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:

$$2.375 = 2+\cfrac{1}{2+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1}}}} \tag{2.1}$$

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

$$= 2+\cfrac{1}{2+\cfrac{1}{1+\cfrac{1}{2}}} \tag{2.3}$$

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

$$= 2+\cfrac{1}{2+\cfrac{2}{3}} \tag{2.5}$$

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

$$= 2 + \frac{3}{8} \tag{2.7}$$

$$= \frac{19}{8} \tag{2.8}$$

$$= [2; 2, 1, 1, 1] \tag{2.9}$$

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 $[a_0; a_1, a_2, ...]$ the rational numbers

$$\frac{h_n}{k_n} = [a_0; a_1, a_2, ..., a_n] \tag{2.10}$$

are called its *convergents*. The nth 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 nth coefficient)[1], d_n (the nth residual denominator), and h_{n-2}, k_{n-2}, h_{n-1}, and k_{n-1} (the previous two convergents) so that:

$$a_n = \lfloor d_n \rfloor \tag{3.1}$$

$$\frac{h_n}{k_n} = \frac{a_n h_{n-1} + h_{n-2}}{a_n k_{n-1} + k_{n-2}} \qquad h_{-1} = k_{-2} = 1, h_{-2} = k_{-1} = 0 \tag{3.2}$$

$$d_{n+1} = \frac{1}{d_n - a_n} \tag{3.3}$$

The example for π is as follows:

$$\pi \approx 3.14159 \tag{3.4}$$

$$\approx \lfloor 3.14159 \rfloor + ... \qquad \approx 3 \qquad = \frac{3}{1} \qquad = \frac{h_0}{k_0} \tag{3.5}$$

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

$$\approx 3+\cfrac{1}{\left\lfloor\frac{1}{0.14159}\right\rfloor+\ldots} \qquad \approx 3+\frac{1}{7} \qquad = \frac{22}{7} \qquad = \frac{h_1}{k_1} \tag{3.6}$$

$$\approx 3+\cfrac{1}{7+\cfrac{1}{\left\lfloor\frac{1}{0.06251}\right\rfloor+\ldots}} \qquad \approx 3+\cfrac{1}{7+\cfrac{1}{15}} \qquad = \frac{333}{106} \qquad = \frac{h_2}{k_2} \tag{3.7}$$

$$\approx 3+\cfrac{1}{7+\cfrac{1}{15+\cfrac{1}{\left\lfloor\frac{1}{0.99659}\right\rfloor+\ldots}}} \qquad \approx 3+\cfrac{1}{7+\cfrac{1}{15+\cfrac{1}{1}}} \qquad = \frac{355}{113} \qquad = \frac{h_3}{k_3} \tag{3.8}$$

$$\approx 3+\cfrac{1}{7+\cfrac{1}{15+\cfrac{1}{1+\cfrac{1}{\left\lfloor\frac{1}{0.00342}\right\rfloor+\ldots}}}} \qquad \approx 3+\cfrac{1}{7+\cfrac{1}{15+\cfrac{1}{1+\cfrac{1}{292}}}} \qquad = \frac{103,993}{33,102} \qquad = \frac{h_4}{k_4} \tag{3.9}$$

$$= [3; 7, 15, 1, 292, \ldots] \tag{3.10}$$

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:

$$\pi = [3; 7, 15, 1, 292, \ldots] \tag{4.1}$$

$$\approx \frac{3\times 1+0}{3\times 0+1} \qquad = \frac{3}{1} \qquad = \frac{h_0}{k_0} \tag{4.2}$$

$$\approx \frac{7\times 3+1}{7\times 1+0} \qquad = \frac{22}{7} \qquad = \frac{h_1}{k_1} \tag{4.3}$$

$$\approx \frac{15\times 22+3}{15\times 7+1} \qquad = \frac{333}{106} \qquad = \frac{h_2}{k_2} \tag{4.4}$$

$$\approx \frac{1\times 333+22}{1\times 106+7} \qquad = \frac{355}{113} \qquad = \frac{h_3}{k_3} \tag{4.5}$$

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

5 Assignment

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

a) write a method `fromDouble` that calculates a continued fraction rational convergent for a value `x`;
b) 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].

