Approximating Rational Numbers by Fractions

Michal Forišek

forisek@dcs.fmph.uniba.sk,
Department of Informatics,
Faculty of Mathematics, Physics and Informatics,
Comenius University,
Mlynská dolina, 842 48 Bratislava, Slovakia

Abstract. In this paper we show a polynomial-time algorithm to find the best rational approximation of a given rational number within a given interval. As a special case, we show how to find the best rational number that after evaluating and rounding exactly matches the input number. In both results, "best" means "having the smallest possible denominator".

1 Motivation

Phillip the physicist is doing an elaborate experiment. He precisely notes each numeric result along with the error estimate. Thus, his results may look as follows: $x = 1.4372 \pm 0.001$

Larry the lazy physicist is doing similar experiments. However, he just takes the exact value he gets, rounds it to several decimal places and writes down the result. Example of Larry's result: $x \sim 2.3134$

Cole the computer scientist is well aware of the Occam's Razor principle (in other words, understands what Kolmogorov complexity is). He knows that the simplest answer is often the right one. In the physicists' case, the exact value might very well be a rational number. However, it is not obvious which rational number this might be. Cole's task will be to find the best, simplest one that matches the measured results.

2 Previous results

The sequence of irreducible rational numbers from $[0, 1]$ with denominator not exceeding a given N is known under the name Farey sequence of order N. For example $F_5 = \{0/1, 1/5, 1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 1/1\}$. Several properties of these sequences are known. The one we are most interested in is the following one: Let p/q, p'/q', and p''/q'' be three successive terms in a Farey sequence. Then

$$p'/q' = (p + p'')/(q + q'') \tag{1}$$

See [3] for Farey's original conjecture of this property, and e.g. [6, 1] for a more involved discussion of the history and properties of these sequences.

Note that the sequence F_N can be created from F_{N-1} by inserting those fractions a/N where a and N are relatively prime. There are $\phi(N)$ such fractions, where $\phi(N)$ is Euler's totient function.

The importance of the property (1) lies in the fact that it gives us more insight into how the sequence is altered with increasing N. It tells us exactly where the next elements will appear – or equivalently, what is the next element that will appear between two currently neighboring ones.

When we denote this addition process graphically, we get the Stern-Brocot tree, (see [9, 2]) as depicted in Figure 2. The construction and some properties of the tree can be found in [5].

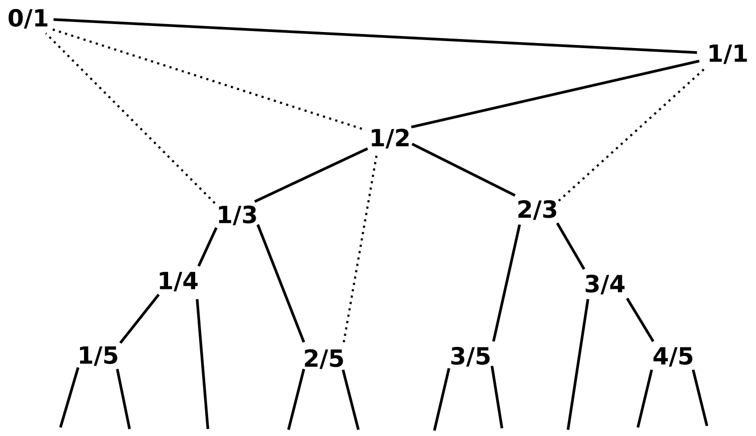

Fig. 1. The Stern-Brocot tree. Full lines show the edges of the tree as commonly defined. For select vertices dotted lines show their other "parent" vertex.

The book [5] also notes the following important fact:

Suppose that $\alpha \in [0, 1)$ is a given real number, and we want to find the best rational approximation of α with denominator not exceeding a given N. Clearly, all we have to do is to find α's place in the sequence F_N, and consider the closest two elements (or one, if α is an element of F_N). This place can be found by an analogy of a "binary search" by descending the Stern-Brocot tree. Each visited vertex of the tree clearly represents either the best lower, or the best upper approximation so far. (Here, "so far" means "with denumerator smaller or equal to the one in the current vertex".)

