.141593	2.000000
6	x	0.028560	0.000000	3.141593	2.000000

4.7.2 Integrating $sin(x) + sin(y)$ function[3]

Expression C file to integrate $sin(x) + sin(y)$ under the triangle with vertices $(0, 0)(1, 0)(1, 1)$

```
#include "function.h"

FUNCTION  ( X, Y )
vec_t function(vec_t v)
{
        RETURN_VEC ( sin(v.it[0]) + sin(v.it[1]) );
}

FUNCTION ( )
vec_t limit_start_outer (vec_t v)
{
        RETURN_VEC ( 0 );
}
```

```
FUNCTION ( )
vec_t limit_end_outer  (vec_t v)
{
        RETURN_VEC ( 1 );
}

FUNCTION ( )
vec_t limit_start_inner(vec_t v)
{
        RETURN_VEC ( 0 );
}

FUNCTION ( Y )
vec_t limit_end_inner  (vec_t v)
{
        RETURN_VEC ( v.it[0] );
}
```

Integrating on variable Y then on the inner variable X under the trianglular region with verticles $(0, 0)(1, 0)(1, 1)$

Step	Variable	h	Start	End	Result
1	x	0.000000	0.000000	0.000167	0.000000
2	x	0.000000	0.000000	0.000333	0.000000
3	x	0.000000	0.000000	0.000500	0.000000
4	x	0.000001	0.000000	0.000667	0.000001
5	x	0.000001	0.000000	0.000833	0.000001
6	x	0.000001	0.000000	0.001000	0.000001
7	x	0.000001	0.000000	0.001000	0.000001
8	x	0.000001	0.000000	0.001167	0.000002
9	x	0.000001	0.000000	0.001333	0.000003
10	x	0.000002	0.000000	0.001500	0.000003
			...		
n+1	x	0.000999	0.000000	0.998833	1.300255
n+2	x	0.000999	0.000000	0.999000	1.300626
n+3	x	0.000999	0.000000	0.999000	1.300626
n+4	x	0.000999	0.000000	0.999167	1.300997
n+5	x	0.000999	0.000000	0.999333	1.299687
n+6	x	0.001000	0.000000	0.999500	1.300057
n+7	x	0.001000	0.000000	0.999667	1.302110
n+8	x	0.001000	0.000000	0.999833	1.302481
n+9	x	0.001000	0.000000	1.000000	1.301169
n+10	y	0.001000	0.000000	1.000000	0.459997

