# Problem Solving By Computer - Project 1

## 1 Cubic Taxicab Number

**Definition 1.1.** A **cubic taxicab number** is a positive integer that can be expressed as the sum of two positive cubic numbers in at least two distinct ways.

**Example 1.2.** The first cubic taxicab number is 1729, since $1729 = 1^3 + 12^3 = 9^3 = 10^3$.

**Problem 1.3.** Given a positive integer $N$, how can we determine the smallest cubic taxicab number greater or equal to $N$? In particular, how can we write a function `CubicTaxicabNum(N)` such that `ctn = CubicTaxicabNum(N)` returns the smallest cubic taxicab number greater or equal to $N$?

### 1.1 Approach

Suppose that we have a function `IsCubicTaxicabNum(N)` that determines whether a positive integer $N$ is a cubic taxicab number, then we are able to deduce the smallest cubic taxicab number greater or equal to $N$ by checking the integers greater than or equal to $N$ in turn, until we find a cubic taxicab number.

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

ctn = N;
while (~IsCubicTaxicabNum(ctn))
    ctn = ctn + 1;
end
end
```

All that remains to do is to implement the function `IsCubicTaxicabNum(N)`, such that the function returns `1` if $N$ is a taxicab number and `0` otherwise.

## 1.2 Is it a Cubic Taxicab Number?

Before looking at the implementation of the function `IsCubicTaxicabNum(N)`, we will first make some observations that will help us derive an implementation for `IsCubicTaxicabNum(N)`.

**Claim 1.4.** Suppose that $t$ is a positive integer such that $t = x^3 + y^3$, where $x, y$ are positive integers, then $x, y \leq \lfloor \sqrt[3]{t} \rfloor$.

*Proof.* Let $x, y, t$ be positive integers such that $t = x^3 + y^3$. Suppose by contradiction, we have $x > \lfloor \sqrt[3]{t} \rfloor$. Note that $\lceil \sqrt[3]{t} \rceil \geq \sqrt[3]{t}$. As $x$ is an integer, we must have $x \geq \lceil \sqrt[3]{t} \rceil$. Together, we have $x \geq \lceil \sqrt[3]{t} \rceil \geq \sqrt[3]{t}$ and so $x^3 \geq \lceil \sqrt[3]{t} \rceil^3 \geq t$. In particular, for $t = x^3 + y^3$ to hold, we must have $t = x^3$, but this means that we must have $y = 0$, contradicting the choice of $y$ being a positive integer. □

**Observation 1.5.** Suppose that $t$ is a positive integer such that $t = x^3 + y^3$, where $x, y$ are positive integers, then there is no positive integer $a \neq x$, such that $t = a^3 + y^3$. Similarly, there is no positive integer $b \neq y$, such that $t = x^3 + b^3$.

*Proof.* Let $a, x, y, t$ be positive integers such that $x^3 + y^3 = t$ and $a^3 + y^3 = t$. We have $x^3 + y^3 = t = a^3 + y^3$, and so $x^3 = a^3$. As both $x$ and $a$ are positive integers, we must have $x = a$. By symmetry, the latter statement also holds. □

**Observation 1.6.** Suppose that $t$ is a positive integer such that $x^3 + y^3 < t$, where $x, y$ are positive integers, with $x \leq y$. Then, there is no positive integer $z < y$, with $x^3 + z^3 = t$.

*Proof.* Let $x, y, t$ be positive integers with $x \leq y$ and $x^3 + y^3 < t$. Let $z$ be any positive integer with $z < y$, then we must have $z^3 < y^3$. It follows that $x^3 + z^3 < x^3 + y^3 < t$ and so the result holds. □

**Observation 1.7.** Suppose that $t$ is a positive integer such that $x^3 + y^3 > t$, where $x, y$ are positive integers, with $x \leq y$. Then, there is no integer $z > x$, with $z^3 + y^3 = t$.

*Proof.* Let $x, y, t$ be positive integers with $x \leq y$ and $x^3 + y^3 > t$. Let $z$ be any positive integer with $z > x$, then we must have $z^3 > x^3$. It follows that $z^3 + y^3 > x^3 + y^3 > t$ and so the result holds. □

### 1.2.1 Implementation

The implementation of `IsCubicTaxicabNum(N)` in MATLAB is as follows.

