Recurring Digits Assignment Report

Marek Michalowski

October 28, 2021

1 Introduction to the problem

The assignment deals with the decimal representation of fractions of the form $\frac{p}{q}$ where $p, q \in \mathbb{Z}^+$, $p < q$ and q ends with a digit 9. Specifically, the first two tasks aim to write two MATLAB functions (employing two different algorithms) which find the recurring digits and their period (the length of the repeating digits sequence) in the decimal representation of $\frac{p}{q}$. Later in task 3, these functions are used to find p, q with $q \in \{10i + 9 | i \in [0, 44]\}$ and $0 < p < q$ such that $\frac{p}{q}$ produces the longest recurring digits sequence and then display that sequence as an array and its length as a scalar. If there are multiple such pairs with the same period, then all these pairs are to be displayed, but the sequence of digits only for one chosen pair is to be shown.

2 Brief overview of the theory

2.1 Useful properties of recurring digits

We begin this section by stating some facts for fractions of the form $\frac{1}{q}$ where $q \in \mathbb{Z}^+$ does not contain powers of 2 and 5 among its prime factors (i.e. p is co-prime to 10), based on the book *History of the Theory of Numbers* by L. E. Dickinson. Such fractions always lead to decimal representations with recurring digits starting immediately after the decimal place.[1] More generally, if $q = 2^m 5^n r$ where $r \in \mathbb{Z}^+$ is not a multiple of powers of 2 and 5, then the recurring digits are delayed by $max(m, n)$.[1]

Since in this problem we are considering denominators of the form $10i + 9$, we can in fact guarantee that these denominators will not be divisible by 2 or 5 ($10i + 9 \equiv 1 \pmod 2$ and $10i + 9 \equiv 4 \pmod 5$). Hence, we can use the fact that the recurring digits start immediately after the decimal point in the implementation of our algorithms.[1]

Another property of repeating digits shows when considering $\frac{1}{p}$ where p is a prime not equal to 2 or 5. In this case, the period $\frac{1}{p}$ (i.e. the length of the repeating sequence of digits) is a divisor of $p - 1$.[1] This is useful as a sanity check when running the implementation of algorithms for primes:

- If the script ran for p iteration and has not found a repeating sequence, then the implementation is wrong.
- If the found period does not divide $p - 1$, then (again) the implementation is wrong.

The above property can be further developed to show that for any $1 < q \in \mathbb{N}$ not containing factors of powers of 2 and 5 the following two statements are equivalent:

$$\text{the period of } \frac{1}{q} \text{ is } l \tag{1}$$

$$l \text{ is the smallest integer such that } 10^l \equiv 1 \text{ (mod q).[1]} \tag{2}$$

Equivalence relation in (2) then gives a way to confirm the computed period of $\frac{1}{q}$ (admittedly only for smaller q as 10^l blows up quickly). Furthermore, it follows that for any q, period($\frac{1}{q}$)$\leq q - 1$.[1]

Finally, we can extend our property to consider $\frac{p}{q}$, where q is given as above and $p \in \mathbb{Z}^+$ with $p < q$. We notice there are two possibilities: (a) p and q are co-prime, and (b) p and q share common factors. Intuitively, in case (a), the period of $\frac{p}{q}$ must be the same as that of $\frac{1}{q}$ since we can notice that the first can be formed from the second by multiplying the digits in sequence by p (carrying over the extra powers of 10 as usual) preserving the underlying periodicity (there is no risk of "overflow" of one period onto another as that would be equivalent to $\frac{p}{q} > 1$ when $p < q$ - a contradiction). In case (b), we can reduce p and q until they do not share any factors. The new pair p' and q' will behave as outlined in (a), albeit with a possibly shorter period. It also follows that if $q > 5$ is prime, then all $\frac{p}{q}$ will have the same period. This intuition is confirmed by Dickinson.[1]

[1]While this property is useful for both algorithms it is not strictly necessary in the case of algorithm 1; an alternative version of the algorithm 1 will be discussed in the next subsection.

2.2 Algorithm 1

The steps in the algorithm for $p = p_0$ and q (as outlined in the introduction) can be summarised as follows:

$$1.\ \frac{p_0}{q} = 0.\dot{a_1}a_2a_3...\dot{a_l}...$$

$$2.\ \frac{10p_0}{q} = a_1.a_2a_3...a_l... \implies \text{floor}(\frac{10p_0}{q}) = a_1$$

$$3.\ \frac{10p_0}{q} - \frac{a_1q}{q} = \frac{10p_0}{q} - \text{floor}(\frac{10p_0}{q}) = 0.a_2a_3...a_l... = \frac{p_1}{q}$$

$$4.\ p_i = 10p_{i-1} - a_iq$$

5. Repeat for all generated p_i until $p_i = p_0$ at which the beginning of the repeating sequence has been reached again

Notice that we have used the property of repeating digits starting right after the decimal point since q does not contain factors of powers of 2 and 5. To extend the algorithm to allow for all values of q, we need to modify the algorithm as follows:

1. If $q = 2^m5^n$, we can immediately say there will not be a repeating sequence of digits.
2. If $q = 2^m5^nr$ for $1 < r \in \mathbb{N}$, $2 \nmid r$ and $5 \nmid r$, the steps in the algorithm can be performed as described as above but the condition for stopping is $p_i = p_{\max(m+1,n+1)}$ for $i > \max(m+1, n+1)$, effectively ignoring the first couple digits.