For example, suppose that $\alpha = 0.39147\ldots$, then the first few vertices visited will be: 1/1, 1/2, 1/3, 2/5, 3/8, 5/13, 7/18, ...

This gives us our first algorithm to find a good rational approximation to a given value α: Walk down the Stern-Brocot tree until the difference between α and the value in the current vertex is close enough to be acceptable. At this moment, you can be sure that no rational number with a smaller denominator gives a better approximation.

Sadly, this algorithm is far from being polynomial in the input size. As a trivial counterexample, consider $\alpha = 10^{-47}$. The path to α is long and boring, as it contains all fractions $1/x$ for $x \leq 10^{47}$. However, these difficulties can be overcome, and our polynomial-time algorithm is derived from this naive algorithm using two improvements.

Another related tool used to find good rational approximations are continued fractions. A simple continued fraction is a fraction of the form:

$$\alpha = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}} \tag{2}$$

To save space, continued fractions are commonly written as $[a_0, a_1, a_2, a_3, \ldots]$. Every rational number has a finite simple continued fraction (or two of them, depending on the exact definition of terminating). Every irrational number can be represented as an infinite continued fraction.

By truncating the continued fraction of a real number α after a finite number of steps we obtain a rational approximation of α. More exactly, the number c_n as defined in (3) is called the n-th convergent of α. It can be proved that any convergent of α is a best rational approximation of α in the sense that the convergent is nearer to α than any other fraction whose denominator is less than that of the convergent. Moreover, more exact bounds on how good this approximation is are known, but we won't need them in this article.

$$c_n = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\cdots + \cfrac{1}{a_n}}}} \tag{3}$$

Note that there may be cases when the desired rational approximation of α is not a convergent of α. More exactly, there are fractions p/q such that:

- No fraction with denominator less than q is closer to α than p/q.
- The fraction p/q is not a convergent of α.

As a simple example, suppose that $\alpha = 0.1$. The fraction $1/7$ satisfies the conditions above.

Furthermore, note that $1/7$ is the first fraction that after evaluating and rounding to one decimal place gives α. In other words, $1/7$ is the best rational approximation of $\alpha = 0.1$ with a given tolerance $d = 0.05$.

3 Problem formulation

The exact problem we are trying to solve can be formulated as follows: given is a rational number $\alpha \in (0,1)$ and a precision d. (Both numbers are given in decimal notation, with N digits each.) The goal is to find positive integers p, q such that p/q lies within the interval $(\alpha - d, \alpha + d)$, and q is minimal.

Several notes on this definition:

We opted to limit the problem to $\alpha \in (0,1)$ instead of the more general task $\alpha \in \mathbb{R}$. However, note that the general problem is easily reduced to our definition: For $\alpha = 0$ the best answer is always $0/1$. Any $\alpha \in \mathbb{R} \setminus [0,1)$ can be written as $\lfloor \alpha \rfloor + \alpha'$, where $\alpha' \in [0,1)$. The best approximation of α can be computed by finding the best approximation of α' and adding the integer $\lfloor \alpha \rfloor$.

Also, by solving this task for the open interval $(\alpha - d, \alpha + d)$, we can easily obtain solutions for half-closed intervals $[\alpha - d, \alpha + d)$, $(\alpha - d, \alpha + d]$ and for the closed interval $[\alpha - d, \alpha + d]$ simply by checking whether the included bounds correspond to a better approximation than the one found.

Theorem 1. *The correct solution of the problem defined above is unique, and* $gcd(p,q) = 1$.

Proof. Both parts will be proved by contradiction. First, suppose that there are two best approximations $A = p_1/q$ and $B = p_2/q$ with the same denominator $q > 1$ and $p_1 < p_2 \leq q$. Consider the fraction $C = p_1/(q-1)$. Clearly we have $C > A$. Moreover, we get:

$$p_2(q-1) = p_2q - p_2 \geq p_2q - q = (p_2 - 1)q \geq p_1q \tag{4}$$

From (4) it follows that $p_2/q \geq p_1/(q-1)$, in other words $B \geq C$. Thus C is also a valid approximation, and it has a smaller denominator.

