# Assignment 2 Design Document

# Little Slice of π Scientific Write Up

Dylan Do

CSE13S - Fall 2021

**Abstract:** This paper will include comparisons between the approximation of pi, e and square root that math.h library file provides in c programming, compared to the ones I coded for the assignment, that are models of other famous approximation formulas. These will be graphed to visually represent their differences and how their iterations manifest and converge to their respective value. This is to explore and learn about how accurate these approximations of these values /math functions are from one another.

**Introduction:** In this scientific write up, we will be exploring the differences between different methods of approximations of pi, a calculation of e, and Newton's method for solving square roots. All of these approximations will be compared to the approximations found in the math.h library in C. Each of these approximation formulas perform different calculations that eventually all converge to a similar point. Although these differences are almost negligible in most applications, this paper should serve as a highlight to these minor differences in approximations. The following section will contain the formulas for the e and pi approximation(s).

*The following abstract of these approximations is found in the DESIGN.pdf*

**Calculating e:** Euler's number is an irrational mathematical constant that equates to around 2.71828. Here is the function to calculate the number:

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} = 1 + \frac{1}{1} + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \frac{1}{720} + \frac{1}{5040} + \frac{1}{40320} + \frac{1}{362880} + \frac{1}{3628800} + \cdots$$

Function provided from the assignment 2 pdf.

The amount of terms that is required to compute will be determined in experimenting as part of the assignment.

$$\frac{x^k}{k!} = \frac{x^{k-1}}{(k-1)!} \times \frac{x}{k}.$$

Function provided from the assignment 2 pdf.

Here is a refined version of the formula. According to the assignment pdf, it explains that this is simpler because the only computation required is $x/k$ starting at k = !0 (which equates to 1). Then using this calculation and multiplying the next term. Done easel with  simple for or while loop.

**Calculating $\pi$:** The assignment pdf highlights multiple functions that can be used to approximate π.

**The Madhava Series:**

$$\sum_{k=0}^{\infty} \frac{(-3)^{-k}}{2k+1} = \sqrt{3}\tan^{-1}\frac{1}{\sqrt{3}} = \frac{\pi}{\sqrt{12}}$$

Function provided from the assignment 2 pdf.

This formula is stated to be related to $\tan^{-1} x$ in the assignment pdf. A more rapid converging version of this formula is:

$$p(n) = \sqrt{12}\sum_{k=0}^{n} \frac{(-3)^{-k}}{2k+1} = \sqrt{12}\left[\frac{1}{2}3^{-n-1}\left((-1)^n\Phi\left(-\frac{1}{3},1,n+\frac{3}{2}\right)+\pi 3^{n+\frac{1}{2}}\right)\right]$$

Function provided from the assignment 2 pdf.

The only issue now is to calculate √12, and using the library is prohibited. The Lerch transcendent (Φ) goes to zero at the limit, which gives the remaining term:

$$\frac{\pi}{2}3^{-n-1}3^{n+\frac{1}{2}} = \frac{\pi}{2\sqrt{3}} = \frac{\pi}{\sqrt{12}}$$

Function provided from the assignment 2 pdf.

**Euler's Solution:**

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \cdots = H_{\infty}^{(2)},$$

Function provided from the assignment 2 pdf.

This equation involves harmonics but gave birth to this series for approximating π.

$$p(n) = \sqrt{6 \sum_{k=1}^{n} \frac{1}{k^2}}$$

Function provided from the assignment 2 pdf.

However, now there is a square root that must be solved.

**The Bailey-Borwein-Plouffe Formula:**

$$p(n) = \sum_{k=0}^{n} 16^{-k} \left( \frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right).$$

Function provided from the assignment 2 pdf.

According to the assignment pdf, it states that this formula is remarkably simple. Reducing it to the least number of multiplication, it can be rewritten in *Horner normal form*:

$$p(n) = \sum_{k=0}^{n} 16^{-k} \times \frac{(k(120k+151)+47)}{k(k(k(512k+1024)+712)+194)+15}$$

Function provided from the assignment 2 pdf.

