

1 Introduction

Continued fractions and their use in approximating rational numbers from decimal equivalents are discussed on Wikipedia. The original problem is known as a Diophantine approximation and dates from the third century CE.

2 Continued fraction for 2.375

Consider the exact decimal equivalent 2.375. It can be written as a continued fraction as follows:

$$2.375 = 2 + \frac{1}{2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1}}}} \quad (2.1)$$

$$= 2 + \frac{1}{2 + \frac{1}{1 + \frac{1}{\left(\frac{2}{1}\right)}}} \quad (2.2)$$

$$= 2 + \frac{1}{2 + \frac{1}{1 + \frac{1}{2}}} \quad (2.3)$$

$$= 2 + \frac{1}{2 + \frac{1}{\left(\frac{3}{2}\right)}} \quad (2.4)$$

$$= 2 + \frac{1}{2 + \frac{2}{3}} \quad (2.5)$$

$$= 2 + \frac{1}{\left(\frac{8}{3}\right)} \quad (2.6)$$

$$= 2 + \frac{3}{8} \quad (2.7)$$

$$= \frac{19}{8} \quad (2.8)$$

$$= [2; 2, 1, 1, 1] \quad (2.9)$$

Exact decimal equivalents for rational numbers, including repeating decimals, can be expressed as continued fractions with a finite number of denominators. Irrational numbers have decimal equivalents that can be expressed as continued fractions with an infinite number of denominators.

Given an infinite continued fraction $[a_0; a_1, a_2, \dots]$ the rational numbers

$$\frac{h_n}{k_n} = [a_0; a_1, a_2, \dots, a_n] \quad (2.10)$$

are called its *convergents*. The n th convergent is always the best rational-number approximation to the infinite continued fraction which has a denominator $\leq k_n$.

3 Continued fraction for π

When calculating the convergents for an irrational number like π , each successive convergent h_n and k_n is calculated as a function of a_n (the n th coefficient)¹, d_n (the n th residual denominator), and h_{n-2} , k_{n-2} , h_{n-1} , and k_{n-1} (the previous two convergents) so that:

$$a_n = \lfloor d_n \rfloor \quad (3.1)$$

$$\frac{h_n}{k_n} = \frac{a_n h_{n-1} + h_{n-2}}{a_n k_{n-1} + k_{n-2}} \quad h_{-1} = k_{-2} = 1, h_{-2} = k_{-1} = 0 \quad (3.2)$$

$$d_{n+1} = \frac{1}{d_n - a_n} \quad (3.3)$$

The example for π is as follows:

$$\pi \approx 3.14159 \quad (3.4)$$

$$\approx [3.14159] + \dots \approx 3 = \frac{3}{1} = \frac{h_0}{k_0} \quad (3.5)$$

¹The floor of the reciprocal of the fractional part of the previous residual denominator.

$$\approx 3 + \frac{1}{\left[\frac{1}{0.14159} \right] + \dots} \quad \approx 3 + \frac{1}{7} \quad = \frac{22}{7} \quad = \frac{h_1}{k_1} \quad (3.6)$$

$$\approx 3 + \frac{1}{7 + \frac{1}{\left[\frac{1}{0.06251} \right] + \dots}} \quad \approx 3 + \frac{1}{7 + \frac{1}{15}} \quad = \frac{333}{106} \quad = \frac{h_2}{k_2} \quad (3.7)$$

$$\approx 3 + \frac{1}{7 + \frac{1}{15 + \frac{1}{\left[\frac{1}{0.99659} \right] + \dots}}} \quad \approx 3 + \frac{1}{7 + \frac{1}{15 + \frac{1}{1}}} \quad = \frac{355}{113} \quad = \frac{h_3}{k_3} \quad (3.8)$$

$$\approx 3 + \frac{1}{7 + \frac{1}{15 + \frac{1}{1 + \frac{1}{\left[\frac{1}{0.00342} \right] + \dots}}}} \quad \approx 3 + \frac{1}{7 + \frac{1}{15 + \frac{1}{1 + \frac{1}{292}}}} \quad = \frac{103,993}{33,102} \quad = \frac{h_4}{k_4} \quad (3.9)$$