The coprimeness of p and q in the optimal solution is obvious. Formally, suppose that $gcd(p,q) = g > 1$. Then clearly $p' = p/g$ and $q' = q/g$ is a valid approximation with a smaller denominator. □

4 Mathematica and similar software

The software package Mathematica by Wolfram Research, Inc. claims to include a function `Rationalize` that solves the problem as stated above. (And variations thereof, as Mathematica can work both with exact and with approximate values.) Citing from the documentation [11],

- `Rationalize[x]` takes Real numbers in x that are close to rationals, and converts them to exact Rational numbers.
- `Rationalize[x,dx]` performs the conversion whenever the error made is smaller in magnitude than dx
- `Rationalize[x,dx]` yields the rational number with the smallest denominator that lies within dx of x

However, we discovered that this is not the case. We tried to contact the authors of the software package, but to date we received no reply. In Table 1 we demonstrate a set of inputs where Mathematica 5.2 fails to find the optimal solution. The exact command used was `Rationalize[alpha‘‘200,d‘‘200]`, where the `‘‘200` part tells the software to consider the values as arbitrary-precision values with 200 digits of accuracy. (See [10] for more details.) The counterexamples in the last two lines of Table 1 seem to scale. (I.e., we can get new inputs by increasing the number of 1s in α and the number of 0s in d by the same amount. Mathematica fails on most of these inputs.)

input	**correct**	**Mathematica**
$\alpha = 0.1$, $d = 0.05$	1/7	1/9
$\alpha = 0.12$, $d = 0.005$	2/17	3/25
$\alpha = 0.15$, $d = 0.005$	2/13	3/20
$\alpha = 0.111\,112$, $d = 0.000\,000\,5$	888 890/8 000 009	1 388 889/12 500 000
$\alpha = 0.111\,125$, $d = 0.000\,000\,5$	859/7730	889/8000

Table 1. Example inputs where Mathematica fails to find the optimal approximation.

Furthermore, if the first argument is an exact number, Mathematica leaves it intact, which might be not optimal. E.g., `Rationalize[30/100,4/100]` returns 3/10 instead of the correct output 1/3.

We are not aware of any other software that claims to have this functionality. For example, Matlab does include a function `rat`, but the documentation [8] doesn't guarantee optimality of approximation in our sense – on the contrary, it directly claims that `rat` only approximates by computing a truncated continued fraction expansion.

5 Outline of our algorithm

We will derive the polynomial algorithm in two steps. First, we will show a compressed way of traversing the Stern-Brocot tree. We will show that this approach is related to generating the approximations given by the continued fraction. However, keeping the Stern-Brocot tree in mind will help us in the second step, where we show how to compute the best approximation once we reach a valid approximation using the compressed tree traversal.

Imagine that we start traversing the Stern-Brocot tree in the vertex 1/1. The path downwards can be written as a sequence of ‘L’s and ‘R’s (going left and right, respectively).

E.g., for $\alpha = 0.112$ we get the sequence $LLLLLLLLLRLLLLLLLLLLLL$. This can be shortened to L^9RL^{12}. As noted in [5], there is a direct correspondence between these exponents and the continued fraction representation of α.

As an example, note that

$$0.112 = 0 + \cfrac{1}{8 + \cfrac{1}{1 + \cfrac{1}{13}}} = 0 + \cfrac{1}{8 + \cfrac{1}{1 + \cfrac{1}{12 + \frac{1}{1}}}} \tag{5}$$

For a given path in the Stern-Brocot tree let $(a_0, a_1, a_2, \ldots)$ be the sequence of exponents obtained using the path compression as described above. Our algorithm will visit vertices corresponding to paths L^{a_0}, $L^{a_0}R^{a_1}$, $L^{a_0}R^{a_1}L^{a_2}$, and so on. In each such vertex we will efficiently compute the next value a_i. A detailed description of this computation can be found in Section 6.