**Viète's Formula:** This formula is an infinite product of nested radicals that can be used for the approximation of π. However, the assignment pdf states that the previous formulas are known to produce an approximation with greater accuracy.

$$\frac{2}{\pi} = \prod_{k=1}^{\infty} \frac{a_i}{2}$$

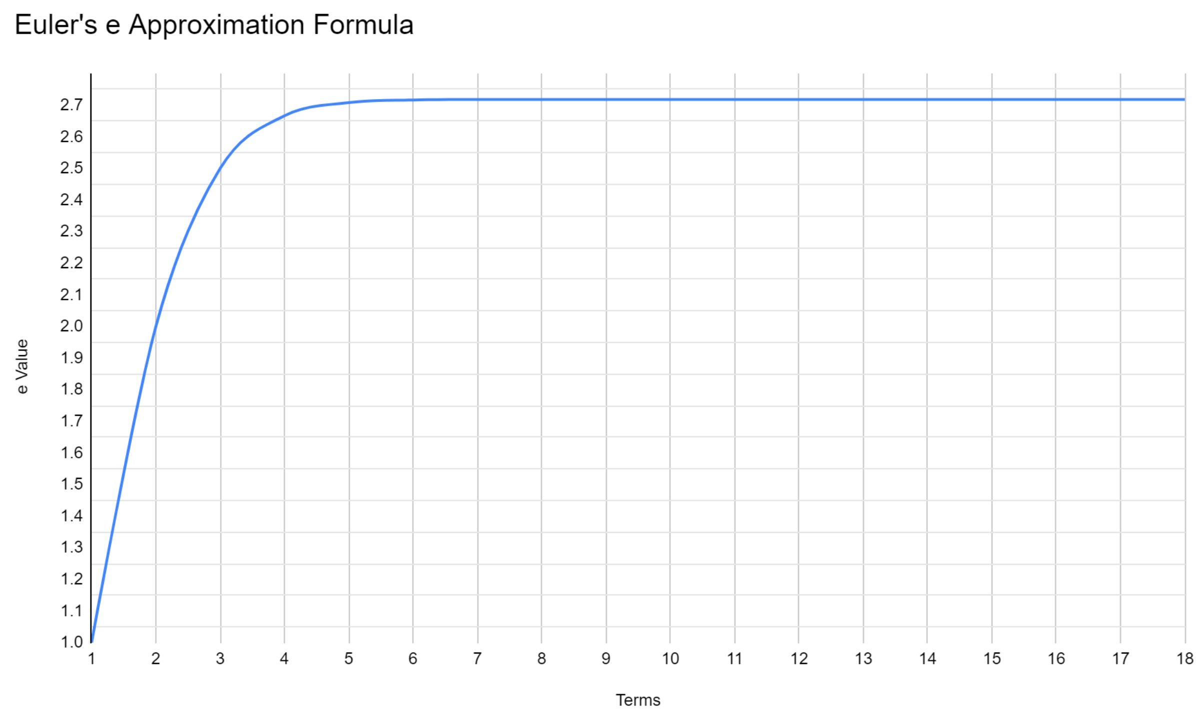
Where $a^1 = \sqrt{2}$ and $a_k = \sqrt{(2+a_{k-1})}$for all $k > 1$.

Function provided from the assignment 2 pdf.

**Method:** The method for data collecting is through the creation of a C program that implements these approximations. This program is also responsible for displaying the calculated results up to

15 decimal places and the differences between the ones found in the math.h library. The adaptation for these formulas are highlighted within the design pdf. To briefly explain, anywhere a summation is called out, it would generally entail a for or while loop to be used in order for it to compute and calculate the approximation. To stop these loops, *epsilon* is used and compared to a number that keeps on getting smaller until it is virtually zero or *epsilon*. The pseudo code for this program can be found in the design pdf.