```
function isTaxicab = IsCubicTaxicabNum(N)
% ISCUBICTAXICABNUM    Returns isTaxicab = 1, if N is a taxicab number
%                      Returns isTaxicab = 0, otherwise

nPairs = 0;
left = 1;
right = floor(nthroot(N, 3));                    % Claim 1.4
while (left ⩽ right)
    sum = left^3 + right^3;
    if (sum == N)
        nPairs = nPairs + 1;
        if (nPairs == 2)                         % If we have found the
            isTaxicab = true;                    % second pair, we are done
            return;
        end
        left = left + 1;                         % Observation 1.5
        right = right - 1;                       % Observation 1.5
    elseif (sum < N)
        left = left + 1;                         % Observation 1.6
    else
        right = right - 1;                       % Observation 1.7
    end
end
isTaxicab = false;
end
```

## 1.3 Analysis

**Example 1.8.** We can verify that the first cubic taxicab number is indeed 1729 by calling `CubicTaxicabNum(1)`.

**Example 1.9.** The smallest cubic taxicab number greater or equal to 36032 is 39312. Indeed, 39312 is a cubic taxicab number since $39312 = 2^3 + 34^3 = 15^3 + 33^3$.

### 1.3.1 Complexity

The implementation of `IsCubicTaxicabNum(N)` makes use of the *two-pointer technique*, in which we have two pointers, `left` and `right`, and we insist that `left ⩽ right`. We compute the sum of the cubes, `S = left^3 + right^3` and if `S < N`, then we increment `left` by 1, and if `S > N`, then we decrement `right` by 1. If `S == N`, then we increment `left` by 1 and decrement `right` by 1. It follows that we have at most $\sqrt[3]{N}$ iterations of the **while** loop, and the operations inside the **while** loop run in constant time, so the time complexity of `IsCubicTaxicabNum(N)` is $O(\sqrt[3]{N})$.

# 2 Catalan's Constant

**Definition 2.1.** The **Catalan's constant** is a mathematical constant named after Eugéne Charles Catalan, and is defined as

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

Although the Catalan's constant $G$ can be expressed in terms of the above sum of series, it is not known whether $G$ is irrational or not.

**Problem 2.2.** In many situations, it would be more convenient to approximate the Catalan's constant as a ratio of two positive integers. Given a positive integer $N$, what is the best rational approximation $p/q$ of the Catalan's constant, subject to the constraint $p + q \leq N$. In particular, how can we write a function `RatAppCat(N)` such that `[p, q] = RatAppCat(N)` returns the pair of integers $p$ and $q$ for the best rational approximation $p/q$ of $G$, such that $p + q \leq N$?

## 2.1 Approximating Catalan's Constant

**Observation 2.3.** Given $G = 0.915965594177219$ and an integer $q$, we can find $p$ such that $p/q = G$, by computing $p = Gq$. Of course, $p$ needs not to be an integer; however, we can find the integer $p_0$ such that $p_0/q$ is closest to $G$ among all integers by $p_0 = \text{round}(p)$.

**Claim 2.4.** Suppose we have integers $p_0, p_1, q_0, q_1$ such that $p_0/q_0 = G$ and $p_1/q_1 = G$. If we have $q_1 > q_0$, then it must the case $p_1 > p_0$.

*Proof.* Let $p_0, p_1, q_0, q_1$ be integers such that $p_0/q_0 = G$, $p_1/q_1 = G$ and $q_0 < q_1$. We have $p_0 = Gq_0$ and $p_1 = Gq_1$, but as $q_0 < q_1$, we have $Gq_0 < Gq_1$. It follows that $p_0 < p_1$. □

### 2.1.1 Approach

As a result of Observation 2.3, we can enumerate over values of $q$ in ascending order, computing the *perfect* integer $p$ for each denominator $q$, and checking whether $p/q$ is a better approximation than any previous pairs of $(p, q)$ considered. Further, when we encounter a pair $(p, q)$ such that $p + q > N$, then we are done. Indeed, as we are enumerating over values of $q$ in ascending order, any subsequent pairs we consider, say $(p_0, q_0)$, we will have $q_0 > q_1$, and so $p_0 > p_1$ by Claim 2.4. In particular, $p_0 + q_0 > p + q > N$.

## 2.2 Implementation

The implementation of `RatAppCat(N)` in MATLAB is as follows.

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

G = 0.915965594177219;
p = 0; q = 1;                          % The best (p, q) pair so far
minDelta = abs(G - p / q);             % The difference between p / q and G

for q0 = 1 : N
  p0 = round(G * q0);                  % Observation 2.3
  if (p0 + q0 > N)                     % Claim 2.4
    return
  end
  delta = abs(G - p0 / q0);
  if (delta < minDelta)                % Update if current pair is better
    minDelta = delta;                  % than the best pair we have seen
    p = p0; q = q0;
  end
end
end
```

