Irrationals

Carson Witt

Grade: 100

Approriate Introduction	Very nice! I really like the use of subsections within the abstract to help the reader with your report.
Proof of the irrationality of $\sqrt{2}$	Excellent outline of the proof.
Explanation of continued fractions	Very nice! It's great to see you add in the additional detail about history and background. Stuff like that really adds to the quality of your report – and also helps the reader to enjoy reading it!
Working Python code to approximate one of e, π, or $\sqrt{2}$	Yes. Good explanations of the code as well.
Use of the lstlisting environment	Yes.
Proper formatting	Yes.

List of Comments

Irrational Numbers Project

Carson Witt

May 2, 2017

Abstract

Introduction:

An irrational number is a number that cannot be expressed as a ratio of two numbers, or a fraction. Commonly known irrational numbers are the ratio π of a circle's circumference to its diameter, Euler's number e, the golden ratio ϕ, and the square root of two. All square roots of natural numbers, other than of perfect squares, are irrational. When expressed as decimals, irrational numbers do not repeat or terminate.

History:

According to Wikipedia, "The first proof of the existence of irrational numbers is usually attributed to a Pythagorean (possibly Hippasus of Metapontum), who probably discovered them while identifying sides of the pentagram. The then- current Pythagorean method would have claimed that there must be some sufficiently small, indivisible unit that could fit evenly into one of these lengths as well as the other."

Task For This Project:

In this project, I will be proving that $\sqrt{2}$ is irrational (by contradiction), explaining continued fractions and showing the continued fraction for e, $\sqrt{2}$,

and π, and writing Python code that approximates e, $\sqrt{2}$, and π to a requested number of digits.

Context/Work

Proving that $\sqrt{2}$ is irrational by contradiction:

1. Let's assume that $\sqrt{2}$ is rational, meaning it can be written as the ratio of two integers, a and b:

$$\sqrt{2} = \frac{a}{b}$$

Where $b \neq 0$ and we assume that a and b have no common factors. If common factors exist, we cancel them in the numerator and denominator.

2. Squaring both sides of the equation gives us:

$$2 = \frac{a^2}{b^2}$$

3. Which implies:

$$a^2 = 2b^2$$

4. This means that $\sqrt{a}$ must be even, since $\sqrt{a}$ is 2 multiplied by some number. We know this to be true because the multiplication of two even numbers will always be even.

5. This also means that a itself is even because if a was odd, $a * a$ would be odd as well.

6. Since a is an even number, it is 2 times another whole number.

$$a = 2k$$

7. If we substitute $a = 2k$ into the squared original equation, we get:

$$2 = \frac{(2k)^2}{b^2}$$
$$2 = \frac{4k^2}{b^2}$$
$$2b^2 = 4k^2$$
$$b^2 = 2k^2$$

8. This means that b^2 is even, which follows that b itself is even.

9. **This is where there is a contradiction.** If a and b are both even numbers, then $\frac{a}{b}$ is not in its simplest form and still has common factors. This is a contradiction because we assumed that the equation was rational and had no common factors from the start. **Therefore, $\sqrt{2}$ must be irrational.**

Continued Fractions:

A continued fraction is a fraction of infinite length whose denominator is a quantity plus a fraction, which latter fraction has a similar denominator, and so on. Continued fractions are great ways to express irrational numbers like e, π, and $\sqrt{2}$.

Interesting Facts:

- John Wallis first used the term "continued fraction" in his Arithmetica Infinitorum of 1653.
- Another word for a continued fraction is anthyphairetic ratio.

The basic form of a continued fraction is as follows:

$$a_0 + \cfrac{b_1}{a_1 + \cfrac{b_2}{a_2 + \cfrac{b_3}{a_3 + \cdots}}}$$

where a_n and b_n are either rational numbers, real numbers, or complex numbers. If $b_n = 1$ for all n the expression is called a simple continued fraction. If the expression has a finite amount of terms, it is called a finite continued fraction. Similarly, if the expression has an infinite number of terms, it is called an infinite continued fraction.

e as a continued fraction:

$$e = 2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{2}{3 + \cfrac{3}{4 + \cdots}}}}$$

π as a continued fraction:

$$\pi = \cfrac{4}{1 + \cfrac{1^2}{3 + \cfrac{2^2}{5 + \cfrac{3^2}{7 + \cfrac{4^2}{9 + \cdot s}}}}}$$

$\sqrt{2}$ as a continued fraction:

$$\sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cdots}}}}$$

Python Code For Approximating e:

Below is the code I wrote to approximate e:

```
n = input("How many decimals of e would you like to approximate?")

sum = 0
desired_e = N(e, digits = n + 1)
term_number = 0

while sum != desired_e:
sum += 1/factorial(term_number)
term_number += 1

print "NOTE: The code will approximate to the " + str(n) + " digits you
requested, but it will show " + str(n+1) + " digits to prevent rounding errors."
print "------------------------------------------------------------"
print "Estimated value of e: " + str(N(sum, digits = n + 1))
print "------------------------------------------------------------"
print "Actual value of e:      " + str(desired_e)
print "------------------------------------------------------------"
print "Difference:              " + str((N(desired_e - sum, digits = 2)))
print "------------------------------------------------------------"
```