**Results:**

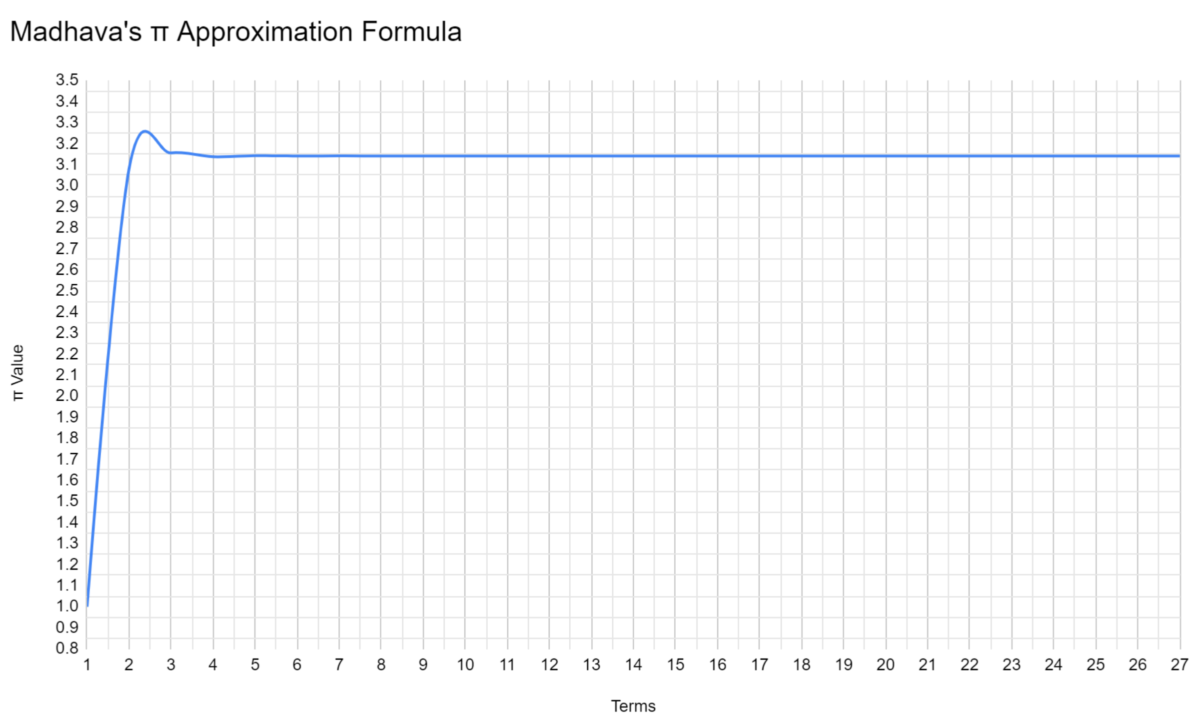



This graph above represents all the terms in respect to each iteration chronologically. (18 terms)

Euler's number Term Table:

| *Term #* | e Terms |
|---|---|
| *1* | *1.00000000000000* |
| *2* | *2.00000000000000* |
| *3* | *2.50000000000000* |

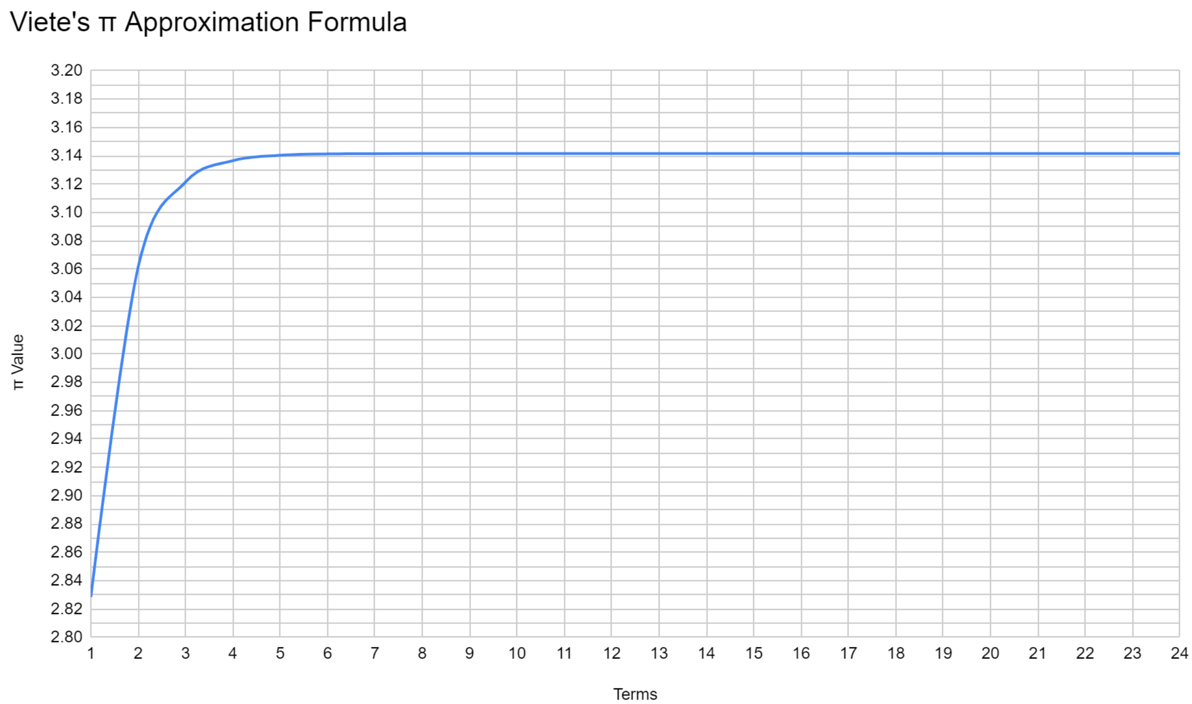
| | |
|---|---|
| *4* | *2.66666666666666* |
| *5* | *2.70833333333333* |
| *6* | *2.71666666666666* |
| *7* | *2.71805555555555* |
| *8* | *2.71825396825396* |
| *9* | *2.71827876984127* |
| *10* | *2.71828152557319* |
| *11* | *2.71828180114638* |
| *12* | *2.71828182619849* |
| *13* | *2.71828182828616* |
| *14* | *2.71828182844675* |
| *15* | *2.71828182845823* |
| *16* | *2.71828182845899* |
| *17* | *2.71828182845904* |
| *18* | *2.71828182845904* |




