CS 3430: S24: Scientific Computing
Assignment 08
Approximating the Transcendental Numbers π and e

Vladimir Kulyukin
Department of Computer Science
Utah State University

March 17, 2024

Learning Objectives

1. Continued Fractions
2. Chudnovsky Algorithm

Introduction

In this assignment, we will use two continued fractions to approximate the transcendental numbers π and e and to implement the Chudnovsky algorithm to approximate π. You will code your solutions to Problem 1 in `ecf.py` and `picf.py`, where `cf` abbreviates *continued fraction*, and to Problem 2 in `cpi.py`, where `cpi` abbreviates *Chudnovsky pi*. Included in the zip for this assignment is `cs3430_s24_hw08_uts.py` with my unit tests. My usual advice is to proceed one unit test at a time as you work on the assignment.

Optional Problem 0: (0 points)

You may want to review your class notes and/or the slides for Lectures 16 and 17 and become comfortable with continued fractions, binary splitting, and the Chudnovsky algorithm.

Problem 1: (2 points)

The module `rat.py` contains my simple implementation of a class to manipulate ratios (henceforth rats), i.e., numbers whose numerators and denominators are integers and denominators are not 0. A continued fraction (c-frac for short) is an infinite fraction that looks like this.

$$\cfrac{N_1}{D_1 + \cfrac{N_2}{D_2 + \cfrac{N_3}{D_3 + \cfrac{N_4}{D_4 + \cfrac{N_5}{D_5 + \cfrac{N_5}{D_6 + \cfrac{N_6}{D_7 + \cfrac{N_7}{D_8 + \cdots}}}}}}}}$$

In the above formula, N abbreviates *numerator* and D abbreviates *denominator*. Note that the numerators and denominators come in pairs, i.e., (N_1, D_1), (N_2, D_2), etc. The numerators $N_1, N_2, \ldots, N_i, \ldots$ and the paired denominators $D_1, D_2, \ldots, D_i, \ldots)$ are typically computed by different functions: one for N_i and the other for D_i.

Another common c-frac form is this.

$$z + \cfrac{N_1}{D_1 + \cfrac{N_2}{D_2 + \cfrac{N_3}{D_3 + \cfrac{N_4}{D_4 + \cfrac{N_5}{D_5 + \cfrac{N_5}{D_6 + \cfrac{N_6}{D_7 + \cfrac{N_7}{D_8 + \cdots}}}}}}}},$$

where z is some whole number, typically a positive integer. E.g., below are two common c-fracs for e and π we discussed in Lecture 16.

$$\mathrm{e} = 2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{4 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{6 + \cdots}}}}}}}}$$

$$\pi = 3 + \cfrac{1^2}{6 + \cfrac{3^2}{6 + \cfrac{5^2}{6 + \cfrac{7^2}{6 + \cfrac{9^2}{6 + \cfrac{11^2}{6 + \cfrac{13^2}{6 + \cfrac{15^2}{6 + \cdots}}}}}}}}$$

Let me introduce some terminology now. We will refer to the number z of a c-frac as the c-frac's whole number and to the infinite rat which is added to z as the c-frac's rat.

To approximate a number with a c-frac, we have to compute some N_i and D_i and then add them to the c-frac's whole number z. Number theory findings indicate that the more paired numerators and denominators of a c-frac we compute, the closer we get to the true value of the number being

approximated by the c-frac. Below are four different approximations where we compute 1, 2, 3, and 4 pairs of numerators and denominators.

$$z + \cfrac{N_1}{D_1}$$

$$z + \cfrac{N_1}{D_1 + \cfrac{N_2}{D_2}}$$

$$z + \cfrac{N_1}{D_1 + \cfrac{N_2}{D_2 + \cfrac{N_3}{D_3}}}$$

$$z + \cfrac{N_1}{D_1 + \cfrac{N_2}{D_2 + \cfrac{N_3}{D_3 + \frac{N_4}{D_4}}}}$$

Of course, we can compute 0 pairs, in which case our approximation boils down to the c-frac's whole number z. Below are the first four approximations of π with 0, 1, 2, and 3 numerator-denominator pairs.

$$\begin{aligned}
\pi &\approx 3; \\
\pi &\approx 3 + \tfrac{1^2}{6}; \\
\pi &\approx 3 + \cfrac{1^2}{6 + \frac{3^2}{6}}; \\
\pi &\approx 3 + \cfrac{1^2}{6 + \cfrac{3^2}{6 + \frac{5^2}{6}}}.
\end{aligned}$$

The classes `ecf` and `picf` in the files `ecf.py` and `picf.py` have the methods `N_i(i)` and `D_i(i)` that you will implement. These methods return the i-th numerator and denominator of the c-frac's rat. E.g., below are the first 3 pairs of the π c-frac (cf. Slide 7, Lecture 16).

```
>>> from picf import picf
>>> from rat import rat
>>> [(picf.N_i(i), picf.D_i(i)) for i in range(1, 4)]
[(1, 6), (9, 6), (25, 6)]
```

