PSBC Project 1: Three Short Questions

Student ID: 9641876

March 6, 2018

1 Cubic Taxicab Number

A *cubic taxicab number* is a positive integer that may be expressed as the sum of two cube numbers, in two or more distinct ways. The purpose of the code in this question is to define a function `CubicTaxicabNum(N)` which takes as input a positive integer N and returns the smallest cubic taxicab number greater than or equal to N.

The function works by checking each integer n greater than or equal to N until a cubic taxicab number is found. For all such n it runs through all positive integers a less than or equal to $\sqrt[3]{n}$ and finds another number b such that $a^3 + b^3 = n$. If it finds b to be an integer, these values of a and b are stored as one way of expressing n as a sum of two cube numbers. It then continues to test for more distinct ways of doing so. As soon as a value of n is found with two distinct ways of expressing it as a sum of two cube numbers, the loops are broken and the output, `ctn` is set equal to n.
Here is the code written to achieve this.

```
function ctn = CubicTaxicabNum(N)
% takes an integer N and return the smallest number
   n >= N such that n is a cubic taxicab number

n = N;

while n >= N

    ways = 0;    % number of ways of expressing n as
       two cube numbers

    cubes = zeros(2);    % 2 by 2 matrix to store
       such cube numbers

    for a = 1:1:(n^(1/3))

        b = nthroot(n-a^3, 3);

        if mod(b, 1) == 0 && ~ismember(b, cubes) %
           checks that b is an integer and is not
           part of a solution already found

            ways = ways + 1;
            cubes(ways, 1:2) = [a, b];  % stores
               solution a and b
            if ways == 2    % checks if two distinct
                ways of expressing n as two cube
               numbers have been found

                ctn = n;
                n = -1;
                break;
            end
        end
    end

    n = n+1;    % increases n for the while loop

end

end
```

The comments in green provide extra detail on certain parts of the code and how they function. To demonstrate the function, take an example value of N, say 36032. The output of the function is then

```
>> CubicTaxicabNum(36032)

ans =

      39312
```

This means that the smallest cubic taxicab number greater than or equal to 36032 is 39312. That it is a cubic taxicab number at all can be checked easily;

$$2^3 + 34^3 = 39312.$$

$$15^3 + 33^3 = 39312.$$

2 Catalan's Constant

Catalan's constant is a mathematical constant given by

$$G = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)^2} = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \ldots \approx 0.915965594177219.$$

It is named after Eugène Charles Catalan and while it may be expressed as varying sums and series, it is not yet proven to be either rational or irrational. For this reason it is often more convenient to approximate it by a rational number $\frac{p}{q}$, where p and q are both positive integers.

This is what motivates the focus of this question. The task was to write a function `RatAppCat(N)` such that `[p, q] = RatAppCat(N)` would return the pair of integers that give the closest approximation to Catalan's constant, provided the sum of these integers is less than or equal to N.

The code for the function runs through all positive integers t less than N and for each t calculates $\frac{t}{s}$ for every positive integer s such that $t + s \leq N$. As it goes through all the possible combinations it checks how close an approximation the given ratio is and gradually stores closer and closer approximations in the variable `app` and stores the corresponding values of t and s as p and q respectively. Once all ratios have been calculated and tested, the function returns the final values of p and q.

The finished code is as follows;

```
function [p, q] = RatAppCat(N)
% RATAPPCAT Finds the best rational approximation p/
   q of the Catalan's constant, among all pairs of (
   p,q) such that p+q <=N

G = 0.915965594177219;
app = N;     % sets an initial value for the best
   approximation

for t = 1:1:N-1

    s = 1;

    while s <= N-t
```

```
        rat = t/s;  % finds the ratio of the two
           positive integers

        close = abs(G - rat);   % evaluates how
           close to G this ratio is

        if close < abs(G - app) % if the ratio is
           less than the current best approximation,
            reassigns the value of the best
           approximation

            p = t;
            q = s;
            app = p/q;
        end

        s = s + 1;  % increases s for the while loop

    end

end

end
```

The comments in green provide additional explanation of the code. The output of this function with $N = 2018$ is then

```
>> [p, q] = RatAppCat(2018)

p =

   109


q =

   119
```

Meaning the rational number obtained by the ratio of these is

$$\frac{109}{119} = 0.915966386554622.$$

which is accurate up to 5 decimal places.

3 Sum of Reciprocal Squares with Prime Factors

To start off this section, consider the sum

$$\sum_{k=1}^{\infty} \frac{1}{k^2}.$$

This is the sum of reciprocal squares, proven by Leonhard Euler to be equal to $\frac{\pi^2}{6}$. The sum to be considered in this question is similar to this, but with each term multiplied by $(-1)^{\Omega(k)}$, where $\Omega(k)$ is the total number of prime factors that k has, with the convention that $\Omega(1) = 0$. The sum looks like this;

$$\sum_{k=1}^{\infty} \frac{(-1)^{\Omega(k)}}{k^2} = \frac{1}{1^2} - \frac{1}{2^2} - \frac{1}{3^2} + \frac{1}{4^2} + \dots$$

The goal is to approximate this sum by using truncations of a finite number of terms, assuming that computing capacity only allows for up to 100,000 terms. The function defined for this takes no input but simply returns the approximation for the sum of reciprocal squares.

The way it works is to calculate the finite sum for N equal to powers of 10 up to $10^5 = 100000$. For each of these finite sums it calculates the truncation error and finds the sum with the smallest truncation error to use as the approximation. The error is then used to evaluate how many decimals the approximation is accurate to.

The function is defined;

```
function SumPF
% SUMPF   find an approximation of the sum of
   reciprocal squares with prime factors, based on
   truncations at most 100000 terms

smallesterr = 1;    % sets an initial value for the
   smallest truncation error

for N = 10.^[0:1:5]

    format long

    sum = 0;    % sets initial value of the sum

    for k = 1:1:N   % loops through all positive
       integers up to N in order to compute the sum
```

```
        if k == 1
            sum = sum + 1;  % accounts for the
               convention, factor(1) = 0 [MATLAB
               gives factor(1) = 1]
        else
            sum = sum + ((-1)^(length(factor(k)))/(k
               ^2));   % for all other values of k,
               adds the term to the sum
        end

    end

    err = 1/((N+1)^2);  % calculates the truncation
       error for each sum [(-1)^(length(factor(k))
       not necessary as absolute value needed]

    if err <= smallesterr   % ensures the
       approximation is the sum with the smallest
       error
    end
        smallesterr = err;
        approx = sum;

end

approx
err = smallesterr

end
```

Again, with additional explanation of each part of the code in the comments.

The output from this function is

```
>> SumPF

approximation =

   0.657973626065316


error =

     9.999800002999960e-11
```

So the sum is approximately equal to 0.657973626065316 with a truncation error of $9.999800002999960 \times 10^{11}$ which can be rounded to $1 \times 10^{10} =$ 0.0000000001. The value of this error then suggests that the approximation is accurate to 9 decimal places.