For the example above, $\alpha = 0.112$, our algorithm will visit the vertices corresponding to L^9, L^9R, and L^9RL^{12}. These correspond to fractions 1/9, 2/17, and $14/125 = \alpha$.

For comparison note that the subsequent approximations of $\alpha = 0.112$ by truncating the continued fractions are: 0, 1/8, 1/9, (13/116 and)[1] $14/125 = \alpha$.

In Section 7 we will modify the compressed tree traversal algorithm slightly. It can be shown that after the modification the set of examined vertices will be a superset of the convergents, however we omit the proof, as it is not necessary to prove the correctness of our algorithm.

Now, let's assume that the exact input instance is $\alpha = 0.112$ and $d = 0.0006$. For this input, the algorithm as defined above skips the optimal solution $9/80 = 0.1125$.

In Section 7 we will show that the best approximation has to lie on the last compressed branch of the visited path. More generally, it has to lie on the first compressed branch such that its final vertex represents a valid approximation. Also, we will show that this approximation can be computed efficiently.

An example implementation of the algorithm can be found in Section 8

6 Compressed tree traversal

We will describe traversal to the left, i.e., to a smaller fraction than the current one. Traversal to the right works in the same way.

Let p_a/q_a and p_b/q_b be the last two vertices visited, with $p_a/q_a < \alpha < p_b/q_b$. At the beginning of the entire algorithm we are in this situation with $p_a = 0$ and $q_a = p_b = q_b = 1$.

We are now in the vertex corresponding to p_b/q_b, and we are going to make several steps to the left. Using (1) we get that in our situation each step to the left corresponds to changing the current fraction p/q into a new fraction $(p + p_a)/(q + q_a)$.

[1] The presence of this continuent depends on the chosen form of the continued fraction, see equation (5).

Let x be the number of steps performed by the naive algorithm. How to compute the value x without actually simulating the process?

Clearly, x is the smallest such value that the fraction we reach is smaller than or equal to α. We get an inequality:

$$\frac{p_b + xp_a}{q_b + xq_a} \leq \alpha \tag{6}$$

This can be simplified to

$$p_b - \alpha q_b \leq x(\alpha q_a - p_a) \tag{7}$$

We assumed that $p_a/q_a < \alpha$, thus $(\alpha q_a - p_a)$ is positive, and we get

$$x \geq \frac{p_b - \alpha q_b}{\alpha q_a - p_a} \tag{8}$$

As x has to be an integer, and we are interested in the smallest possible x, the correct x can be computed as follows:

$$x = \left\lceil \frac{p_b - \alpha q_b}{\alpha q_a - p_a} \right\rceil \tag{9}$$

7 Locating the best approximation

Theorem 2. *Suppose that while going in one direction the naive algorithm visited exactly the fractions p_0/q_0, …, p_n/q_n. Consider the fractions p_0/q_0, p_{n-1}/q_{n-1} and p_n/q_n. If neither of these three fractions represents a valid approximation (i.e., lies within d of α), then none of the p_i/q_i represent a valid approximation.*

Proof. WLOG let's consider steps to the left. p_n/q_n is the first and only value out of all p_i/q_i such that $p_n/q_n \leq \alpha$. The fact that p_n/q_n is not a valid approximation gives us that in fact $p_n/q_n \leq \alpha - d$. Similarly, $p_{n-1}/q_{n-1} \geq \alpha + d$, and clearly all other p_i/q_i are even greater. □

This gives us a way how to check that our compressed tree traversal algorithm doesn't skip any valid approximations: In each step, after computing the value x from (9), check whether making either $x - 1$ or x steps yields a valid approximation. If not, Theorem 2 tells us that we may make x steps without skipping a valid approximation. If we found a valid approximation, we need to find the smallest one.

Again, for simplicity we will only show the case when the naive algorithm makes steps to the left. In this case, the best approximation simply corresponds to the smallest $k \leq x$ such that the fraction after k steps is a valid approximation – or, equivalently, is less than $\alpha + d$.