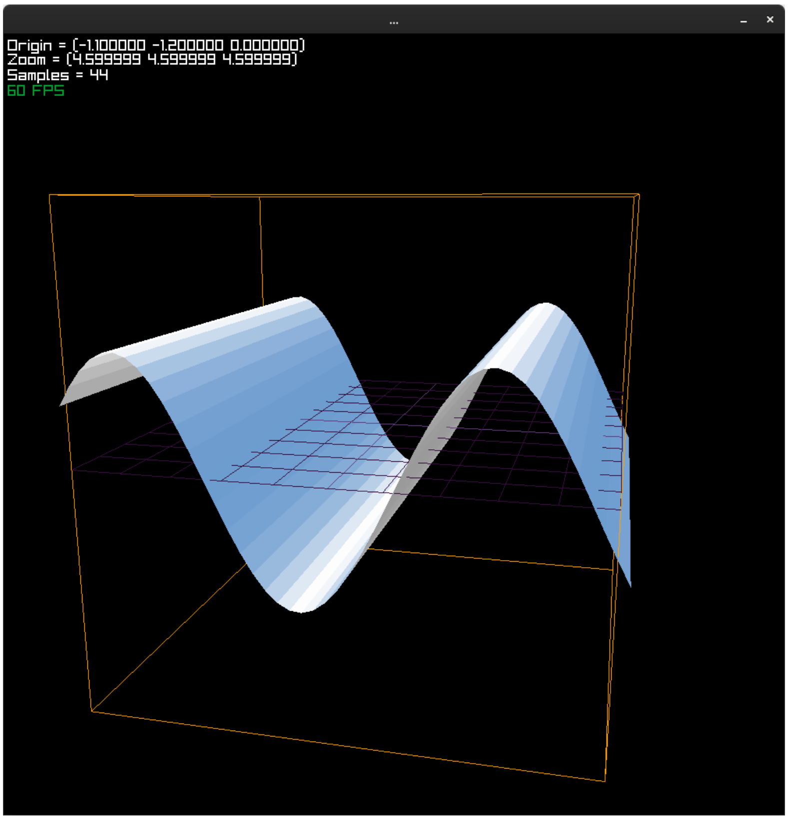

Figure 2: $sin(x)$

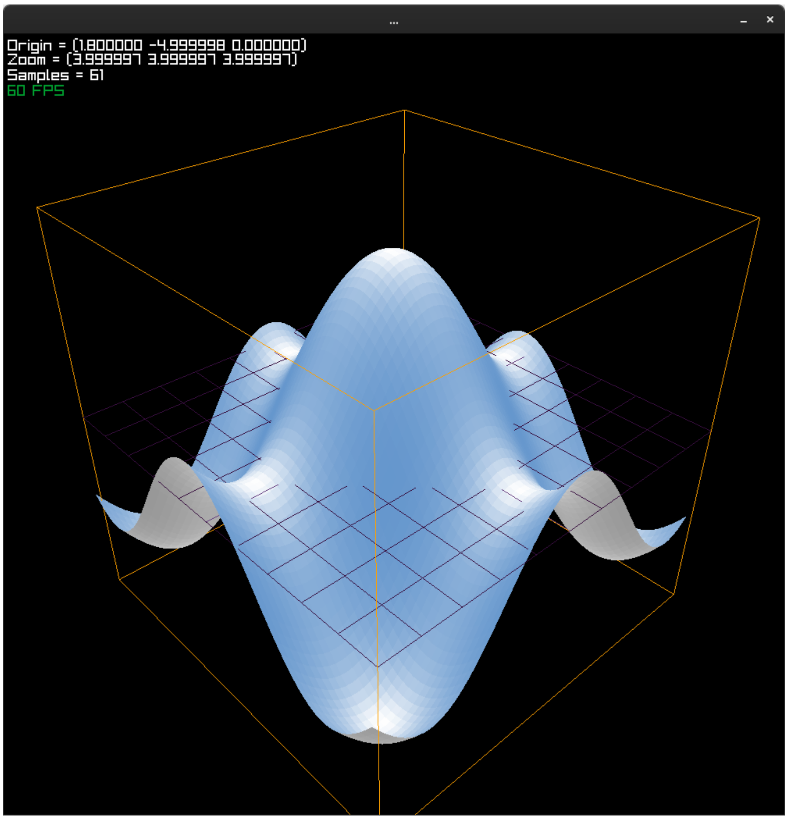

Figure 3: $sin(x) + sin(y)$

5 Results: Computing the value of the Gamma Function numerically

The gamma function can be defined as,

$$\Gamma(z) = \frac{1}{z}\prod_{n=1}^{\infty}\left[\frac{1}{1+\frac{z}{n}}\left(1+\frac{1}{n}\right)^{z}\right]$$

$$= \int_0^{\infty} t^{z-1}e^{-t}dt$$

$$= z\Gamma(z-1)$$

The gamma function is an relatively difficult function to compute with these simple numerical methods from a complexity perspective. The major reasons for this being the case are:

- The gamma function is computed through an improper integrals.
- The function grows at the rate of $O(n!)$ which is very difficult to manage, and precompute.
- The function also has discontinuities at the $-ve$ integers

To understand the limitations of the program we try to compute and plot the values of the gamma function using various methods implemented under the plotting and runtime expression compilation tooling we have built.

In the following sections we describe algorithms that have been used to compute the gamma functions. We also measured the time for computation required to execute these algorithms when implemented under the constraints of the software we have built. We utilize the implementation of gamma function in pythons math libraries to estimate the precision of each method.

5.1 Computing values of the Gamma Function using the infinite product definition.

Input	Output	Actual	Error
0.1	9.513508	9.513507698668732	3.0133126749376515e-07
0.2	4.590844	4.5908437119988035	2.8800119622474085e-07
0.3	2.991569	2.991568987687591	1.2312409314318984e-08
0.4	2.21816	2.2181595437576878	4.562423123743997e-07
2	1.0	1.0	0.0
3	1.999999	2.0	-9.999999999177334e-07
4	5.999996	6.0	-3.9999999996709334e-06
5	23.999976	24.0	-2.3999999999801958e-05

5.2 Computing values of the Gamma Function using the Euler integral definition.

Input	Output	Actual	Error
0.1	5.625076	9.513507698668732	-3.8884316986687324
0.2	3.857274	4.5908437119988035	-0.7335697119988036
0.3	2.811185	2.991568987687591	-0.18038398768759079
0.4	2.167838	2.2181595437576878	-0.0503215437576876
2	0.999988	1.0	-1.2000000000012001e-05
3	2.0	2.0	0.0
4	6.0	6.0	0.0
5	24.0	24.0	0.0

5.3 Computing values of the Gamma Function using the Lanczos approximation.

Input	Output	Actual	Error
0.1	-9.513508	9.513507698668732	-19.027015698668734
0.2	-4.590844	4.5908437119988035	-9.181687711998803
0.3	-2.991569	2.991568987687591	-5.983137987687591
0.4	-2.21816	2.2181595437576878	-4.436319543757688
2	1.0	1.0	0.0
3	2.0	2.0	0.0
4	6.0	6.0	0.0
5	24.0	24.0	0.0

6 Conclusion

dfkljdkljadjfaldkjf adklfjadlfkjlja aldkjadflj

7 References

References

[1] Godfrey, Paul (2001). "Lanczos Implementation of the Gamma Function" http://www.numericana.com/answer/info/godfrey.htm