J. Appl. Math. & Informatics Vol. **38***(2020), No. 3 - 4, pp. 277 - 290*
https://doi.org/10.14317/jami.2020.277

NUMERICAL ANALYSIS OF CHORDS SUMMATION ALGORITHM FOR π VALUE[†]

HYUN IL PARK, SAURAV PAHADIA, CHRISTINE HWANG AND CHI-OK HWANG*

Abstract. We propose and analyze a chord summation algorithm, which combines the ideas of Viète and Archimedes to calculate the value of π. The error of the algorithm decreases exponentially per iteration and becomes pinched at a critical iteration, depending on the accuracy of the first input value, $\sqrt{2}$. The critical iteration is also analyzed.

AMS Mathematics Subject Classification : 65D20, 65D99.
Key words and phrases : π value, chord summation algorithm, numerical analysis.

1. Introduction

Pi(π), approximately 3.14 numerically, is widely applied in the fields of mathematics, science, and engineering. Due to its popularity, there have been many attempts to calculate the numerical value of π more and more accurately. One of the earliest methods to calculate π is Archimedes algorithm from the third century BC [4], in which the perimeters of the inscribed and the circumscribed regular polygons with 6×2^nsides provide the lower and upper bounds of π respectively. Viète in the 16th century [2] twisted the Archimedes algorithm by considering the area of the regular polygons with 2^n sides inscribed in a circle, giving the infinite product expression for the value of π. Brent-Salamin algorithm [5] was also successful in calculating π, using converging arithmetic-geometric mean and elliptic integrals. There are many other methods that are also very fast and accurate for π calculation that are not mentioned here, including Chudnovsky algorithm [3], Bailey-Borwein-Plouffe formula [1], and so on.

Received December 9, 2019. Revised February 27, 2020. Accepted March 5, 2020.
*Corresponding author.
[†]This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (2017R1E1A1A03070543).

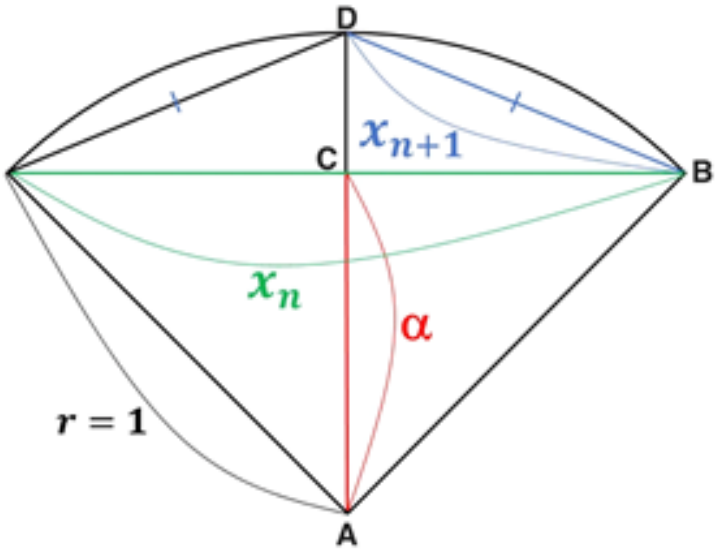

FIGURE 1. Geometry of a sector with unit radius and chords x_n and x_{n+1}.

A new and intuitive algorithm for π calculation called chords summation algorithm, which combines the ideas of Viète algorithm and Archimedes algorithm, is introduced. The error of the algorithm decreases exponentially per iteration until it becomes pinched and stops decreasing due to the approximation error of the input of $\sqrt{2}$.

2. Derivation of Chords Summation Algorithm

Using a semicircle with a radius of 1, let x_n be the length of a chord when there are 2^n congruent chords distributed within the semicircle. For example, x_1 is equal to $\sqrt{2}$, because two chords and the diameter form two isosceles right triangles with a hypotenuse of 2. A recurrence relation is derived by considering Figure 1.