This graph above represents the Term per iteration of my madhava code. (27 Terms)

Madhava Term Table:

| *Term #* | Madhava Terms |
|---|---|
| *1* | 1.00000000000000 |
| *2* | *3.07920143567800* |
| *3* | *3.15618147156995* |
| *4* | *3.13785289159568* |
| *5* | *3.14260474566308* |
| *6* | *3.14130878546288* |
| *7* | *3.14167431269883* |
| *8* | *3.14156871594178* |
| *9* | *3.14159977381150* |
| *10* | *3.14159051093808* |
| *11* | *3.14159330450308* |
| *12* | *3.14159245428764* |
| *13* | *3.14159271502038* |
| *14* | *3.14159263454731* |
| *15* | *3.14159265952171* |
| *16* | *3.14159265173399* |
| *17* | *3.14159265417257* |
| *18* | *3.14159265340616* |
| 19 | *3.14159265364782* |
| 20 | 3.14159265357140 |
| 21 | 3.14159265359563 |
| 22 | 3.14159265358793 |
| 23 | 3.14159265359038 |
| 24 | 3.14159265358960 |
| 25 | 3.14159265358985 |
| 26 | 3.14159265358977 |
| 27 | 3.14159265358980 |

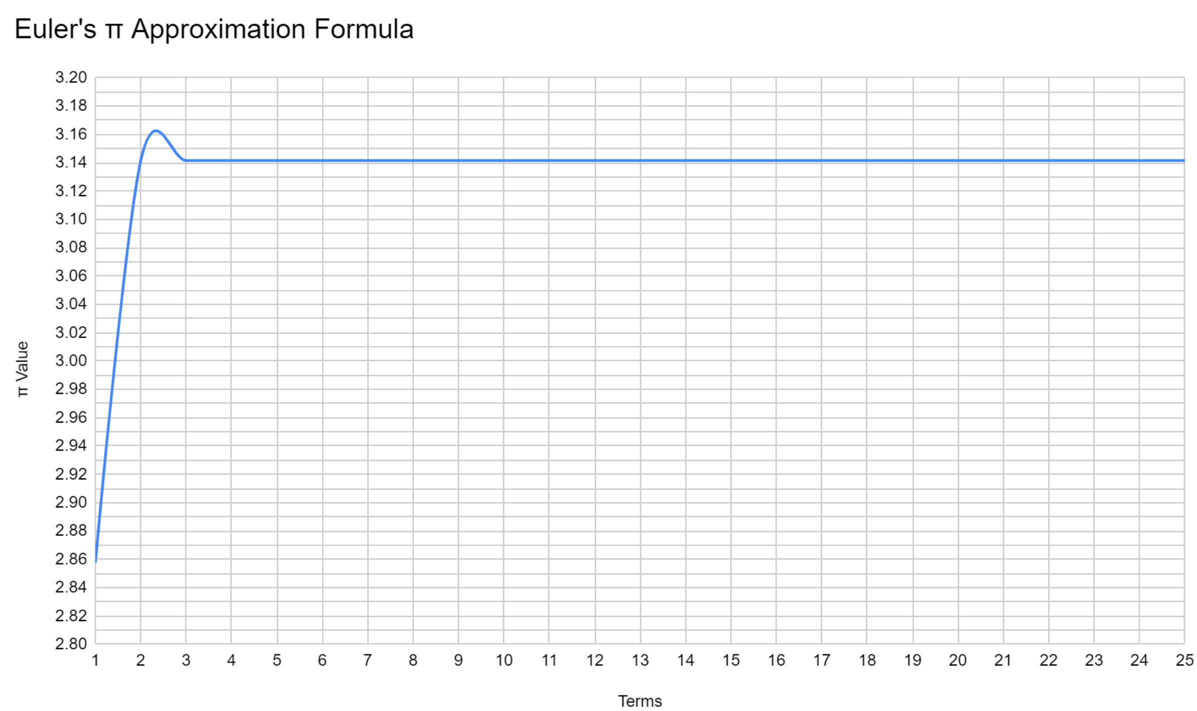



This graph above represents the Term per iteration of my viete code. (24 Terms)