2.3 Algorithm 2

The next algorithm comes from a book titled "The Vedic Mathematics", a collection of algorithms aiming to aid mental arithmetic.[2] Unlike algorithm 1, this one requires q to end with digit 9 to work. Its steps can be outlined as follows ($\text{mod}(a, b)$ signifies the remainder after division of a by b):

$$1.\ \frac{p_0}{q} = 0.\dot{a_1}a_2a_3...\dot{a_l}...$$

$$2.\ q = q_1q_2...q_s9 \rightarrow q' = q_1q_2...q_s + 1 \text{ for some } s \in \mathbb{N}$$

$$3.\ a_i = \text{floor}(\frac{p_{i-1}}{q'}) \text{ since } q' = \frac{q}{10} + 0.1$$

4. $p_i = 10\text{mod}(p_{i-1}, q') + a_i$

5. Repeat for $i \in \{1, 2, ...\}$ until $p_i = p_0$ at which point the beginning of the repeating sequence has been reached again

3 Implementation

3.1 Task 1 - `RecFrac1a`

Two versions of this function have been written. The first (`RecFrac1a`) deals with q ending with digit 9 and validates that this is the case, while the second (`RecFrac1b`) handles the general case of any $q \in \mathbb{N}$ with the only restriction being $0 < p < q$ (which is validated for both functions). The code below shows the implementation of `RecFrac1a`, while the other function can be seen in Appendix A.

```
function [k , a ] = RecFrac1a(p, q)
% RecFrac1 uses Algorithm 1 and returns the recurring digits in the
% decimal expansion of p /q , among all pairs of
% positive integers (p , q ) such that p < q and the last digit ...
    of q is a 9
    if p > q
        fprintf('Invalid input: %d is larger than %d.\n', p, q);
        k = [];
        a = [];
        return
    end
    q_ = num2str(q);
    if q_(length(q_)) ~= string(9)
        fprintf('Invalid input: last digit of %d is not 9.\n', q)
        k = [];
        a = [];
        return
    end

    a = [];
    p_ = p;

    while true
        a_ = floor(10*p_ / q);
        p_ = 10*p_ - q*a_;
        a = [a a_];
        if p == p_
            break
        end
    end
    k = length(a);
end
```

The reason for separating the implementation into two scripts is that `RecFrac1a` has shorter run-time than either of `RecFrac1b` and `RecFrac2` (the function employing algorithm 2). It is therefore preferable to use it in task 3.

3.2 Task 2 - RecFrac2

Only one version of `RecFrac2` has been written and it can be seen below.

```
function [k , a ] = RecFrac2( p,q )
% RecFrac2 uses Algorithm 2 and returns the recurring digits in the
% decimal expansion of p /q , among all pairs of
% positive integers (p , q ) such that p< q and the last digit ...
    of q is a 9.
    if p > q
        fprintf('Invalid input: %d is larger than %d.\n', p, q);
        k = [];
        a = [];
        return
    end
    q_ = num2str(q);
    if q_(length(q_)) ~= string(9)
        fprintf('Invalid input: last digit of %d is not 9.\n', q)
        k = [];
        a = [];
        return
    end

    if q > 10
        q = str2double(q_(1:length(q_)-1)) + 1;
    else
        q = 1;
    end
    p_ = p;
    a = [];

    while true
        a_ = floor(p_/q);
        a = [a a_];
        p_ = 10*mod(p_, q) + a_;
        if p_ == p
            break
        end
    end
    k = length(a);
end
```

Note that although the algorithm 2 is in practice faster than algorithm 1 to use as a mental arithmetic tool, it is slightly more costly computationally. While RecFrac1 requires only a single division per iteration of the loop, `RecFrac2` needs to perform

both a standard division and a modulo division. The effect is barely noticeable for smaller p, q but it does accumulate for larger values. For example, for $p = 3433356$ and $q = 12345678999$ the times taken for `RecFrac1a` and `RecFrac2` are (on average over 10 runs each) 134s and 140s respectively.