Let point D lie on the semicircle such that line segment AD bisects the chord perpendicularly. Let point C be the intersection between the chord and the bisector AD. Let α be the length of the segment. Applying Pythagorean theorem on $\triangle ABC$, an algebraic expression for α is

$$\left(\frac{x_n}{2}\right)^2 + \alpha^2 = r^2 = 1. \tag{1}$$

From Eq.(1), we obtain

$$\alpha = \sqrt{1 - \frac{x_n^2}{4}}. \tag{2}$$

Applying Pythagorean theorem on $\triangle BCD$, a recurrence relation is derived:

$$(1-\alpha)^2 + \left(\frac{x_n}{2}\right)^2 = x_{n+1}^2. \tag{3}$$

Inserting α from Eq.(2) into Eq.(3), we obtain

$$\left(1 - \sqrt{1 - \frac{x_n^2}{4}}\right)^2 + \left(\frac{x_n}{2}\right)^2 = x_{n+1}^2. \tag{4}$$

Finally, the recurrence relation becomes

$$x_{n+1} = \sqrt{2 - \sqrt{4 - x_n^2}}. \tag{5}$$

The value of x_n can be successively obtained by starting with $x_1 = \sqrt{2}$. Let π_n be the sum of all the length of the chords present at the n^{th} iteration. Chords summation algorithm calculates the value of π_n,

$$\pi_n = 2^n \times x_n, \tag{6}$$

which is convergent to π,

$$\lim_{n\to\infty} \pi_n = \pi. \tag{7}$$

π_n is an appropriate choice in approximating π due to its quick exponential convergence. This will be discussed in detail in the following section.

3. Numerical Analysis

3.1. Rate of Convergence. Let e_n be the absolute error between the approximate and real value of π at the n^{th} iteration:

$$e_n = |\pi - \pi_n| = \pi - \pi_n \quad (\because \forall n \in \mathbb{N}, \quad \pi \geq \pi_n). \tag{8}$$

In theory, e_n should converge to zero as n tends to infinity. However, in practice, e_n eventually converges to a constant that is not zero. This is due to the rounding error of $\sqrt{2}$, an irrational number that cannot be expressed exactly on computer floating point numbering systems. The effect of this rounding error on the deviation from the exponential decrease of e_n will be discussed in detail in the next subsection.

In the following, it is proven that the error of chords summation algorithm, e_n, decreases by a quarter per valid iteration.

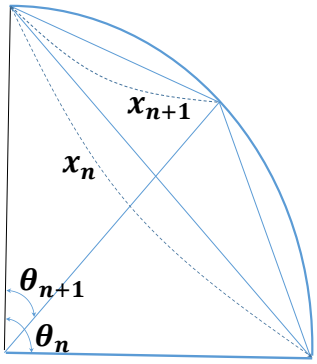

FIGURE 2. Geometry of a sector with unit radius, angles θ_n and θ_{n+1}, and chords x_n and x_{n+1}

Theorem 3.1. *The gradient of* $\log_{10} e_n$ *is:*

$$\log_{10} e_{n+1} - \log_{10} e_n = \log_{10}\left(\frac{e_{n+1}}{e_n}\right) = -0.602. \tag{9}$$

Proof. Let θ_n be the angle formed by chord x_n in Figure 2, giving an equation $\theta_n = 2 \times \theta_{n+1} = \pi/2^n$ and $x_n = 2\sin(\theta_n/2)$.

Ratio of errors between an iteration is calculated as the following:

$$\frac{e_n}{e_{n+1}} = \frac{\pi - \pi_n}{\pi - \pi_{n+1}} = \frac{\pi - 2^n \times x_n}{\pi - 2^{n+1} \times x_{n+1}} = \frac{\pi - 2^n \times 2\sin(\theta_n/2)}{\pi - 2^{n+1} \times 2\sin(\theta_{n+1}/2)}. \tag{10}$$

The RHS of Eq.(10) is simplified as

$$\frac{\pi - 2^n \times 2\sin(\theta_{n+1})}{\pi - 2^{n+1} \times 2\sin(\theta_{n+1}/2)}. \tag{11}$$