## Viete Term Table:

| *Term #* | Viete Terms |
|---|---|
| *1* | 2.82842712474619 |
| *2* | *3.06146745892071* |
| *3* | *3.12144515225805* |
| *4* | *3.13654849054594* |
| *5* | *3.14033115695475* |
| *6* | *3.14127725093277* |
| *7* | *3.14151380114430* |
| *8* | *3.14157294036709* |
| *9* | *3.14158772527716* |
| *10* | *3.14159142151120* |
| *11* | *3.14159234557011* |
| *12* | *3.14159257658487* |
| *13* | *3.14159263433856* |
| *14* | *3.14159264877698* |

| | |
|---|---|
| *15* | *3.14159265238659* |
| *16* | *3.14159265328899* |
| *17* | *3.14159265351459* |
| *18* | *3.14159265357099* |
| 19 | *3.14159265358509* |
| 20 | 3.14159265358861 |
| 21 | 3.14159265358950 |
| 22 | 3.14159265358972 |
| 23 | 3.14159265358977 |
| 24 | 3.14159265358978 |

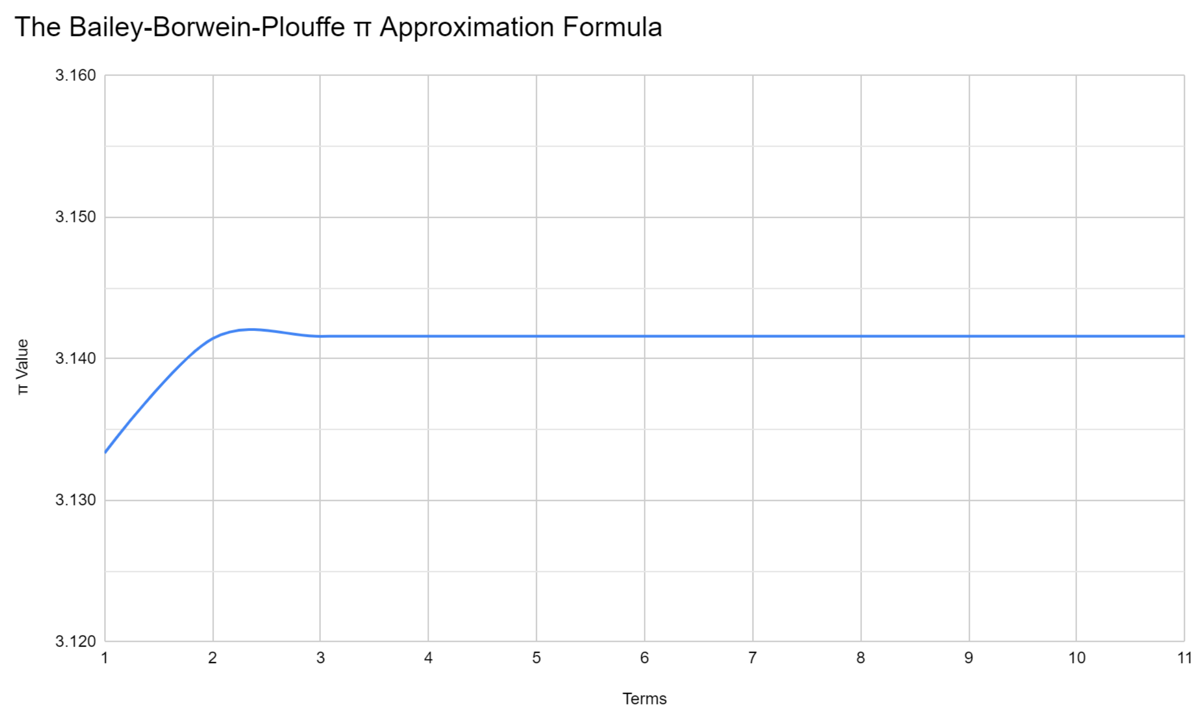



This graph above represents the Term per iteration of my euler code. (25 Terms)

## Euler Term Table:

| *Term #* | Viete Terms |
|---|---|
| *1* | 2.85773803324704 |
| *2* | *3.14159026626775* |

| | |
|---|---|
| *3* | *3.14159145992827* |
| *4* | *3.14159185781535* |
| *5* | *3.14159205675892* |
| *6* | *3.14159217612508* |
| *7* | *3.14159225570252* |
| *8* | *3.14159231254359* |
| *9* | *3.14159235517433* |
| *10* | *3.14159238833171* |
| *11* | *3.14159241485754* |
| *12* | *3.14159243656044* |
| *13* | *3.14159245464634* |
| *14* | *3.14159246994971* |
| *15* | *3.14159248306683* |
| *16* | *3.14159249443471* |
| *17* | *3.14159250438183* |
| *18* | *3.14159251315927* |
| 19 | *3.14159252096042* |
| 20 | 3.14159252794077 |
| 21 | 3.14159253422394 |
| 22 | 3.14159253990736 |
| 23 | 3.14159254507462 |
| 24 | 3.14159254979276 |
| 25 | 3.14159255809590 |




This graph above represents the Term per iteration of my bbp code. (11 Terms)

