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
```