# 1 Problem Statement

It is well-known that $2^{10} = 1024$ is nearly $10^3 = 1000$. What other powers of 2 are surprisingly close to a power of 10? That is, for $2^a \approx 10^b$, what are the $a$ and $b$? To start, find a power of 2 that is a better approximation than $2^{10} \approx 10^3$ where "better" means the relative error is less. (The relative error is $|2^{10} - 10^3|/(10^3) = 0.024$ or 2.4%.)

# 2 Answer

We wish to minimize the relative error. We can then minimize the absolute error, of course. What we want is

$$2^a = 10^b \tag{1}$$

We can rewrite the left hand side as

$$2^a = (10^{\log_{10} 2})^a = 10^{a \log_{10} 2} \tag{2}$$

with $\log_{10}$ meaning the base 10 logarithm. Then we have

$$2^a 10^{-b} = 10^{a \log_{10} 2 - b} = 1 = 10^0 \tag{3}$$

Taking base ten logarithms of both sides we are left with the equation for the exponents of

$$a \log_{10} 2 - b = 0 \tag{4}$$

$$\frac{a}{b} = \frac{1}{\log_{10} 2} \tag{5}$$

$$\tag{6}$$

Now $\log_{10} 2$ is not a rational number, but it is well-known that continued fraction representations give the most accurate representations for equations of the form (4).

To find a continued fraction representation of a number, you change it into the form $[a_0; a_1, a_2, \ldots]$ which means

$$a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \cdots}}} \tag{7}$$

Given a representation of a number, it is actually fairly simple to find the continued fraction representation. Given a number $b$, we first truncate it to its nearest integer. That is $a_0$. Then then we find $b_1 = 1/(b - a_0)$. Truncate this to its nearest integer and that is $a_1$. Then we find $b_2 = 1/(b_1 - a_1)$. Truncate this to the nearest integer and we ahve $a_2$. Continue this process with $b_n = 1/(b_{n-1} - a_{n-1})$ and $a_n = \text{floor}(b_n)$ with the floor function doing the truncation of the integer.

As an example consider $b = 5.43$. Then $a_0 = 5$, $b_1 = 1/(0.43) \approx 2.3256$, $a_1 = 2$, then $b_2 = 1/([1/.43] - 2) \approx 3.07143$ so $a_2 = 3$ and $b_3 = 1/(b_2 - 3) = 14$, and the algorithm ends. The continued fraction is then $[5; 2, 3, 14]$. His forms better and better fraction approximations as

follows

$$[a_0] = 5 \to \frac{|5.43 - 5|}{5.43} \approx 0.079 \tag{8}$$

$$[a_0; a_1] = 5 + \frac{1}{2} = \frac{11}{2} \to \frac{|5.43 - 5.5|}{5.43} \approx 0.013 \tag{9}$$

$$[a_0; a_1, a_2] = 5 + \frac{1}{2 + \frac{1}{3}} = \frac{38}{7} \to \frac{|5.43 - \frac{38}{7}|}{5.43} \approx 0.00026 \tag{10}$$

$$[a_0; a_1, a_2, a_3] = 5 + \frac{1}{2 + \frac{1}{3 + \frac{1}{14}}} = \frac{543}{100} \tag{11}$$

These are called convergents, as they converge to the number we are representing.

It is easy to create an algorithm to do this. Because floating point arithmetic often fails to get the answer exactly correct, it will spuriously continue after a perfect approximation, but, in practice, this is not so difficult to avoid. You can simply check that the denominator is not bigger than the rational approximation you initially feed in (so we would ignore any answers with denominators bigger than 100 for 5.43).

Now we can apply this to $a \equiv \log_{10} 2 = \ln(2)/\ln(10)$. We find the approximation is $[0; 3, 3, 9, 2, 2, 4, 6, 2, 1, \dots]$ and it continues forever. This gives as fractional approximations

$$a_0 = \frac{0}{1} \to \frac{|2^1 - 10^0|}{10^0} \approx 1 \tag{12}$$

$$a_1 = \frac{1}{3} \to \frac{|2^3 - 10^1|}{10^1} \approx 0.2 \tag{13}$$

$$a_2 = \frac{3}{10} \to \frac{|2^{10} - 10^3|}{10^3} \approx 0.024 \tag{14}$$

$$a_3 = \frac{28}{93} \to \frac{|2^{93} - 10^{28}|}{10^{28}} \approx 0.0096 \tag{15}$$

$$a_4 = \frac{59}{196} \to \frac{|2^{196} - 10^{59}|}{10^{59}} \approx 0.0043 \tag{16}$$

$$a_5 = \frac{146}{485} \to \frac{|2^{485} - 10^{146}|}{10^{146}} \approx 0.0010 \tag{17}$$

$$a_6 = \frac{643}{2136} \to \frac{|2^{2136} - 10^{643}|}{10^{643}} \approx 0.00016 \tag{18}$$

$$a_7 = \frac{4004}{13301} \to \frac{|2^{13301} - 10^{4004}|}{10^{4004}} \approx 6.4 \times 10^{-5} \tag{19}$$

$$a_8 = \frac{8651}{28738} \to \frac{|2^{28738} - 10^{8651}|}{10^{8651}} \approx 3.54 \times 10^{-5} \tag{20}$$

$$a_9 = \frac{12655}{42039} \to \frac{|2^{42039} - 10^{12655}|}{10^{12655}} \approx 2.8 \times 10^{-5} \tag{21}$$

And so we see that the next best approximation is $10^{28}$ and $2^{93}$ after $2^{10}$ and $10^3$.