## Bailey-Borwein_Plouffe Term Table:

| *Term #* | Viete Terms |
|---|---|
| *1* | 3.13333333333333 |
| *2* | *3.14142246642246* |
| *3* | *3.14158739034658* |
| *4* | *3.14159245756743* |
| *5* | *3.14159264546033* |
| *6* | *3.14159265322808* |
| *7* | *3.14159265357288* |
| *8* | *3.14159265358897* |
| *9* | *3.14159265358975* |
| *10* | *3.14159265358979* |
| *11* | *3.14159265358979* |

Newton Square Root Difference from sqrt()

| Number | sqrt_newton | sqrt | difference |
|---|---|---|---|
| 0.0 | 0.000000000000007 | 0.0000000000000000 | *0.000000000000007* |
| 0.1 | 0.316227766016838 | 0.3162277660168380 | *0.000000000000000* |
| 0.2 | 0.447213595499958 | 0.4472135954999580 | *0.000000000000000* |
| 0.3 | 0.547722557505166 | 0.5477225575051660 | *0.000000000000000* |
| 0.4 | 0.632455532033676 | 0.6324555320336760 | *0.000000000000000* |
| 0.5 | 0.707106781186547 | 0.7071067811865480 | *0.000000000000000* |
| 0.6 | 0.774596669241483 | 0.7745966692414830 | *0.000000000000000* |
| 0.7 | 0.836660026534076 | 0.8366600265340760 | *0.000000000000000* |
| 0.8 | 0.894427190999916 | 0.8944271909999160 | *0.000000000000000* |
| 0.9 | 0.948683298050514 | 0.9486832980505140 | *0.000000000000000* |
| 1.0 | 1.000000000000000 | 1.0000000000000000 | *0.000000000000000* |
| 1.1 | 1.048808848170150 | 1.0488088481701500 | *0.000000000000000* |
| 1.2 | 1.095445115010330 | 1.0954451150103300 | *0.000000000000000* |
| 1.3 | 1.140175425099130 | 1.1401754250991300 | *0.000000000000000* |
| 1.4 | 1.183215956619920 | 1.1832159566199200 | *0.000000000000000* |
| 1.5 | 1.224744871391580 | 1.2247448713915800 | *0.000000000000000* |
| 1.6 | 1.264911064067350 | 1.2649110640673500 | *0.000000000000000* |
| 1.7 | 1.303840481040530 | 1.3038404810405300 | *0.000000000000000* |
| 1.8 | 1.341640786499870 | 1.3416407864998700 | *0.000000000000000* |
| 1.9 | 1.378404875209020 | 1.3784048752090200 | *0.000000000000000* |

| | | | |
|---|---|---|---|
| 2.0 | 1.414213562373090 | 1.4142135623730900 | *0.000000000000000* |
| 2.1 | 1.449137674618940 | 1.4491376746189400 | *0.000000000000000* |
| 2.2 | 1.483239697419130 | 1.4832396974191300 | *0.000000000000000* |
| 2.3 | 1.516575088810310 | 1.5165750888103100 | *0.000000000000000* |
| 2.4 | 1.549193338482960 | 1.5491933384829600 | *0.000000000000000* |
| 2.5 | 1.581138830084190 | 1.5811388300841900 | *0.000000000000000* |
| 2.6 | 1.612451549659710 | 1.6124515496597100 | *0.000000000000000* |
| 2.7 | 1.643167672515490 | 1.6431676725154900 | *0.000000000000000* |
| 2.8 | 1.673320053068150 | 1.6733200530681500 | *0.000000000000000* |
| 2.9 | 1.702938636592640 | 1.7029386365926400 | *0.000000000000000* |
| 3.0 | 1.732050807568870 | 1.7320508075688700 | *0.000000000000000* |
| 3.1 | 1.760681686165900 | 1.7606816861659000 | *0.000000000000000* |
| 3.2 | 1.788854381999830 | 1.7888543819998300 | *0.000000000000000* |
| 3.3 | 1.816590212458490 | 1.8165902124584900 | *0.000000000000000* |
| 3.4 | 1.843908891458570 | 1.8439088914585700 | *0.000000000000000* |
| 3.5 | 1.870828693386970 | 1.8708286933869700 | *0.000000000000000* |
| 3.6 | 1.897366596101020 | 1.8973665961010200 | *0.000000000000000* |
| 3.7 | 1.923538406167130 | 1.9235384061671300 | *0.000000000000000* |
| 3.8 | 1.949358868961790 | 1.9493588689617900 | *0.000000000000000* |
| 3.9 | 1.974841765813150 | 1.9748417658131500 | *0.000000000000000* |
| 4.0 | 2.000000000000000 | 2.0000000000000000 | *0.000000000000000* |
| 4.1 | 2.024845673131650 | 2.0248456731316500 | *0.000000000000000* |