$$= [3; 7, 15, 1, 292, \dots] \quad (3.10)$$

4 Rational approximations of π

The continued fraction convergents for any number can be calculated recursively based on Theorem 1. As an example, the convergents for π can be calculated as follows:

$$\pi = [3; 7, 15, 1, 292, \dots] \quad (4.1)$$

$$\approx \frac{3 \times 1 + 0}{3 \times 0 + 1} = \frac{3}{1} = \frac{h_0}{k_0} \quad (4.2)$$

$$\approx \frac{7 \times 3 + 1}{7 \times 1 + 0} = \frac{22}{7} = \frac{h_1}{k_1} \quad (4.3)$$

$$\approx \frac{15 \times 22 + 3}{15 \times 7 + 1} = \frac{333}{106} = \frac{h_2}{k_2} \quad (4.4)$$

$$\approx \frac{1 \times 333 + 22}{1 \times 106 + 7} = \frac{355}{113} = \frac{h_3}{k_3} \quad (4.5)$$

$$\approx \frac{292 \times 355 + 333}{292 \times 113 + 106} = \frac{103,993}{33,102} = \frac{h_4}{k_4} \quad (4.6)$$

5 Assignment

The assignment (available as a Codecheck.io) is to:

- write a method `fromDouble` that calculates a continued fraction rational convergent for a value `x`;
- write a constructor that takes a double parameter and initializes the numerator and denominator to those of the convergent calculated by `fromDouble`.

It may be best to use a recursive solution, where the base cases for `fromDouble` are:

- ▶ no more than a specific number of convergents `N` must be calculated; or
- ▶ the convergent denominator must be less than a specific value `LIM`; or
- ▶ $h / k / x$ must differ from 1 by no more than a specific value `EPS`.

A *stub* version of `fromDouble` is below in Listing 1. This version takes a parameter `steps` and uses it to calculate the values for the base-case parameters `N`, `LIM`, and `EPS`. You are not limited to that approach and can calculate the base cases however you see fit.

Possible initial values for `fromDouble` used in a `Fraction` constructor (where `x` is the double to be approximated) are: `fromDouble(x, x, 0, 1, 0, 0, 1, 22, false)`².

```

1  /**
2   * Return continued fraction rational convergent for x ( h, k ) such that:
3   * n >= N - 1, or k is > LIM, or h / k / x differs from 1 by less than EPS.
4   * @link http://en.wikipedia.org/wiki/Continued_fraction#Some_useful_theorems
5   * @param x original double to be approximated
6   * @param d current continued fraction denominator
7   * @param n number of iterations
8   * @param h1 last numerator
9   * @param h2 second-to-last numerator
10  * @param k1 last denominator
11  * @param k2 second-to-last denominator
12  * @param steps parameter used to set LIM, N, and EPS (on [2, 62])
13  * @param v verbose: print intermediate results, if verbose
14  * @return continued fraction rational convergent for x
15  */
16  private static Fraction fromDouble(double x, double d, int n,
17      int h1, int h2, int k1, int k2, int steps, boolean v) {
18      final int LIM = 1 << steps;
19      final int N = steps;
20      final double EPS = 1.0 / (1L << steps);
21  
```

²These initial values are assumed for the Codecheck.io `FractionTest` data.

```

22         // YOUR CODE HERE
23     }