We can compute this k in very much the same fashion as when we computed x in the previous section. The exact formula for k for the going-to-left case:

$$k = \left\lfloor \frac{p_b - (\alpha + d)q_b}{(\alpha + d)q_a - p_a} \right\rfloor + 1 \tag{10}$$

8 Implementation

A proof-of-concept implementation of our algorithm in the open-source calculator `bc` follows. All computations are done using arbitrary-precision integers. For this reason, the input is given in the variables `alpha_num`, `d_num`, and `denum`. These variables represent the values α = `alpha_num`/`denum` and d = `d_num`/`denum`. We assume that $0 <$ `d_num` $<$ `alpha_num` $<$ `denum` and that `d_num` $+$ `alpha_num` $<$ `denum`. This exactly covers the cases when $\alpha \in (0, 1)$ and the answer is not 0/1 or 1/1.

Example: The input $\alpha = 0.33456$ and $d = 0.000005$ can be entered as `alpha_num` $= 334560$, `d_num` $= 5$, and `denum` $= 1000000$.

```
# input variables: alpha_num, d_num, denum
#
# we seek the first fraction that falls into the interval
# (alpha-d, alpha+d) =
#   = ( (alpha_num-d_num)/denum, (alpha_num+d_num)/denum )

# fraction comparison: compare (a/b) and (c/d)
define less(a,b,c,d) { return (a*d < b*c); }
define less_or_equal(a,b,c,d) { return (a*d <= b*c); }

# check whether a/b is a valid approximation
define matches(a,b) {
  if (less_or_equal(a,b,alpha_num-d_num,denum)) return 0;
  if (less(a,b,alpha_num+d_num,denum)) return 1;
  return 0;
}

# set initial bounds for the search:
p_a = 0 ; q_a = 1 ; p_b = 1 ; q_b = 1

define find_exact_solution_left(p_a,q_a,p_b,q_b) {
  k_num = denum * p_b - (alpha_num + d_num) * q_b
  k_denum = (alpha_num + d_num) * q_a - denum * p_a
  k = (k_num / k_denum) + 1
  print (p_b + k*p_a)," ",(q_b + k*q_a),"\n";
}
define find_exact_solution_right(p_a,q_a,p_b,q_b) {
  k_num = - denum * p_b + (alpha_num - d_num) * q_b
  k_denum = - (alpha_num - d_num) * q_a + denum * p_a
  k = (k_num / k_denum) + 1
  print (p_b + k*p_a)," ",(q_b + k*q_a),"\n";
}

while (1) {
  # compute the number of steps to the left
  x_num = denum * p_b - alpha_num * q_b
  x_denum = - denum * p_a + alpha_num * q_a
  x = (x_num + x_denum - 1) / x_denum # = ceil(x_num / x_denum)

  # check whether we have a valid approximation
  aa = matches( p_b + x*p_a, q_b + x*q_a )
  bb = matches( p_b + (x-1)*p_a, q_b + (x-1)*q_a )
  if (aa || bb) { cc = find_exact_solution_left(p_a,q_a,p_b,q_b); break; }

  # update the interval
  new_p_a = p_b + (x-1)*p_a ; new_q_a =  q_b + (x-1)*q_a
  new_p_b = p_b + x*p_a     ; new_q_b =  q_b + x*q_a

  p_a = new_p_a ; q_a = new_q_a
  p_b = new_p_b ; q_b = new_q_b

  # compute the number of steps to the right
  x_num = alpha_num * q_b - denum * p_b
  x_denum = - alpha_num * q_a + denum * p_a
  x = (x_num + x_denum - 1) / x_denum # = ceil(x_num / x_denum)

  # check whether we have a valid approximation
  aa = matches( p_b + x*p_a, q_b + x*q_a )
  bb = matches( p_b + (x-1)*p_a, q_b + (x-1)*q_a )
  if (aa || bb) { cc = find_exact_solution_right(p_a,q_a,p_b,q_b); break; }

  # update the interval
  new_p_a = p_b + (x-1)*p_a ; new_q_a =  q_b + (x-1)*q_a
  new_p_b = p_b + x*p_a     ; new_q_b =  q_b + x*q_a

  p_a = new_p_a ; q_a = new_q_a
  p_b = new_p_b ; q_b = new_q_b
}
```