Taylor Series expansion for $\sin\theta$ up to the third order gives:

$$\sin\theta \approx \theta - \frac{\theta^3}{3!}. \tag{12}$$

Assume that θ is sufficiently small to make the approximation of Eq.(12) valid. Combining Eq.(11) and Eq.(12) gives:

$$\frac{e_n}{e_{n+1}} \approx \frac{\pi - 2^{n+1}\left(\theta_{n+1} - \frac{\theta_{n+1}^3}{6}\right)}{\pi - 2^{n+2}\left(\frac{\theta_{n+1}}{2} - \frac{\theta_{n+1}^3}{48}\right)}$$

$$= \frac{\pi - 2^{n+1}\left(\frac{\pi}{2^{n+1}} - \frac{\pi^3}{6\times 2^{3n+3}}\right)}{\pi - 2^{n+2}\left(\frac{\pi}{2^{n+2}} - \frac{\pi^3}{48\times 2^{3n+3}}\right)}$$

$$= \frac{2^{n+1}\pi^3/6 \times 2^{3n+3}}{2^{n+1}\pi^3/24 \times 2^{3n+3}}$$

$$= 4. \tag{13}$$

This result predicts that the gradient of $\log_{10} e_n$ is:

$$\log_{10} e_{n+1} - \log_{10} e_n = \log_{10}\left(\frac{e_{n+1}}{e_n}\right) = -0.602. \tag{14}$$

□

3.2. Validity of the Algorithm.

Theorem 3.2. *For a criterion of the number of valid iterations, we formulate the following:*

$$\frac{32 \times \alpha^{(k)}\delta^{(k)}}{e_n} < 0.25 \Longleftrightarrow n < \log_4\left(\frac{e_1}{32 \times \alpha^{(k)}\delta^{(k)}}\right). \tag{15}$$

Now, we will obtain n_c, the criterion of the number of valid iterations:

$$n_c = \lfloor \log_4\left(\frac{e_1}{32 \times \alpha^{(k)}\delta^{(k)}}\right) \rfloor. \tag{16}$$

Proof. Let $x_1^{(k)}$ be an approximate value of $\sqrt{2}$ with $k\,(\in \mathbb{N})$ decimal places. Let $f(x)$ be defined as $f(x) = \sqrt{2 - \sqrt{4 - x^2}}$. From Eq.(5), $x_2^{(k)}$ is calculated as $f\left(x_1^{(k)}\right)$. Likewise, $x_n^{(k)}$ is calculated as $x_n^{(k)} = f\left(x_{n-1}^{(k)}\right)$. Let $\delta^{(k)}$ be the rounding error of $x_1^{(k)}$: $\delta^{(k)} = \sqrt{2} - x_1^{(k)}$. For example, if k is 3, $x_1^{(3)} = 1.414$ and $\delta^{(3)} = 2.1356 \cdots \times 10^{-4}$.

x_2 of chords summation algorithm can be calculated as the following, using the first order Taylor expansion:

$$x_2 = f(x_1) = f\left(x_1^{(k)} + \delta^{(k)}\right) \approx f\left(x_1^{(k)}\right) + f'\left(x_1^{(k)}\right)\delta^{(k)}. \tag{17}$$

The RHS of Eq.(17) becomes

$$x_2^{(k)} + f'\left(x_1^{(k)}\right)\delta^{(k)}, \tag{18}$$

where derivative of $f(x)$ is given as the following:

$$f'(x) = \frac{x}{2\sqrt{4 - x^2} \times f(x)}. \tag{19}$$

Similarly, for x_3

$$x_3 = f\left(x_2^{(k)} + f^{'}(x_1^{(k)})\delta^{(k)}\right) \approx f\left(x_2^{(k)}\right) + f^{'}\left(x_2^{(k)}\right) f^{'}\left(x_1^{(k)}\right)\delta^{(k)}. \tag{20}$$

The RHS is simplified as

$$x_3^{(k)} + f^{'}\left(x_2^{(k)}\right) f^{'}\left(x_1^{(k)}\right)\delta^{(k)}, \tag{21}$$

and for $n > 5$,

$$x_n \approx x_n^{(k)} + f^{'}\left(x_{n-1}^{(k)}\right) f^{'}\left(x_{n-2}^{(k)}\right)\cdots f^{'}\left(x_5^{(k)}\right)\alpha^{(k)}\delta^{(k)}, \tag{22}$$

where

$$\alpha^{(k)} = f^{'}\left(x_4^{(k)}\right) f^{'}\left(x_3^{(k)}\right) f^{'}\left(x_2^{(k)}\right) f^{'}\left(x_1^{(k)}\right). \tag{23}$$

It is safe to assume $2f^{'}(x_n) \approx 1$ for a sufficiently small x_n because $2f^{'}(x)$ converges to 1 as x tends to zero. The assumption is valid for $n \geq 5$. To be specific, $2f^{'}(x_5) = 1.0009 \approx 1$, and further iterations of $2f^{'}(x_n)$ give the values closer to 1.