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

```

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

```
        // YOUR CODE HERE
    }
```

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.

```
/*
 * The Fraction class based on http://skylit.com/javamethods3/studentfiles.zip
 * where the following are also implemented:
 * - all methods of FractionI;
 *
 * JM3e Chapter 10.3 - Author: Alex
 *
//HIDE
 * @author David C. Petty // http://j.mp/psb_david_petty
 //EDIT * @author YOUR NAME <your@email.address>
 */

public class Fraction implements FractionI
{
    /////////////////////////////// FIELDS ///////////////////////////////

    private int num;
    private int den;

    //////////////////////////// CONSTRUCTORS ////////////////////////////

    public Fraction() {             // no-args constructor
        num = 0;
        den = 1;
    }

    public Fraction(int n) {
        num = n;
        den = 1;
    }

    public Fraction(int num, int den) {
        if (den == 0)
            throw new IllegalArgumentException(
                "Fraction␣construction␣error:␣denominator␣is␣0");
        // Otherwise... initialize fields and reduce to canonical form
        this.num = num;
        this.den = den;
        this.reduce();
    }

```

```java
    // Copy constructor
    public Fraction(Fraction other) {
        this.num = other.num;
        this.den = other.den;
    }

    ///////////////////////////// METHODS //////////////////////////////

    // Accessor methods
    public int getNumerator() { return num; }
    public int getDenominator() { return den; }

    // Returns the value of this fraction as a double
    public double doubleValue() {
        return (double) num / (double) den;
    }

    // Returns a string representation of this fraction
    @Override
    public String toString() {
        return num + "/" + den;
    }

    // Returns the sum of this fraction and other
    public Fraction add(Fraction other) {
        int gcd = gcd(den, other.den);
        // Divide first to reduce overflow.
        int newNum = other.den / gcd * num + den / gcd * other.num;
        int newDenom = den / gcd * other.den;
        return new Fraction(newNum, newDenom);
    }

    // Returns the sum of this fraction and m
    public Fraction add(int m) {
        return add(new Fraction(m));
    }

    // Returns the product of this fraction and other
    public Fraction multiply(Fraction other) {
        int gcd1 = gcd(num, other.den), gcd2 = gcd(other.num, den);
        // Divide first to reduce overflow.
        int newNum = (num / gcd1) * (other.num / gcd2);
        int newDenom = (den / gcd2) * (other.den / gcd1);
        return new Fraction(newNum, newDenom);
    }

    // Returns the product of this fraction and m
    public Fraction multiply(int m) {
        // return new Fraction(num * m, den);
        return multiply(new Fraction(m));
    }

    ///////////////////////// PRIVATE METHODS //////////////////////////

    // Reduce this fraction to canonical form: gcd(num, den) == 1 and den > 0
```

```
    private void reduce() {
        if (num == 0) {
            den = 1;
            return;
        }

        if (den < 0) {
            num = -num;
            den = -den;
        }

        int q = gcd(num, den);
        num /= q;
        den /= q;
    }

    // Returns the greatest common divisor of two integers
    private static int gcd(int n, int d) {
        if (d == 0) return Math.abs(n); // Math.abs allows negative arguments
        return gcd(d, n % d);
    }
}
```

Listing 2: `Fraction.java`

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

```
/*
 * FractionI.java
 *
 * Interface for Fraction.
 *
 * @author David C. Petty // http://j.mp/psb_david_petty
 */

public interface FractionI
{
    // public instance methods

    int getNumerator();
    int getDenominator();
    double doubleValue();
    String toString();

    Fraction add(Fraction f);
    Fraction add(int m);
    Fraction multiply(Fraction f);
    Fraction multiply(int m);

    Fraction negate();
    Fraction subtract(Fraction f);
    Fraction subtract(int m);
```

```java
    Fraction reciprocal();
    Fraction divide(Fraction f);
    Fraction divide(int m);

    public boolean equals(Fraction other);
}
```

Listing 3: `FractionI.java`