## 2.3 Analysis

**Example 2.5.** For $N = 2018$, the pair of integers 109 and 119 provides the best rational approximation of the Catalan's constant, with $109/119 = 0.915966386554622$.

### 2.3.1 Complexity

The implementation of `RatAppCat(N)` loops over possible denominators from 1 to $N$, and for each possible denominator, we compute the perfect numerator, and check whether this pair gives a better approximation than any previously encountered pairs. The operations in each iteration are run in constant time, and so the time complexity of `RatAppCat(N)` is $O(N)$, as the loop is run at most $N$ times. The space complexity of `RatAppCat(N)` is $O(1)$, as we only store a constant number of variables.

# 3 Sum of Reciprocal Squares with Prime Factors

**Definition 3.1.** The **sum of reciprocal squares**, also known as the Basel problem, asks for the precise summation of the reciprocal squares of the natural numbers. That is,

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \dots$$

which was shown to be exactly $\pi^2/6$ by Leonhard Euler.

**Definition 3.2.** The **prime omega function (with multiplicity)** $\Omega(n)$ counts the number of prime factors of a natural number $n$, counting multiplicity. By convention, we have $\Omega(1) = 0$.

**Example 3.3.** $\Omega(12) = 3$ as $12 = 2^2 \cdot 3^1$, $\Omega(25) = 2$ as $25 = 5^2$ and $\Omega(47) = 1$ as 47 is a prime number. Moreover, $\Omega(p) = 1$ for any prime number $p$, as prime numbers only have one prime factor, by definition.

**Definition 3.4.** The **sum of reciprocal squares with prime factors** asks for the precise summation of the following sum.

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

**Problem 3.5.** How can we derive a reasonable approximation of the sum of the reciprocal squares with prime factors? In particular, can we make any observations about the answer by truncating a finite number of terms?

## 3.1 Approximating the Sum of Reciprocal Squares with Prime Factors

**Observation 3.6.** For any $a, b \in \mathbb{N}$ with $a < b$, we have

$$-\sum_{n=a}^{b} \frac{1}{n^2} \leq \sum_{n=a}^{b} \frac{(-1)^{\Omega(n)}}{n^2} \leq \sum_{n=a}^{b} \frac{1}{n^2} \quad \text{and} \quad -\sum_{n=a}^{\infty} \frac{1}{n^2} \leq \sum_{n=a}^{\infty} \frac{(-1)^{\Omega(n)}}{n^2} \leq \sum_{n=a}^{\infty} \frac{1}{n^2}$$

**Observation 3.7.** Let $x, y, z \in \mathbb{R}$ with $x \leq y \leq z$. Let $n \in \mathbb{N} \cup \{0\}$. Let **round**(a, n) be a function that rounds $a$ to $n$ decimal places. If **round**(x, n) = **round**(z, n) = c, then **round**(y, n) = c. That is, if $x$ and $z$ round to the same value for $n$ decimal places, say $c$, then $y$ rounded to $n$ decimal places is also $c$.