You will also implement the method `cf_rat(i)` that computes the i-th rat of a c-frac. E.g., below are the first 3 rats of the c-frac for π.

```
>>> str(picf.cf_rat(1))
'1/6'
>>> str(picf.cf_rat(2))
'2/15'
>>> str(picf.cf_rat(3))
'61/420'
>>> r1 = picf.cf_rat(1)
```

```
>>> isinstance(r1, rat)
True
>>> r1.get_n() ## get the rat's numerator
1
>>> r1.get_d() ## get the rat's denominator
6
>>> [str(picf.cf_rat(i)) for i in range(1, 4)]
['1/6', '2/15', '61/420']
```

Once we know how to compute rats of a c-frac, we have 2 choices to approximate the real number approximated by the c-frac. First, we can compute the i-th rat and add it to the c-frac's whole number z and return the answer as a rat. Second, we can compute the rat and turn it into a real by doing the actual float division. The first choice is more numerically stable because we are dealing with whole numbers, but is inappropriate if we need precision. The first choice is what the methods `arx_rat(i)` in `ecf.py` and `picf.py` should be implemented to do, i.e., they approximate (arx) a given real number with the i-th c-frac by computing it as a rat. We can capture a few π rats and linearize them into strings.

```
>>> pi_rat1 = picf.arx_rat(1)
>>> str(pi_rat1)
'19/6'
>>> pi_rat1.get_n()
19
>>> pi_rat1.get_d()
6
>>> pi_rat2 = picf.arx_rat(2)
>>> str(pi_rat2)
'47/15'
>>> pi_rat3 = picf.arx_rat(3)
>>> str(pi_rat3)
'1321/420'
>>> pi_rat13 = picf.arx_rat(13)
>>> str(pi_rat13)
'31539920369/10039179150'
```

The second choice is what the methods `arx_real(i)` in `ecf.py` and `picf.py` should be implemented to do, i.e., they compute the i-th rat and turn it into a real with the float point division.

```
>>> picf.arx_real(1)
Decimal('3.1666666666666666667')
>>> 19/6
3.1666666666666665
>>> picf.arx_real(2)
Decimal('3.1333333333333333333')
>>> 47/15
3.1333333333333333
>>> picf.arx_real(3)
Decimal('3.1452380952380952381')
>>> 1321/420
3.145238095238095
>>> picf.arx_real(13)
Decimal('3.1416831892077550982')
>>> 1321/420
3.145238095238095
```

Let us hunt for some e rats and turn them into reals.

```
>>> from ecf import ecf
>>> [str(ecf.cf_rat(i)) for i in range(1, 4)]
['1/1', '2/3', '3/4']
>>> [str(ecf.arx_rat(i)) for i in range(1, 4)]
['3/1', '8/3', '11/4']
>>> [str(ecf.arx_real(i)) for i in range(1, 4)]
['3', '2.6666666666666667', '2.75']
>>> str(ecf.arx_rat(13))
'49171/18089'
>>> ecf.arx_real(13)
Decimal('2.7182818287356957267')
>>> 49171/18089
2.718281828735696
```

Problem 2: (3 points)

Implement the Chudnovsky algorithm whose pseudocode is given on Slides 24 and 25 in Lecture 17. The file `pi.txt` contains the pi approximation with a manitissa of almost 100,000 digits. We load it into a string and get rid of spaces and new line characters.

```
>>> inf = open('pi.txt', 'r')
>>> pi_str = ''.join(i for i in inf.readline() if i != ' ').strip()
>>> len(pi_str)
100001
>>> len(pi_str[2:])
99999
>>>
```

Now we compute the Chudnovsky π approximation on the interval $[1, 30]$ with a precision of 105 and compare it with the `pi_str`.

```
>>> chud_pi = cpi.arx_real(30, prec=105)
>>> chud_pi
Decimal('3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798215')
>>> str(chud_pi)[:105] == pi_str[:105]
True
>>> str(chud_pi)[:106] == pi_str[:106]
False
>>> len(str(chud_pi))
106
>>> pi_str[:106]
'3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214'
```

It looks like we are off by only one (one!) last digit, i.e., the first 105 digits of the mantissas coincide. We increase the precision to 1001 and approximate π again on [1,30].

```
>>> chud_pi = cpi.arx_real(30, prec=1001)
>>> len(str(chud_pi))
1002
>>> chud_pi_str = str(chud_pi)
>>> for i in range(1, len(chud_pi_str)):
...     if chud_pi_str[:i] != pi_str[:i]:
...         print(i)
...         break
427
>>> str(chud_pi)[:426] == pi_str[:426]
True
```

This time we got 426 digits of the mantissa correctly. The zip contains the file `e.txt` with the first 100,000 digits of the e mantissa. You can experiment with it if you want. Here is how I computed the first 100 digts of the mantissa with my implementation of the c-frac for e on Slide 6 in Lecture 16.

```
>>> from ecf import ecf
>>> inf = open('e.txt', 'r')
>>> e_str = ''.join(i for i in inf.readline() if i != ' ').strip()
>>> e_approx = ecf.arx_real(100, prec=101)
>>> str(e_approx)[:100] == e_str[:100]
True
```

What To Submit

Submit your code in `ecf.py`, `picf.py`, and `cpi.py`.

Happy Hacking and Enjoy the Rat Hunt and the Chudnovsky Split!

Acknowledgements

I took the approximations in `pi.txt` and `e.txt` from these sites.

1. www.geom.uiuc.edu/ huberty/math5337/groupe/digits.html;
2. www.math.utah.edu/ pa/math/e.html.