```

Listing 1: fromDouble method

Appendix

This is the original student file Fraction.java. It does not include implementations of all the methods listed in the FractionI.java interface.

```

1  /*
2  * The Fraction class based on http://skylit.com/javamethods3/studentfiles.zip
3  * where the following are also implemented:
4  * - all methods of FractionI;
5  *
6  * JM3e Chapter 10.3 - Author: Alex
7  *
8  //HIDE
9  * @author David C. Petty // http://j.mp/psb_david_petty
10 //EDIT * @author YOUR NAME <your@email.address>
11 */
12
13 public class Fraction implements FractionI
14 {
15     //////////////////////////////////// FIELDS ////////////////////////////////////
16
17     private int num;
18     private int den;
19
20     //////////////////////////////////// CONSTRUCTORS ////////////////////////////////////
21
22     public Fraction() {                // no-args constructor
23         num = 0;
24         den = 1;
25     }
26
27     public Fraction(int n) {
28         num = n;
29         den = 1;
30     }
31
32     public Fraction(int num, int den) {
33         if (den == 0)
34             throw new IllegalArgumentException(
35                 "Fraction_construction_error:_denominator_is_0");
36         // Otherwise... initialize fields and reduce to canonical form
37         this.num = num;
38         this.den = den;
39         this.reduce();
40     }
41

```

```

42 // Copy constructor
43 public Fraction(Fraction other) {
44     this.num = other.num;
45     this.den = other.den;
46 }
47
48 ////////////////////////////////////// METHODS //////////////////////////////////////
49
50 // Accessor methods
51 public int getNumerator() { return num; }
52 public int getDenominator() { return den; }
53
54 // Returns the value of this fraction as a double
55 public double doubleValue() {
56     return (double) num / (double) den;
57 }
58
59 // Returns a string representation of this fraction
60 @Override
61 public String toString() {
62     return num + "/" + den;
63 }
64
65 // Returns the sum of this fraction and other
66 public Fraction add(Fraction other) {
67     int gcd = gcd(den, other.den);
68     // Divide first to reduce overflow.
69     int newNum = other.den / gcd * num + den / gcd * other.num;
70     int newDenom = den / gcd * other.den;
71     return new Fraction(newNum, newDenom);
72 }
73
74 // Returns the sum of this fraction and m
75 public Fraction add(int m) {
76     return add(new Fraction(m));
77 }
78
79 // Returns the product of this fraction and other
80 public Fraction multiply(Fraction other) {
81     int gcd1 = gcd(num, other.den), gcd2 = gcd(other.num, den);
82     // Divide first to reduce overflow.
83     int newNum = (num / gcd1) * (other.num / gcd2);
84     int newDenom = (den / gcd2) * (other.den / gcd1);
85     return new Fraction(newNum, newDenom);
86 }
87
88 // Returns the product of this fraction and m
89 public Fraction multiply(int m) {
90     // return new Fraction(num * m, den);
91     return multiply(new Fraction(m));
92 }
93
94 ////////////////////////////////////// PRIVATE METHODS //////////////////////////////////////
95
96 // Reduce this fraction to canonical form: gcd(num, den) == 1 and den > 0

```

```

97     private void reduce() {
98         if (num == 0) {
99             den = 1;
100            return;
101        }
102
103        if (den < 0) {
104            num = -num;
105            den = -den;
106        }
107
108        int q = gcd(num, den);
109        num /= q;
110        den /= q;
111    }
112
113    // Returns the greatest common divisor of two integers
114    private static int gcd(int n, int d) {
115        if (d == 0) return Math.abs(n); // Math.abs allows negative arguments
116        return gcd(d, n % d);
117    }
118 }

```

Listing 2: Fraction.java

This is the FractionI.java interface file, including all the methods to be implemented in Fraction.java.

```

1  /*
2   * FractionI.java
3   *
4   * Interface for Fraction.
5   *
6   * @author David C. Petty // http://j.mp/psb_david_petty
7   */
8
9  public interface FractionI
10 {
11     // public instance methods
12
13     int getNumerator();
14     int getDenominator();
15     double doubleValue();
16     String toString();
17
18     Fraction add(Fraction f);
19     Fraction add(int m);
20     Fraction multiply(Fraction f);
21     Fraction multiply(int m);
22
23     Fraction negate();
24     Fraction subtract(Fraction f);
25     Fraction subtract(int m);

```

```
26     Fraction reciprocal();
27     Fraction divide(Fraction f);
28     Fraction divide(int m);
29
30     public boolean equals(Fraction other);
31 }
```

Listing 3: FractionI.java