9 Time complexity

We claim that after each pass through the main `while`-loop of our implementation each of the values q_a and q_b at least doubles. To prove this, note the following: After the steps to the left the new two denominators are greater than or equal to q_b and $q_a + q_b$. After the steps to the right the final two denominators are greater than or equal to $q_a + q_b$ and $q_a + 2q_b$.

Now, suppose that the input numbers have at most N digits. The fraction `alpha_num/denum` is clearly a valid approximation. Therefore the optimal solution has a denominator with at most N digits.

After the `while`-loop was executed K times, the denominators of the currently examined fractions are greater than or equal to 2^K. Clearly after $4N$ loops the denominators would exceed N digits in length. Thus the `while`-loop will be executed $O(N)$ times only.

Each execution of the `while`-loop involves a constant number of operations with $O(N)$ digits long integers. The required operations are addition, subtraction, multiplication and division. The first two operations can easily be done in $O(N)$. For multiplication and division our implementation uses the naive $O(N^2)$ algorithm. Thus the running time of our solution is $O(N^3)$ – polynomial in the input size.

Clearly the bottleneck are the algorithms for multiplication and division. However, there are faster algorithms for both operations, for example the FFT-based multiplication algorithm running in $O(N \log N)$, and its corresponding division algorithm that uses multiplication and Newton's method to estimate the reciprocal of the denominator. For a discussion of these algorithms, see [7].

10 Acknowledgements and Notes

This work has been supported in part by the VEGA grant 1/3106/06.

The author would like to thank three anonymous referees, whose remarks helped to improve the readability of the article.

The author used the main result of this article as a task in the programming competition IPSC, see [4].

References

1. Beiler, Albert H.: Farey Tails. Ch. 16 in Recreations in the Theory of Numbers: The Queen of Mathematics Entertains. New York: Dover, 1966.
2. Brocot, A.: Calcul des rouages par approximation, nouvelle methode. Revue Chronométrique 6 (1860), 186–194.
3. Farey, J.: On a Curious Property of Vulgar Fractions. London, Edinburgh and Dublin Phil. Mag. 47, 385, 1816.
4. Michal Forišek: IPSC task "Exact? Approximate!" `http://ipsc.ksp.sk/contests/ipsc2005/real/problems/e.php` [accessed Jan 2007]
5. Graham, R. L., Knuth, D. E., Patashnik, O.: Concrete Mathematics: A Foundation for Computer Science (2nd Edition). Addison-Wesley Professional, 1994.
6. Hardy, G. H., Wright, E. M.: Farey Series and a Theorem of Minkowski. Ch. 3 in An Introduction to the Theory of Numbers, 5th ed. Oxford, England: Clarendon Press, pp. 23-37, 1979.
7. Knuth, D. E.: How Fast Can We Multiply? Sec. 4.3.3 in The Art of Computer Programming II, Addison-Wesley Professional, 1997.
8. The MathWorks, Inc.: MATLAB Function Reference: rat, rats `http://www.mathworks.com/access/helpdesk/help/techdoc/ref/rat.html` [accessed Jan 2007]
9. Stern, M. A.: Ueber eine zahlentheoretische Funktion. Journal für die reine und angewandte Mathematik 55 (1858), 193–220.
10. Wolfram Research, Inc.: Mathematica 5.2 Documentation: Numerical Precision `http://documents.wolfram.com/mathematica/book/section-3.1.4` [accessed Jan 2007]
11. Wolfram Research, Inc.: Mathematica 5.2 Documentation: Rationalize `http://documents.wolfram.com/mathematica/functions/Rationalize` [accessed Jan 2007]