Note that if we were solving (5) we would need to worry about numbers with denominators in-between those listed above. That is, we would have to try changing the $a_i$ in our continued fraction representation to ensure we are getting the best approximations. However (4) removes this difficulty because then the continued fraction representation is basically always the best approximation. I

have checked the other best fraction representations to make sure this is all we need, but you can try it yourself if you are not convinced. Look up semiconvergents to see the methods.

The difference between

$$\frac{a}{b} = ca - bc = 0 \tag{22}$$

comes from what we consider a "best approximation". In the first, we desire our best approximation of $c$ as $a/b$ to satisfy

$$\left|c - \frac{a}{b}\right| \leq \left|c - \frac{a'}{b'}\right| \tag{23}$$

for all rational numbers with $a'$ and $b'$ different than $a$ and $b$ with $0 < b' \leq b$.

In the second case, we desire

$$|bc - a| < |b'c - a'| \tag{24}$$

for all rational numbers with $a'$ and $b'$ different than $a$ and $b$ with $0 < b'leb$.

You may think there is no difference, but suppose we divided through by $b$ in the second case. This would mean

$$\left|c - \frac{a}{b}\right| < \left|\frac{b'c}{b} - \frac{a'}{b}\right| \tag{25}$$

$$\left|c - \frac{a}{b}\right| < \frac{b'}{b}\left|c - \frac{a'}{b'}\right| \tag{26}$$

Remember that $b \geq b'$ so $b'/b \leq 1$ and so these are in fact two different criteria. The $|bc - a|$ criteria is stricter because it is more restrictivesince $b'/b \leq 1$.

The program below calculates continued fraction representations, outputs the convergents, and can evaluate the convergents decimal approximations.

PowerTwoPowerTen.py

```
import numpy as np
# This is for Riddler Express March 13, 2020

# use 2=10^(np.log(2)/np.log(10))
# we then want 2^a/10^b less than 0.25 for integer a and b
# instead we rewrite and want the exponent nearly equal to zero
# so then we want b/a=np.log(2)/np.log(10)
# (or b/a=np.log10(2))
# use a continued fraction representation for closest a and b

# takes in number z and number of terms to compute with size
# outputs continued fraction in standard form as list
# [a_0,a_1,a_2,...,a_size]
def cont_frac(z,size=4):
  #create list for fraction in standard
  # form [a_0;a_1,a_2,...,a_size]
   confac=[]
  #initial truncation
   x=int(z)
   # put truncation in
   confac.append(x)
   # prepare remainder
   y=np.float(z)-x
```

```
24    if y==0:
25      # we finished
26      return confac
27    # otherwise
28    for i in range(size):
29      # invert remainder
30      z=1/y
31      # truncate this
32      x=int(z)
33      # prepare next remainder
34      y=z-x
35      # put truncated part in
36      confac.append(x)
37      if y==0:
38        # we finished
39        return confac
40    return confac
41
42  # takes in continued fraction in standard form
43  # and outputs all the convergents (the fractions)
44  def ratio_cont_frac(confac):
45    # easier to work with reversed continued fraction
46    bconfac=confac[::-1]
47    out=[]
48    size=len(confac)
49    for i in range(size):
50      # write 1/a_i as far1 to start with ith truncated
51      # continued fraction
52      far1=[1,bconfac[i]]
53      for j in range(i,size-1):
54        # add the a_i+1/a_{i+1}
55        far2=[bconfac[j+1]*far1[1]+far1[0],far1[1]]
56        # flip numerator and denominator for next addition
57        far1=far2[::-1]
58      if i==size-1:
59        # in this case there is no adding
60        out.append(far1[::-1])
61      else:
62        out.append(far2)
63      #reverse so worse approximations come first
64    return out[::-1]
65
66  # takes in list of convergents in fraction form
67  # outputs a list of their decimal values
68  def ratioout(outconfac):
69    out=[]
70    for i in outconfac:
71      # spit out decimal form from fraction form
72      out.append(i[0]/np.float(i[1]))
73    return out
74
75  # takes in list of convergents in decimal form
76  # converts into our original problem and
77  # computes the relative error
78  def errorpowertwoten(outconfac):
79    out=[]
80    for i in outconfac:
81      # write out the error in terms of powers of 10
82      out.append(np.abs(10**(np.log(2)/np.log(10)*i[1]-i[0])-1.))
83    return out
```