Let $\pi_n^{(k)}$ be the estimate π value of chords summation algorithm using $x_n^{(k)}$ instead of x_n. Referring to Equation 8, let $e_n^{(k)}$ be the absolute error defined as $e_n^{(k)} = \pi - \pi_n^{(k)}$. The numerical value of the error is $e_n^{(k)}$ rather than e_n, which is assumed to use the exact value of $\sqrt{2}$.

$$e_n^{(k)} - e_n = \left(\pi - \pi_n^{(k)}\right) - (\pi - \pi_n) = \pi_n - \pi_n^{(k)} = 2^n\left(x_n - x_n^{(k)}\right). \tag{24}$$

Finally, we obtain

$$e_n^{(k)} - e_n = 2^n \times f^{'}\left(x_{n-1}^{(k)}\right) f^{'}\left(x_{n-2}^{(k)}\right)\cdots f^{'}\left(x_5^{(k)}\right)\alpha^{(k)}\delta^{(k)} = 32 \times \alpha^{(k)}\delta^{(k)}, \tag{25}$$

where $\alpha^{(k)}$ is constant for a given k. For example, $\alpha^{(7)} = 0.0882815651$.

The value of error does not decrease due to the constant nature of $32\times\alpha^{(k)}\delta^{(k)}$, while $e_n \approx e\left(\frac{1}{4}\right)^{n-1}$ term continues to decrease. For a criterion of the number of valid iterations, we formulate the following:

$$\frac{32 \times \alpha^{(k)}\delta^{(k)}}{e_n} < 0.25 \Longleftrightarrow n < \log_4\left(\frac{e_1}{32 \times \alpha^{(k)}\delta^{(k)}}\right). \tag{26}$$

Now, we will obtain n_c, the criterion of the number of valid iterations:

$$n_c = \lfloor \log_4\left(\frac{e_1}{32 \times \alpha^{(k)}\delta^{(k)}}\right)\rfloor. \tag{27}$$

□

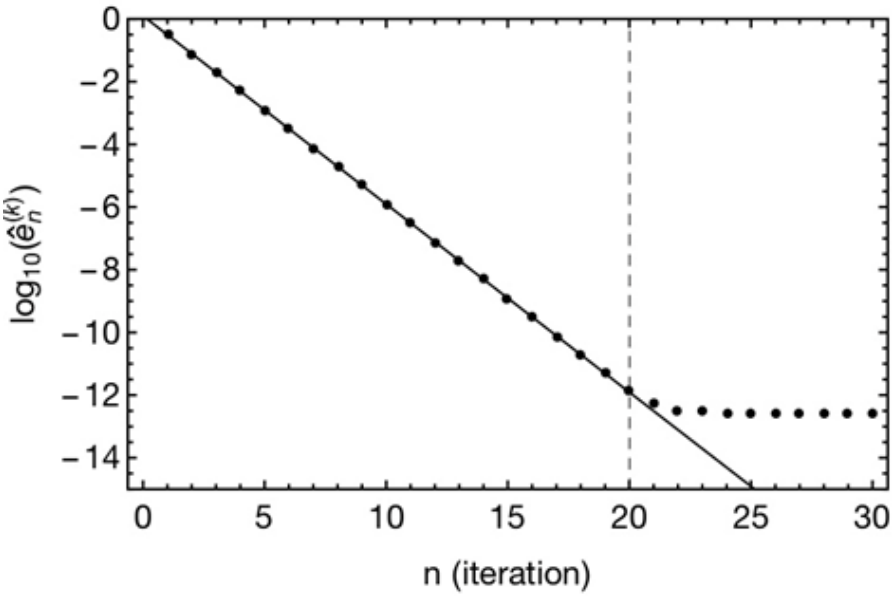

FIGURE 3. $e_n^{(k)}$ Log Values of the absolute errors at nth iteration are represented by dots. Solid line refers to the linear fitting of data points up to the 20th iteration, which is marked by dashed vertical line. Valid iterations of the algorithm are left to the dashed line. After the 20th iteration, values of the error starts to deviate from the linear fitting line, converging to -12.57.

