MATH36032 PSBC Assignment 1: Three short questions

March 2, 2020

Introduction

For this report any reference to Listing-X, for some integer X, is a reference in the "Appendix" section corresponding to a complete Matlab source code. Futhermore any reference of [X], for some integer X, is a reference in the "References" section.

1 Cubic taxicab number

A cubic taxicab number M is a positive integer that can be expressed as at least 2 distinct pairs (a, b) to $M = a^3 + b^3$, for some positive integer constants a and b. An example of this can be given by the smallest cubic taxicab number 1729:

$$1729 = 1^3 + 12^3 = 9^3 + 10^3$$

Here we have the two distinct pairs $(1, 12)$ and $(9, 10)$, hence 1729 is a cubic taxicab number.

1.1 The smallest cubic taxicab number greater than N

It can be quite hard to find a taxicab number, as the only known way we can find one is by testing every number against the criteria for a cubic taxicab number. For a human this is highly infeasible, so instead I will write a matlab function which will take some constant N parameter, and return the lowest cubic taxicab number that is greater than or equal to N.

The first step would be to set up some constant loop, with an incrementing variable N at the end of the loop, for which we will test if the current N is a cubic taxicab number, and a counter variable to count the number of pair solutions (a, b) for the current N, which we will reset to zero at the end of every loop. Now to test the current N, there are two methods:

The first and easiest way to do this would be to test the sum of cubes of a and b for every integer value between 0 to $\sqrt[3]{N}$ for if they are equal to N, count the number of pairs, then return the value of N when the counter is greater than 2 (not equal to as we will have to consider repeated pairs). Clearly it can be seen that this this a bad method to use when we start to test for large values of N, as the time taken to complete will increase drastically (the list of possible values to test through would grow by $\sqrt[3]{N}^2$).

However, a much better method to use would be to start with $a = 1$ and $b = \sqrt[3]{N}$ where b is round down. Then we have a constant while loop testing whether $a < b$, as this will eleminate duplicates pair solutions from showing. Inside the loop we will calculate the sum of the cubes of a and b, i.e: sum $= a^3 + b^3$. Then make a 3 way decision, if the sum is:

- less than N, then we don't have a pair solution, so we increase the value of a by one.
- greater than N, then we don't have a pair solution, for any value of a, so instead we decrease b by one.
- equal to N, then we have a pair solution to the problem, so we increase the counter variable by one, and then increase a by one, and decrease b by one, as we know there cannot be another pair with either values of a and b.

After the decision, we check if the counter is greater than or equal to 2, as this means we have found 2 or more pair solutions. If this is true, then we set the return variable equal to the current N and return back out of the function. The code for this can be found in the appendix under Listing 1.

To test this function, we record the following result for $N = 36032$, which gives:

Result for Cubic Taxicab Number

```
>> CubicTaxicabNum(36032)

ans =

       39312
```

Using a online resource [1] we can see that the lowest cubic taxicab number greater than or equal to 36032 is 39312, which is exactly the same answer we got. Futhermore, if we use Matlab's "Run and Time" function, we can see the function took a total time of 0.056 seconds, which shows it is sufficiently fast.

2 Catalan's constant

Catalans's constant often appears in estimates of combinatorial functions [2] and is one of the lesser known constants. This constant can be expressed in different ways by the use of various sums or special integrals, such as:

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

2.1 Estimating Catalan's constant

Unfortunately, this constant has not been proven to be either rational or irrational, so to get around this we will write a Matlab function which takes some constant positive integer N, and will return the best two values p and q such that, $\frac{p}{q}$ is the best rational approximation we can make for G (that is $\left|\frac{p}{q} - G\right|$ is smallest). As we are estimating G to a "potential" infinite amount of decimals, we will apply the constraint that $p + q$ must be less than or equal to N.

The issue with this, is there is no "clever" way to quickly find the "best" approximation, we have to test each possible fraction that obeys the constraint. This can become an issue when trying to calculate the approximation for large values of N, as it will take significantly longer (the potential values to look through increase by roughly N^2). However, there are certain methods we can use

which will significantly reduce the list of fractions we must search through. The first method is simply to use the constraint, so we only need to look at the values of p and q for which the sum is less than or equal to N. The next method we can use is, if we look at the Catalan's constant $G = 0.915965594177219$, we can note that $G > 0.9$ which implies the numerator p must always be fairly close to the denominator q for some given tolerance, i.e: $\frac{p}{q} > 0.9 \implies p > 0.9q$. Therefore, we only look at numerators values p, greater than 90% of the denominator q. The code for this can be found in the appendix under Listing 2.