Listing 1: Estimator for e

Explanation:

The code will ask how many digits of e you would like to approximate and store it in variable n. The variable "sum" has an initial value of 0. While "sum" is not equal to the actual value of e ("desired_e"), the series expansion for e (shown below) will be continuously added to "sum", increasing the "term_number" (k) by 1 integer each time until "sum" does equal "desired_e". When "sum" does equal "desired_e", the code will exit the While loop. The code will then print the value of e estimated with the variable "sum" with $n+1$ digits. Underneath that, the code will print the actual value of e (stored as a constant by SageMathCloud) with the variable "desired_e" with $n+1$ digits. The code will then calculate the difference between the estimated and actual value of e and print that number. NOTE: The difference should always be zero if the code is properly functioning.

Series Expansion Used for e:

$$e = \sum_{k=0}^{\infty} \frac{1}{k!}$$

Python Code For Approximating $\sqrt{2}$:

Below is the code I wrote to approximate $\sqrt{2}$:

```
n_2 = input("How many decimals of sqrt(2) would you like to approximate?")

sum_2 = 0
desired_2 = N(sqrt(2), digits = n_2 + 1)
term_number_2 = 0

while sum_2 != desired_2:
sum_2 += (factorial(2*(term_number_2)+1))/((2^(3*(term_number_2)+1))*(factorial(
term_number_2))^2)
term_number_2 += 1

print "NOTE: The code will approximate to the " + str(n_2) + " digits you
requested, but it will show " + str(n_2+1) + " digits to prevent rounding
errors."
print "--------------------------------------------------------------------"
print "Estimated value of 2: " + str(N(sum_2, digits = n_2 + 1))
print "--------------------------------------------------------------------"
print "Actual value of 2:    " + str(desired_2)
print "--------------------------------------------------------------------"
print "Difference:           " + str((N(desired_2 - sum_2, digits = 2)))
print "--------------------------------------------------------------------"
```

Listing 2: Estimator For $\sqrt{2}$

Explanation:

The code for approximating $\sqrt{2}$ is essentially the same as the code for approximating e. The only difference is in the variables and the series expansions.

Series Expansion Used for $\sqrt{2}$:

$$\sqrt{2} = \sum_{k=0}^{\infty} \frac{(2k+1)!}{2^{3k+1} \left(k!\right)^2}$$

Python Code For Approximating π

Below is the code I wrote to approximate π:

```
n_pi = input("How many decimals of pi would you like to approximate ?")

sum_pi = 0
desired_pi = N(pi, digits = n_pi + 1)
term_number_pi = 0

while sum_pi != desired_pi:
sum_pi += (1/(16^term_number_pi))*((4/(8*(term_number_pi)+1)) -(2/(8*(term_number
_pi)+4))-(1/(8*(term_number_pi)+5))-(1/(8*(term_number_pi)+6)))
term_number_pi += 1

print "NOTE: The code will approximate to the " + str(n_pi) + " digits you
requested, but it will show " + str(n_pi+1) + " digits to prevent rounding
errors."
print "————————————————————————————————————————————"
print "Estimated value of pi: " + str(N(sum_pi, digits = n_pi + 1))
print "————————————————————————————————————————————"
print "Actual value of pi:    " + str(desired_pi)
print "————————————————————————————————————————————"
print "Difference:            " + str((N(desired_pi - sum_pi, digits = 2)))
print "————————————————————————————————————————————"
```

Listing 3: Estimator For π

Explanation:

The code for approximating π is essentially the same as the code for approximating e and $\sqrt{2}$. The only difference is in the variables and the series expansions.

Series Expansion for π (Bailey–Borwein–Plouffe Formula):

$$\pi = \sum_{k=0}^{\infty} \left[\frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right]$$

Interestingly enough, the code for approximating π gave me the most trouble. The series expansions that I had previously used in the code either did not converge fast enough or somehow made the variable "sum_pi" infinitely locked in the While loop. After some research and trial and error, I discovered the Bailey–Borwein–Plouffe Formula and decided to try it. It worked perfectly.

Conclusion

Overall, this has been my favorite project. The code was fairly challenging and I initially ran into a couple of issues, but this was the first project where I figured the code out on my own. While I went to Mr. Abell when technical issues arose, the basic structure of the code was my own. In addition, this is the first project that I started well in advance of the due date. I usually try to figure the project out the day it is assigned, but then I put it off until the last few days. This time, I had finished the code a week or two before the due date, making it possible for me to write the report without any stress.

I have really enjoyed brushing up my Python skills and learning LaTeX this year, and I hope I can continue to use these tools in the future, whether it be for school or just for fun.