4. Numerical Results

Figure 3 shows the log plot of $e_n^{(k)}$ with respect to iteration n, where $\sqrt{2}$, with 12 correct decimal places, is used to calculate $e_n^{(k)}$. It gives the value of π with 50 correct decimal places. It is interesting to observe an initial exponential decrease of $e_n^{(k)}$. Data points of $\log_{10} e_n^{(k)}$ lie on the straight line almost perfectly up to the 20th iteration, after which they start to deviate, eventually becoming constant. Valid iterations of chords summation algorithm are the first 20 iterations that lie on the line and decrease exponentially. It shows that $e_n^{(k)}$ eventually converges to a constant and stops decreasing after a certain number of iterations due to the rounding error of $x_1 = \sqrt{2}$.

In fact, the linear fitting in Figure 3 is applied from $n = 1$ to $n = 20$, giving the equation of $-0.600x + 0.096$. The gradient of the linear fitting line is -0.6, which is very close to $\log_{10}(1/4) = -0.602\ldots$, showing that e_n is quartered per iteration. Small discrepancy between the actual gradient (-0.6) and the predicted

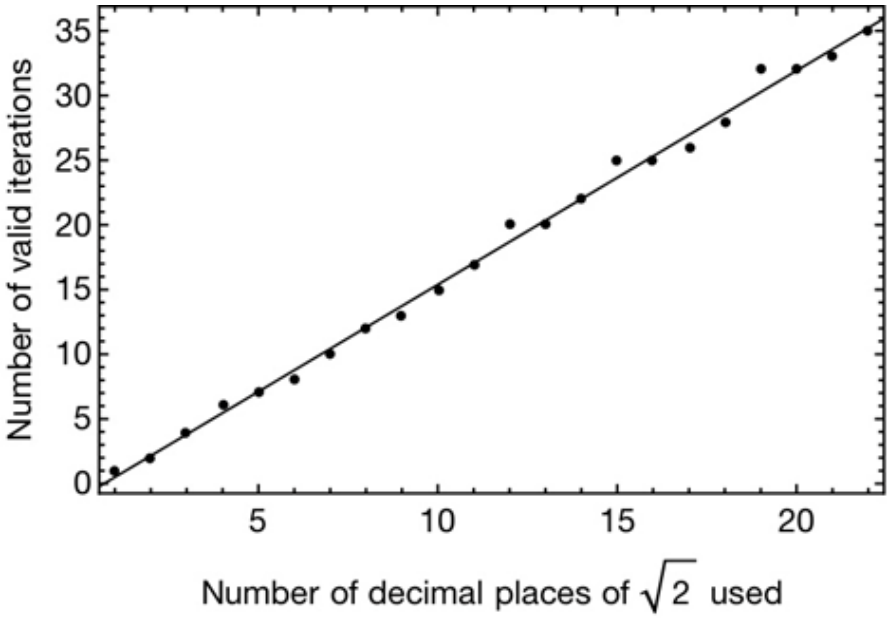

FIGURE 4. Number of decimal places of $\sqrt{2}$ used.

gradient (-0.602) shows that the Taylor expansion approximation in Eq. 12 is a valid assumption.

Also, it is observed in Figure 3 that the exponential decrease of $e_n^{(k)}$ is valid up to more iterations if more decimal places of $x_1 = \sqrt{2}$ are used. It is noted that a criterion for determining the number of valid iterations of the algorithm at a given number of decimal places of $x_1 = \sqrt{2}$ is formulated.

In Figure 4, the effect of the number of decimal places of $\sqrt{2}$ used in chords summation algorithm on the number of valid iterations, for which $e_n^{(k)}$ decreases exponentially, is shown.