To test this function, we record the result for $N = 2018$, which gives:

Result for Catalans constant

```
>> tic
[a, b] = RatAppCat(2018);
toc
Elapsed time is 0.002152 seconds.
>> a,b

a =

   109


b =

   119
```

We use the functions tic/toc to measure the time it took for the function to find the best fraction. We get a time of 002152 seconds, which shows it is fairly fast, however this time will start to drastically increase when we start looking at much greater values of N, which would be an area than can be improved upon. If we look at the value of the fraction:

Value of result

```
>> 109/119

ans =

   0.915966386554622
```

We see, when comparing it to Catalan's constant, it is accurate up to 5 decimal places, which is fairly accurate.

3 Sum of reciprocal squares with prime factors

If we look at the sum of the reciprocal squares (also known as the Basel problem), given by [3]:

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{1}{1^2} + \frac{1}{2^2} + \ldots = \frac{\pi^2}{6}$$

Then we can raise a similar problem, that is the sum of reciprocal squares with prime factors given by:

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

where $\Omega(k)$ is the total number of prime factors of a positive integer k. Then we can ask ourself a simple question, what does this equal/does the sequence converge to?

3.1 Estimating the sum of reciprocal squares with prime factors

We can use a Matlab function to find the answer to our sum of reciprocal squares problem by truncating a finite number of terms N, of which we can keep increasing N to find out what the sum tends towards. The code for this can be given by:

Complete Matlab code for estimating the sum of reciprocal squares with prime factors

```
function SumPF
% SUMPF find an approximation of the sum of reciprocal squares with prime factors

% Short function, which returns the number of primes .
primeFactors = @(x) numel(factor(x));
% Set k as a symbol.
syms k;
% Set f as the function we are using for the sum.
f = (-1)^primeFactors(k)/k^2;
% Find the sum of f between k=1 to 1000000.
symsum(f, k, 1, 1)
symsum(f, k, 1, 2)
symsum(f, k, 1, 10000)
symsum(f, k, 1, 100000)
symsum(f, k, 1, 1000000)
```

The output for this when called is given by:

Result of SumPF function

```
>> SumPF

ans =
-1

ans =
-5/4

ans =
psi(1, 10001) - pi^2/6

ans =
psi(1, 100001) - pi^2/6

ans =
psi(1, 1000001) - pi^2/6
```

A close look would show that we can rewrite the first 2 results as:

$$\text{psi}(1,2) - \pi^2/6 = -1$$
$$\text{psi}(1,3) - \pi^2/6 = -5/4$$

This gives a very clear relation that each term can be written as $\text{psi}(1, X)$ for some integer $X \geq 2$. Futhermore we can see the relationship that as N tends to infinity then X must also tend to infinity. Now when X tends to infinity, what does $\text{psi}(1, X)$ tend to? To find this we have to look at how the psi function calculations work, which is given by [4]. We learn that psi (also known as the digamma function) is the logarithmic derivative of the gamma function. As it is a log derivative of the gamma function then it must tend to zero. We can verify this manually by finding the value as we increase X:

Increasing values of X

```
>> psi(1, 2)

ans =
   0.644934066848226

>> psi(1, 3)
```

```
ans =
   0.394934066848226

>> psi(1, 10001)

ans =
     9.999500016666657e-05

>> psi(1, 100001)

ans =
     9.999950000166661e-06

>> psi(1, 1000001)

ans =
     9.999995000001671e-07
```

Again we can see that psi$(1, X)$ tends to 0 as N tends to infinity. Therefore this leaves us with $-\frac{\pi^2}{6}$ as the answer to the sum, that is:

$$\sum_{k=1}^{\infty} \frac{(-1)^{\Omega(k)}}{k^2} = \frac{1}{1^2} - \frac{1}{2^2} - \frac{1}{3^2} + \frac{1}{4^2} + \ldots = -\frac{\pi^2}{6}$$

Futhermore, if we want to know how many correct decimals our estimate has against $\frac{\pi^2}{6}$ for some integer value of N, we can just look at the value of psi$(1, X)$. As we are just adding psi$(1, X)$ to $\frac{\pi^2}{6}$, then we can use the fact that psi$(1, X)$ decrease by 10^{-1} every time that X increase by $10^1 + 1$, which results in if:

- $N = 1$ then it is accurate to 1 significant figures and 0 decimal places.
- $N = 2$ to 10 then it is accurate to 1 significant figures and 0 decimal places.
- $10^P + 1 \leq N \leq 10^{P+1}$ for $N \geq 11$ and for some positive integer constant P, then it is accurate to $P + 1$ significant figures or P decimal places when rounded.