**Notation 3.8.**

$$S(a, b) = \sum_{n=a}^{b} \frac{(-1)^{\Omega(n)}}{n^2} \quad \text{and} \quad T(a, b) = \sum_{n=a}^{b} \frac{1}{n^2}$$

We know that $T(1, \infty) = \pi^2/6$ and our goal is to approximate $S(1, \infty)$.

**Observation 3.9.** We can compute $T(k+1,\infty)$ by $T(k+1,\infty) = T(1,\infty) - T(1,k) = \pi^2/6 - T(1,k)$. That is, we can compute $T(k+1,\infty)$ by subtracting the first $k$ terms from $\pi^2/6$.

### 3.1.1 Approach

Let $k \in \mathbb{N}$. It follows from Observation 3.6 that we have

$$S(1,k) - T(k+1,\infty) \leq \underbrace{S(1,k) + S(k+1,\infty)}_{S(1,\infty)} \leq S(1,k) + T(k+1,\infty)$$

In particular, for some $d \in \mathbb{N}$, if we are able to find $k$ such that the above inequality holds, with the lower and upper bound rounding to the same value (up to $d$ decimal places), then by Observation 3.7, this value is an accurate approximation for $S(1,\infty)$ up to $d$ decimal places.

## 3.2 Implementation

### 3.2.1 Prime Omega Function

First, we need to implement the function `PrimeOmega(N)` such that `n = PrimeOmega(N)` returns the number of prime factors of $N$ (counting multiplicity). In fact, we can make use of the `factor(N)` function provided by MATLAB, which returns a vector containing the prime factors of $N$. It follows that `PrimeOmega(N)` can be implemented by counting the length of the vector returned by `factor(N)` and can be implemented as an anonymous function, as follows.

```
PrimeOmega = @(x) (x ~= 1) * length(factor(x));
```

As `factor(1) = [1]`, we use `(x ~= 1)` to handle the case $\Omega(1) = 0$. That is, if $x \neq 1$, then `PrimeOmega(x)` returns `length(factor(x))`, otherwise it returns `0`.

### 3.2.2 Finding the value of $k$

Let $d$ be the number of decimal places which we wish to get an approximation for $S(1,\infty)$. Let $k = 0$, $S = 0$ and $T = \pi^2/6$. We can enumerate over values of $k$ in turn, adding $(-1)^{\Omega(k)}/k^2$ to $S$, and subtracting $1/k^2$ from $T$. If $S - T$ and $S + T$ round to the same value up to $d$ decimal places, then we have an approximation for $S(1,\infty)$ up to $d$ decimal places, by considering the first $k$ terms.

### 3.2.3 Sum of Reciprocal Squares with Prime Factors

The implementation of `SumPF` in MATLAB is as follows.

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

S = 0;
T = pi^2 / 6;
k = 0; d = 1;
PrimeOmega = @(n) (n ~= 1) * length(factor(n));

fprintf('%-8s %-8s %8s\n', 'Terms', 'Value', 'Accuracy');
while (k < 1000000)
  if (round(S - T, d) == round(S + T, d))
    fprintf('%-8d %8f %-8d\n', k, round(S, d), d);
    d = d + 1;
  end
  k = k + 1;
  S = S + (-1)^PrimeOmega(k) / k^2;
  T = T - 1 / k^2;
end
end
```

## 3.3 Analysis