5. Conclusions

In this paper, by combining the ideas of Viète and Archimedes, we proposed a chord summation algorithm for calculating the π value and analyzed the algorithm.

The chords summation algorithm provides an approximation of π and has an error that decreases exponentially. We analyzed the accuracy of the approximation and showed that the accuracy depends on the number of iterations, along with the number of decimal digits used in the initial input value of $\sqrt{2}$.

As a reference, we put our python program for the algorithm in Appendix.

Appendix

```
from decimal import *
import math
import matplotlib.pyplot as plt

piDecimalPlaces = 0
sqrtDecimalPlaces = 0
iterations = 0

# Gets maximum number of iterations valid
# based on current value of sqrtDecimalPlaces
def getMaximumIterations():
    i = (getError(Decimal('2') * Decimal('2').sqrt()) /
                ((Decimal('32') * getAlpha()
                   * getDelta())))
    i = math.log(i, 4)
    return i

# Gets values for piDecimalPlaces, sqrtDecimalPlaces, and
  iterations
# from user Displays number of maximum iterations.
def getInput():
    global piDecimalPlaces, sqrtDecimalPlaces, iterations
    piDecimalPlaces = int(input(
        "\nPlease enter the number of decimal digits for
              PI: ")) + 1
    sqrtDecimalPlaces = int(input(
        "Please enter the number of decimal digits for
               sqrt of 2: ")) + 1

    maximumIterations = int(getMaximumIterations())

    print("\nThe maximum number of valid iterations is "
          + str(maximumIterations) + "\n")
    iterations = int(input(
    "Please enter the number of iterations you would like
       to complete: "))

# Recursively calculates the chord length at the given
  number of iterations
def getChordLength(chordLength, iterationsLeft):
```

```
    if(iterationsLeft == 1):
        return chordLength

    newChordLength = Decimal(Decimal(
        '2') - Decimal(Decimal('4')
            - chordLength**2).sqrt()).sqrt()
    return getChordLength(newChordLength, iterationsLeft
        - 1)

    # Calculates the error in the given approximation
      of pi, using math.pi
# as the "correct" value for pi
def getError(piApproximation):
    getcontext().prec = piDecimalPlaces
    return abs(Decimal(math.pi) - piApproximation)

# Calculates approximation of pi using the current
  number of decimal places
# forsqrt 2, pi, and the number of iterations


def approximatePi():
    getcontext().prec = sqrtDecimalPlaces
    initialChordLength = Decimal('2').sqrt()

    getcontext().prec = piDecimalPlaces
    piApproximation = 2**iterations
                      * getChordLength(initialChordLength,
    iterations)

    return piApproximation

# Used in calculating the alpha value


def getDerivative(chordLength):

    quantizedPart = Decimal(Decimal('4')
                    - chordLength**2).sqrt()
    places = Decimal('10')** (-1*(sqrtDecimalPlaces-1))
```

```
    quantizedPart
     = quantizedPart.quantize(places, ROUND_DOWN)
    print(quantizedPart)

    getcontext().prec = sqrtDecimalPlaces
    denominator = Decimal('2') * Decimal(Decimal('4')
                  - chordLength**2).sqrt()
    * Decimal(Decimal('2') - quantizedPart).sqrt()

    return chordLength / denominator

# Used to calculate the value for alpha

def getAlpha():
    return getAlphaHelper(4)

# [HELPER METHOD] Used to recursively calculate the
 value for alpha


def getAlphaHelper(iterationsLeft):
    initialChordLength = Decimal('2').sqrt()

    if (iterationsLeft == 1):
        return getDerivative(initialChordLength)
    return getDerivative(getChordLength(initialChordLength
      ,iterationsLeft))*getAlphaHelper(iterationsLeft - 1)

# Used to calculate value of delta


def getDelta():

    getcontext().prec = sqrtDecimalPlaces * 2
    accurateRoot = Decimal('2').sqrt()
    approximateRoot = Decimal('2').sqrt()

    places = Decimal('10') ** (-1*(sqrtDecimalPlaces-1))
    approximateRoot = approximateRoot.quantize(places,
      ROUND_DOWN)
```

```
    difference = accurateRoot - approximateRoot
    return difference

# Used to create error vs. iterations graph

def iterationsVsError():
    global iterations
    originalIterations = iterations
    maximumIterations = int(getMaximumIterations())
                + 10

    iterationList = populateList(maximumIterations)
    errorList = [0] * maximumIterations

    for currentNumIterations in range(1,
     maximumIterations):
        iterations = currentNumIterations
        errorList[currentNumIterations] = math.log(
            getError(approximatePi()), 10)
    iterations = originalIterations
    plt.figure(1)
    plt.ylabel("log(e)")
    plt.xlabel("n (iteration)")
    plt.plot(iterationList, errorList, "ro")

# Used to create sqrt vs. iterations graph


def sqrtVsIterations():
    global sqrtDecimalPlaces
    maximumDecimalPlaces = sqrtDecimalPlaces
    decimalPlaceList = populateList(maximumDecimalPlaces)
    iterationList = [0] * maximumDecimalPlaces

    for currentDecimalPlace in range(2,
     maximumDecimalPlaces+1):
        sqrtDecimalPlaces = currentDecimalPlace
        iterationList[currentDecimalPlace-1]
         = getMaximumIterations()
```

```
    plt.figure(2)
    plt.ylabel("Number of Valid Iterations")
    plt.xlabel("Number of Decimal Places of Root 2 Used")
    plt.plot(decimalPlaceList, iterationList, "ro")

    def sqrtVsError():
    global sqrtDecimalPlaces
    maximumDecimalPlaces = sqrtDecimalPlaces

    decimalPlaceList = populateList(maximumDecimalPlaces)
    errorList = [0] * maximumDecimalPlaces

    for currentDecimalPlace in range(2,
    maximumDecimalPlaces+1):
        sqrtDecimalPlaces = currentDecimalPlace
        errorList[currentDecimalPlace - 1] = math.log(
            getError(approximatePi()), 10)

    plt.figure(3)
    plt.ylabel("log(e)")
    plt.xlabel("Number of Decimal Places of Root 2 Used")
    plt.plot(decimalPlaceList, errorList, "ro")


# Used to create x-axis list easily for graphs


def populateList(maximum):
    list = [0] * maximum
    for i in range(0, maximum):
        list[i] = i
    return list


getInput()
iterationsVsError()
sqrtVsIterations()
sqrtVsError()
plt.show()
```