3.3 Task 3

As mentioned above, the script for task 3 employs `RecFrac1a` due to its faster performance. The actual implementation can be seen below.

```
1  % This script uses RecFrac1a to find pairs (p,q), 0<p<q<maxN+1, ...
       q ends with
2  % digit 9 such that p/q has the longest period of repeating ...
       digits. It then
3  % prints out all those pairs and the repeating digits sequence ...
       for one of
4  % the pairs.
5
6  format compact
7  maxN = 449;
8  maxPairs = [];
9  maxDigits = [];
10
11 for s = 9:10:maxN
12     for r = 1:s-1
13         [k, a] = RecFrac1a(r, s);
14         if k > length(maxDigits)
15             maxPairs = [r, s];
16             maxDigits = a;
17         elseif k == length(maxDigits)
18             maxPairs = [maxPairs;r, s];
19         end
20     end
21 end
22
23 fprintf("The highest number of reccuring digits is %d.\n", ...
       length(maxDigits))
24 fprintf("The pairs r, s which produce chains of that length are:\n")
25 disp(maxPairs)
26 fprintf("An example of such chain for r=%d and s=%d is:\n", ...
       maxPairs(1,1), maxPairs(1,2))
27 disp(maxDigits)
```

Nonetheless, in the testing of the script the other two functions have been used as well. The following piece of MATLAB code had been inserted between lines 13 and 14 to catch potential divergences in results. Although it uses `RecFrac2`, the same code can be (and has been) used with `RecFrac1b` by replacing the function call.

```
[k2 a2] = RecFrac2(r, s);
if k2 ≠ k || a2 ≠ a
   fprintf("Incompatible results for r = %d and s = %d\n", r, s)
   fprintf("k = %d ; k2 = %d ; a = %d ; a2 = %d\n", k, k2, a, a2)
end
```

As a result of testing, no discrepancies have been found.

4 Results

After running the script for task 3 we find that the longest period (of size 418) is produced when $q = 419$ and $p = 1, 2, 3, ..., 418$. Going back to the properties of repeating fractions we see that it makes sense that the longest period occurs for one of the largest q since we know it cannot be larger than $q - 1$. We can however wonder why it is 419 and not 429, 439 or 449. In case of 429 we see that it is a composite number so the period of its fraction will depend on periods of its prime factors, which will be lower (and additionally the period will vary with p). On the other hand, 419, 439 and 449 are all prime. The solution here is very much non-trivial, but 419 is a long prime, that is a prime p for which $\frac{10^{p-1}-1}{p}$ generates a cyclic number (a number for which consequent integer multiples only permute the order of digits). It turns out that for such p, $\frac{1}{p}$ is guaranteed to have $p - 1$ recurring digits. It also turns out that $p = 40k + 9$ and $p = 40k + 39$, $k \in \mathbb{N}$ can never be long primes, and so $\frac{1}{p}$ must have less than $p - 1$ recurring digits.[3] As 449 and 439 are examples of such primes (for $k = 11$ and $k = 10$ respectively) we can now see why 419 produces the longest period. For more information, the sequence of long primes can be found the OEIS.[4]

References

[1] Leonard Eugene Dickson. *History of the Theory of Numbers: Divisibility and Primality*. Vol. 1. Dover Publications Inc., 2005.

[2] Bharati Krsna Tirthaji. *Vedic Mathematics: Sixteen Simple Mathematical Formulae From The Vedas*. 16th. Motilal Banarsidass, 2000.

[3] John H. Conway and Richard K. Guy. *The Book of Numbers*. Copernicus, 1998.

[4] N.J.A. Sloane. *The On-Line Encyclopedia of Integer Sequences*. 2021.

5 Appendix A - RecFrac1b

```
function [k , a ] = RecFrac1b(p, q)
% RecFrac1 uses Algorithm 1 and returns the recurring digits in the
% decimal expansion of p /q , among all pairs of
% positive integers (p , q ) such that p < q.
    if p > q
        fprintf('Invalid input: %d is larger than %d.\n', p, q);
        k = [];
        a = [];
        return
    end

    m = 0;
    q_temp = q;
    while true
        if mod(q_temp, 2) ~= 0
            break
        end
        q_temp = q_temp/2;
        m = m+1;
    end
    n = 0;
    while true
        if mod(q_temp, 5) ~= 0
            break
        end
        q_temp = q_temp/5;
        n = n+1;
    end
    if q_temp == 1
        disp("The decimal terminates - no repeated digits.")
        a = [];
        k = [];
        return
    end

    delay_index = max(n+1, m+1);
    a = [];
    p_ = [p];

    for i = 1:q
        a_ = floor(10*p_(i) / q);
        p_ = [p_ (10*p_(i)-q*a_)];
        if i >= delay_index+1
            if p_(delay_index) == p_(i)
                break
            end
        end
        a = [a a_];
    end
    a = a(delay_index:length(a));
    k = length(a);
end
```