```
>> SumPF
Terms    Value    Accuracy
123      0.700000 1
334      0.660000 2
2102     0.658000 3
42353    0.658000 4
728565   0.657970 5
```

**Example 3.10.** By considering the first 2102 terms of the sum of reciprocal squares with prime factors, we get that $S = 0.6580$ to 4 decimal places. Similarly, we get that $S = 0.65797$ to 5 decimal places, by considering 728565 terms.

### 3.3.1 Alternate Approach

Our implementation of `SumPF` makes use of the sum of reciprocal squares, which was shown to be $\pi^2/6$. As a result, our implementation would not be suitable to compute an approximation for an arbitrary sum. Alternatively, we could derive an approach that deduces such bounds without knowledge of another summation.

# 4 Appendix

## 4.1 Cubic Taxicab Number

**Listing 1: CubicTaxicabNum.m**

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

ctn = N;
while (~IsCubicTaxicabNum(ctn))
    ctn = ctn + 1;
end
end
```

**Listing 2: IsCubicTaxicabNum.m**

```
function isTaxicab = IsCubicTaxicabNum(N)
% ISCUBICTAXICABNUM    Returns isTaxicab = 1, if N is a taxicab number
%                      Returns isTaxicab = 0, otherwise

nPairs = 0;
left = 1;
right = floor(nthroot(N, 3));                    % Claim 1.4

while (left <= right)
    sum = left^3 + right^3;
    if (sum == N)
        nPairs = nPairs + 1;
        if (nPairs == 2)                         % If we have found the
            isTaxicab = true;                    % second pair, we are done
            return;
        end
        left = left + 1;                         % Observation 1.5
        right = right - 1;                       % Observation 1.5
    elseif (sum < N)
        left = left + 1;                         % Observation 1.6
    else
        right = right - 1;                       % Observation 1.7
    end
end
isTaxicab = false;
end
```

## 4.2 Cubic Taxicab Number (Matrix Based Approach)

This is an alternate implementation for the Cubic Taxicab Number problem. The idea here is to compute a multiplication table for cubic numbers and count how many times $N$ occurs in the multiplication table. If $N$ appears at least 4 times in the multiplication table, then $N$ is a cubic taxicab number. Notice that we require $N$ to appear 4 times as

Although this implementation is compact in the sense that it can be written in 7 lines, it requires a large storage overhead to store the multiplication table.

**Listing 3: MatrixCubicTaxicabNum.m**

```
function ctn = MatrixCubicTaxicabNum(N)
ctn = N;
isCtn = @(n, s) nnz(((1:s).^3' * ones(1, s) + (1:s).^3) == n) >= 4;
while (~isCtn(ctn, floor(nthroot(ctn, 3))))
  ctn = ctn + 1;
end
end
```

## 4.3 Catalan's Constant

**Listing 4: RatAppCat.m**

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

G = 0.915965594177219;
p = 0; q = 1;                          % The best (p, q) pair so far
minDelta = abs(G - p / q);             % The difference between p / q and G

for q0 = 1 : N
  p0 = round(G * q0);                  % Observation 2.3
  if (p0 + q0 > N)                     % Claim 2.4
    return
  end
  delta = abs(G - p0 / q0);
  if (delta < minDelta)                % Update if current pair is better
    minDelta = delta;                  % than the best pair we have seen
    p = p0; q = q0;
  end
end
end
```

## 4.4 Sum of Reciprocal Squares with Prime Factors

**Listing 5: SumPF.m**

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

S = 0;
T = pi^2 / 6;
k = 0; d = 1;
PrimeOmega = @(n) (n ~= 1) * length(factor(n));

fprintf('%-8s %-8s %8s\n', 'Terms', 'Value', 'Accuracy');
while (k < 1000000)
  if (round(S - T, d) == round(S + T, d))
    fprintf('%-8d %8f %-8d\n', k, round(S, d), d);
    d = d + 1;
  end
  k = k + 1;
  S = S + (-1)^PrimeOmega(k) / k^2;
  T = T - 1 / k^2;
end
end
```