References

1. D.H. Bailey, P.B. Borwein, and S. Plouffe, *On the rapid computation of various polylogarithmic constants*, Math. Comput. **66** (1997), 903-913.
2. P. Beckmann, *A history of π (2nd ed.)*, Golem Press, 1971.
3. D. Chudnovsky and G. Chudnovsky, *Approximation and complex multiplication according to Ramanujan*, Proceedings of the Centenary Conference 1988, 375-472.
4. T.L. Heath, *The works of archimedes*, Dover, 1953.
5. E. Salamin, *Computation of pi using arithmetic-geometric mean*, Math. Comput. **30** (1976), 565-570.

Hyun Il Park is a graduate student at University of Cambridge, Cambridge, UK.

Department of Chemical Engineering and Biotechnology, University of Cambridge, Cambridge, UK.
e-mail: hip23@cam.ac.uk

Saurav Pahadia is an undergraduate at University of Washington at Seattle, WA, USA.

Department of Computer Science, University of Washington at Seattle, WA, USA.
e-mail: sauravp@uw.edu

Christine Hwang is an undergraduate at Johns Hopkins University, Baltimore, Maryland, USA.

Department of Chemical & Biomolecular Engineering, Johns Hopkins University, Baltimore, Maryland, USA.
e-mail: chwang14@jhu.edu

Chi-Ok Hwang received M.Sc. from Seoul National University and University of Southern Mississippi, and Ph.D. from University of Southern Mississippi. He is currently a professor at Gwangju Institute of Science and Technology since 2010. His research interests are Monte Carlo methods and their applications.

Division of Liberal Arts and Sciences, GIST College, Gwangju Institute of Science and Technology, Gwangju Metropolitan City 61005, South Korea.
e-mail: chwang@gist.ac.kr