| | | | |
|---|---|---|---|
| 4.2 | 2.049390153191920 | 2.0493901531919200 | *0.000000000000000* |
| 4.3 | 2.073644135332770 | 2.0736441353327700 | *0.000000000000000* |
| 4.4 | 2.097617696340300 | 2.0976176963403000 | *0.000000000000000* |
| 4.5 | 2.121320343559640 | 2.1213203435596400 | *0.000000000000000* |
| 4.6 | 2.144761058952720 | 2.1447610589527200 | *0.000000000000000* |
| 4.7 | 2.167948338867880 | 2.1679483388678800 | *0.000000000000000* |
| 4.8 | 2.190890230020660 | 2.1908902300206600 | *0.000000000000000* |
| 4.9 | 2.213594362117860 | 2.2135943621178600 | *0.000000000000000* |
| 5.0 | 2.236067977499780 | 2.2360679774997800 | *0.000000000000000* |
| 5.1 | 2.258317958127240 | 2.2583179581272400 | *0.000000000000000* |
| 5.2 | 2.280350850198270 | 2.2803508501982700 | *0.000000000000000* |
| 5.3 | 2.302172886644260 | 2.3021728866442600 | *0.000000000000000* |
| 5.4 | 2.323790007724440 | 2.3237900077244400 | *0.000000000000000* |
| 5.5 | 2.345207879911710 | 2.3452078799117100 | *0.000000000000000* |
| 5.6 | 2.366431913239840 | 2.3664319132398400 | *0.000000000000000* |
| 5.7 | 2.387467277262660 | 2.3874672772626600 | *0.000000000000000* |
| 5.8 | 2.408318915758450 | 2.4083189157584500 | *0.000000000000000* |
| 5.9 | 2.428991560298220 | 2.4289915602982200 | *0.000000000000000* |
| 6.0 | 2.449489742783170 | 2.4494897427831700 | *0.000000000000000* |
| 6.1 | 2.469817807045690 | 2.4698178070456900 | *0.000000000000000* |
| 6.2 | 2.489979919597740 | 2.4899799195977400 | *0.000000000000000* |
| 6.3 | 2.509980079602220 | 2.5099800796022200 | *0.000000000000000* |

| | | | |
|---|---|---|---|
| 6.4 | 2.529822128134700 | 2.5298221281347000 | *0.000000000000000* |
| 6.5 | 2.549509756796390 | 2.5495097567963900 | *0.000000000000000* |
| 6.6 | 2.569046515733020 | 2.5690465157330200 | *0.000000000000000* |
| 6.7 | 2.588435821108950 | 2.5884358211089500 | *0.000000000000000* |
| 6.8 | 2.607680962081050 | 2.6076809620810500 | *0.000000000000000* |
| 6.9 | 2.626785107312730 | 2.6267851073127300 | *0.000000000000000* |
| 7.0 | 2.645751311064580 | 2.6457513110645800 | *0.000000000000000* |
| 7.1 | 2.664582518894840 | 2.6645825188948400 | *0.000000000000000* |
| 7.2 | 2.683281572999740 | 2.6832815729997400 | *0.000000000000000* |
| 7.3 | 2.701851217221250 | 2.7018512172212500 | *0.000000000000000* |
| 7.4 | 2.720294101747080 | 2.7202941017470800 | *0.000000000000000* |
| 7.5 | 2.738612787525820 | 2.7386127875258200 | *0.000000000000000* |
| 7.6 | 2.756809750418040 | 2.7568097504180400 | *0.000000000000000* |
| 7.7 | 2.774887385102310 | 2.7748873851023100 | *0.000000000000000* |
| 7.8 | 2.792848008753780 | 2.7928480087537800 | *0.000000000000000* |
| 7.9 | 2.810693864511030 | 2.8106938645110300 | *0.000000000000000* |
| 8.0 | 2.828427124746180 | 2.8284271247461800 | *0.000000000000000* |
| 8.1 | 2.846049894151530 | 2.8460498941515300 | *0.000000000000000* |
| 8.2 | 2.863564212655260 | 2.8635642126552600 | *0.000000000000000* |
| 8.3 | 2.880972058177580 | 2.8809720581775800 | *0.000000000000000* |
| 8.4 | 2.898275349237880 | 2.8982753492378800 | *0.000000000000007* |
| 8.5 | 2.915475947422640 | 2.9154759474226400 | *0.000000000000000* |