References

[1] N. J. A. Sloane, `http://oeis.org/A001235`, updated 2020.

[2] Jonathan Sondow & Oleg Marichev `http://mathworld.wolfram.com/CatalansConstant.html`, updated 2020.

[3] Wikipedia `https://en.wikipedia.org/wiki/Basel_problem`, updated 2020.

[4] MathWorks `https://www.mathworks.com/help/matlab/ref/psi.html`

Appendix

Listing 1: Complete Matlab code for Cubic Taxicab Number

```
function ctn = CubicTaxicabNum(N)
% CUBICTAXICABNUM return the smallest cubic taxicab number greater than
% or equal to N %

```

```
% Want to find at least one solution to a^3+b^3=c^3+d^3=N for some constants a,b,c,d

% Set initial value of smallest cubic taxicab number
ctn = 0;
% Set initial value of counter
counter=0;
% Keep iterating untill we find the smallest cubic taxicab number
while(1)
    % Set 'a' to the smallest possible solution and 'b' to the largest
    % possible solution.
    a=1;
    b=floor(nthroot(N, 3));
    % We use 2 facts here; first when we have found a pair, then neither
    % number of that pair can appear again as a solution for the current
    % 'N' so we increase 'a' by 1 and decrease 'b' by 1.
    % Second, we want to tend to a potential solution. So if the sum of
    % our current values of 'a' and 'b' is:
    %   - less than our current 'N' we want to increase our 'a' by 1.
    %   - larger than our current 'N', then this tell us that there is no
    %     potential value of 'a' that will give us a potential solution, so
    %     we decrease b by 1 instead.
    % When we find a solution, then we dont need to store the pair of
    % values, so instead we use a "counter" variable to count the number of
    % pairs we have found.
    % We also obviously dont want to find repeat solutions, so we only keep
    % looking for solutions for 'N 'while 'a<b'.
    while(a<b)
            sum = a^3 + b^3; % Find the sum of a^3 and b^3.
            % Now test the sum and choose the correct decision for 'a' and
            % 'b'.
            if(sum<N)
               a = a+1;
            elseif(sum>N)
               b = b-1;
            else
               counter = counter+1; % Increase the counter.
               a = a+1;
               b = b-1;
            end
        % Check if we have more than 2 pairs, if so set the return varible
        % ctn to N, and then return control back to parent script.
        if(counter>=2)
            ctn = N;
            return
        end
    end
    % Every time we test a new value for if it is a cubic taxicab number we
    % want to reset the counter to 0/
    counter = 0;
    % Iterate to the next potential cubic taxicab number until it is
    % correct.
    N = N+1;
end
```

Listing 2: Complete Matlab code for Catalan's constant

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

%% Set initial values.
% Catalan s constant.
G = 0.915965594177219;
p=0; % Initial values of p.
q=1; % Initial values of q.
i=1; % Initial testing value for numerator.
j=1; % Initial testing value for denominator.
oldDiff = 1; % The difference between G and the last value of i & j.

% Keep looping while under constraint, and the denominator doesnt equal N
% (As the only value of i when j=N is 0).
while(i+j<=N && j~=N)
   % Calculate the new difference between G and the current i/j. Note we
   % use abs, as we don't care about sign, only size difference.
   diff = abs(G-i/j);

```

```
    % Check if the current i/j difference is smaller than the last best
    % difference. If so we update the return variables p,q, and oldDiff.
    if(diff < oldDiff)
        p = i;
        q = j;
        oldDiff = diff;
    % We check if the next variable of i+j+1 will exceed the restraint, and
    % if so we just want to move on to the next value of j. Note that we set
    % i equal to roughly 90% of j, as the Catalan's constant is greater than
    % 0.9, so we can skip a large amount of values by setting i high enough.
    elseif(i+j+1>N)
        j = j+1;
        i = i-ceil(0.1*j);
    % Check if the difference is still greater than the old difference, if
    % so we just want to increase i by 1 to attempt to get closer. Note we
    % also include i cannot equal j, as we want to increase j rather than
    % increase i at i=j.
    elseif(diff > oldDiff && i~=j)
        i = i+1;
    % Increase j by 1, and using the same reason as above for i.
    else
        j = j+1;
        i = i-ceil(0.1*j);
    end
end
```