| | | | |
|---|---|---|---|
| 8.6 | 2.932575659723030 | 2.9325756597230300 | *0.000000000000000* |
| 8.7 | 2.949576240750520 | 2.9495762407505200 | *0.000000000000000* |
| 8.8 | 2.966479394838260 | 2.9664793948382600 | *0.000000000000000* |
| 8.9 | 2.983286778035250 | 2.9832867780352500 | *0.000000000000000* |
| 9.0 | 2.999999999999990 | 2.9999999999999900 | *0.000000000000000* |
| 9.1 | 3.016620625799660 | 3.0166206257996600 | *0.000000000000000* |
| 9.2 | 3.033150177620610 | 3.0331501776206100 | *0.000000000000000* |
| 9.3 | 3.049590136395370 | 3.0495901363953700 | *0.000000000000000* |
| 9.4 | 3.065941943351170 | 3.0659419433511700 | *0.000000000000000* |
| 9.5 | 3.082207001484480 | 3.0822070014844800 | *0.000000000000000* |
| 9.6 | 3.098386676965930 | 3.0983866769659300 | *0.000000000000000* |
| 9.7 | 3.114482300479480 | 3.1144823004794800 | *0.000000000000000* |
| 9.8 | 3.130495168499700 | 3.1304951684997000 | *0.000000000000000* |
| 9.9 | 3.146426544510450 | 3.1464265445104500 | *0.000000000000000* |
| 10.0 | 3.162277660168370 | 3.1622776601683700 | *0.000000000000000* |

π Difference from from math.h

| | mathlib.h | math.h | diff |
|---|---|---|---|
| bbp | 3.141592653589790 | 3.1415926535897900 | *0.000000000000000* |
| madhava | 3.141592653589800 | 3.1415926535897900 | *0.000000000000007* |
| euler | 3.141592653589790 | 3.1415926535897900 | *0.000000095493891* |
| viete | 3.141592653589780 | 3.1415926535897900 | *0.000000000000004* |

Euler’s Number Formula difference from math.h

| mathlib.h | math.h | *diff* |
|---|---|---|
| 2.718281828459040 | 2.7182818284590400 | *0.000000000000000* |

**Discussion:**

To begin, there are a lot of numbers and graphs to digest. However this section of the write up exists for this sole purpose.

**Euler's Number Result:**

The calculation for Euler's number only took 18 iterations and calculated fairly quickly. The difference from the math.h definition of e has 0 differences showing fifteen decimal places. We can conclude that the formula and my adaptation to C coding works pretty well based on the math.h standards.

**Madhava's Approximation Result:**

The Madhava's approximation for pi calculated fairly quickly as well with only 27 iterations. The difference between the math.h definition of pi is $7 \times 10^{-15}$ which is very small but a difference nonetheless. When graphing the terms for Madhava's approximation, there is a noticeable hump at the end of the growth, which is an unknown occurrence. Based on how it calculates, it should result in a smooth curve near the end of the growth.

**Viete's Approximation Result:**

Viete's approximation for pi calculated fairly quickly too with only 14 iterations. Adjusting the graph to only show the numbers it calculated, the curve looks similar to Euler's number graph. Based on the calculation the graph seems to follow the mathematical trend the formula exhibits. The difference between the math.h definition of pi is $4 \times 10^{-15}$ which is also very small just like the difference with Madhava's approximation, but a difference nonetheless.

**Euler's Approximation Results:**

Euler's approximation for pi calculated very slowly compared to the other approximation formulas, resolving itself within $1 \times 10^{7}$ iterations. For the graph, I graphed every $4 \times 10^{5}$ term and the first term. The difference between the math.h definition of pi is $9.5493891 \times 10^{-8}$ which is a significant amount more than the rest of the differences. From this, we can conclude that Euler's approximation is a poor formula to approximate pi compared to the others.

**Bailey-Borwein-Plouffe's Approximation Results:**

Bailey-Borwein-Plouffe's approximation for pi calculated the quickest with only eleven iterations. Additionally the difference from the math.h definition of pi is 0. We can conclude that this approximation is the most accurate in regards to the definition that math.h provides.

**Newton's Method to Solving Root Results:**

From numbers 1 - 10, incrementing by $1 \times 10^{-1}$, the difference between the math.h function of sqrt and my adaptation of Newton's method to calculating roots are 0 except for the very first comparison. The square root of 0 calculated by my implementation of Newton's method resulted in $7 \times 10^{-15}$, while the function provided by math.h yielded 0. Although this difference is small, it is a difference, and very odd considering that the square root of 0 should only equal 0.

**Conclusion:**

Besides the Euler’s formula of approximation, all other formulas and math functions seemed to express great accuracy in comparison to the math.h definition and math functions. The formulas that yielded the least differences were both the Euler’s Number Formula that was adapted into code, and the Bailey-Borwein-Plouffe's approximation that was adapted into code as well. The difference calculated in the mathlib-test program resulted in 0.