Advanced PiE Assignment Report

191158510 - Programming in Engineering (PiE)

Complexity, Debugging, Optimizing

Kirtan Premprakash Patel - s2935848

UNIVERSITEIT TWENTE.

Contents

1 Exercise 1 - Debugging : Errors and Corrections

The given code contained several errors which were corrected to obtain the required output.

- **Input Check**

```
% original code
if length(N) == 1
    ...
```

a scalar input has length = 1, the error must be printed when the length of input is not 1. The original code prints the error when a scalar is input. It needs to be corrected to print the error when the input is not a scalar.

```
% corrected code
if length(N) ~= 1
    ...
```

```
% original code
if primes(k) ~= 0
    primes((k*k):k:length(primes)) = 0;
end
```

- **Array Indexing**

MATLAB uses 1-based index i.e. the first element holds index = 1. The for-loop in which we check for primes has a loop variable k that starts from **2** and goes upto sqrt(N). The list of primes which we generate before crossing out numbers from it, goes from 2 upto N. This means that number 2 holds index = 1, number 3 holds index = 2 and so on. Thus, we can not directly use the loop variable as the index to call and change that numeric value.

The number *num*, is actually at an index *num-1*, and hence the loop variable needs to be reduced by 1 to get an index which is used to access the number corresponding to the loop variable. This error in indexing causes the integers after a prime number to be crossed out instead of the prime-number itself.

- **Logical Error**

the numbers to be crossed out are being considered from k^2 to *length(primes)*. primes has been initialized from 2 to N and hence the length of primes is N-1, thus the last number in the initialized array (the input number N itself) is not checked for being a multiple of the previous prime-numbers.

Thus, the array generation to cross out multiples must be done from k^2 to N, followed by the index correction mentioned in the above point.

```
% corrected code
if primes(k-1) ~= 0
    primes(((k*k):k:N)-1) = 0;
end
```

2 Exercise 2 - Complexity : Analysis and Verification

2.1 Algorithm

Algorithm 1 A simple algorithm for the Matrix product

```
for i = 1 to a do
  for j = 1 to c do
    P_{i,j} ← 0
    for i = 1 to b do
      P_{i,j} ← P_{i,j} + M_{i,k}.N_{k,j}
    end for
  end for
end for
```

2.2 Complexity Analysis

2.2.1 Time Complexity

The innermost loop is a for-loop which runs from 1 to b. The code written inside the body of the innermost for-loop has complexity O(1). Thus the innermost for-loop has a complexity O(b).The 2 other for-loops in the algorithm have complexities O(a) and O(c).

All the 3 loops are nested and hence the total time complexity of the algorithm is :

$$\boxed{O(a).O(b).O(c) = O(a.b.c)} \tag{1}$$

2.2.2 Memory Complexity

In the 2 outer-most loops of the algorithm, memory is being allocated, and they form a matrix with dimensions aXc, thus, the memory complexity of the algorithm is :

$$\boxed{O(a).O(c) = O(a.c)} \tag{2}$$

2.3 MATLAB Verification

Matrix multiplication of 2 nXn random matrices has been carried out many 1e6 times and the cpu runtime has been noted. Since all times will be scaled, it will not affect the time relation, and we will get better values. To find the time complexity, best fit curves were plotted using the MATLAB polyval().

n	0	10	50	100	150	200
cputime (1e-6)	0	0.85	16.49	77.06	348.75	716.57

All values of n,cputime were used to plot the best fit curve , **except the last value [200,716.57]**.This value will be used to check which ones of the curves is more accurate.

It is found that $\boxed{\mathbf{O(n^2)}}$ is the better approximation of the time complexity of MATLAB Matrix Multiplication, which is lower than estimated earlier for matrix multiplication ($O(n^3)$)

Using the quadratic fit, it is estimated that 2 nXn matrices can be calculated under 10 seconds for a maximum value of $\boxed{n = \mathbf{19758}}$, although there might be memory limitations in doing so.

3 Exercise 3 - Optimization

3.1 Optimized Code

The old code has multiple steps involved including :

1. calculating length of input vector and allocating it to a variable
2. initialize an array of zeroes with dimensions[n,1]
3. retrieve x,y,z for each point
4. calculate distance using normal mathematical operations
5. assign the calculated
6. getting minimum value from the array of distance

These steps are performed seperately and hence the code does not utilize the power of matrix calculations that MATLAB has to offer.

The entire code can be reduced to 2 lines of code which uses MATLAB's power of matrix calculations.

```
function d = minDistance(x,y,z)
% Given a set of 3D points specified by column vectors x,y,z, this
% function computes the minimum distance to the origin

d = sqrt(x.^2 + y.^2 + z.^2);

% get minimum distance
d = min(d);
```

3.2 Runtime Reduction Verification

The execution speed has increased significantly. This can be seen using the *tic-toc* functions of MATLAB. To see the evident difference in the runtimes, the computation has been run 1e5 times for different length of arrays(n) and the runtimes have been stated below.

n	0	100	200	300	400	500
original_code runtime	0	0.206987	0.300897	0.392742	0.473406	0.546095
optimized_code runtime	0	0.193965	0.282835	0.351725	0.456697	0.509601

Thus the new code is optimized and uses the power of matrix multiplication of MATLAB as much as possible.

4 Exercise 4 - Complexity : Analysis and Verification

4.1 Complexity Analysis

4.1.1 Time Complexity

The main function perms() calls permsr() to carry out the calculations, hence time complexity of permsr() is the time complexity of perms().

The innermost loop (inside permsr()) is a for-loop which runs from *nn*-1 to 1 and hence has complexity. But the inner loop us dependent on the outer loop.

nn	2	3	4	...	...	n
inner-loop iterations	1	2	3	...	...	n-1
total iterations	1 + 2 + 3 + ... (n-1) $=\frac{n.(n+1)}{2}$ -2 = O(n^2)					

Thus, the total time complexity can be denoted as : $O(n^2)$

4.1.2 Memory Complexity

In permsr(), in the inner-most loop of the algorithm, memory is not being allocated.It is being allocated only in the outer loop which runs from 2:n and they form a matrix with dimensions $nn.mXnn$.

nn	2	3	4	5	...	n
Psmall size	1	2	6	24	...	$n!$
matrix size	2x2	6x3	24x4	120x5	...	$n!$xn
memory allocation	$n!xn$					

Thus, the memory complexity of the algorithm is : $O(n.n!)$

4.2 MATLAB Verification

The operation has been carried out many 1e5 times and the cpu runtime has been noted. Since all times will be scaled, it will not affect the time relation, and we will get better values. To find the time

size of input array	0	1	3	4	5	7
cputime (1e-5)	0	1.24	2.34	6.12	8.40	15.65

complexity, best fit curves were plotted using the MATLAB polyval(). All values of n,cputime were used to plot the best fit curve , **except the last value [7,15.65]**.This value will be used to check which ones of the curves is more accurate.

It is found that **O(n^2)** is the better approximation of the time complexity of the MATLAB perms() function, which is the same as estimated earlier.

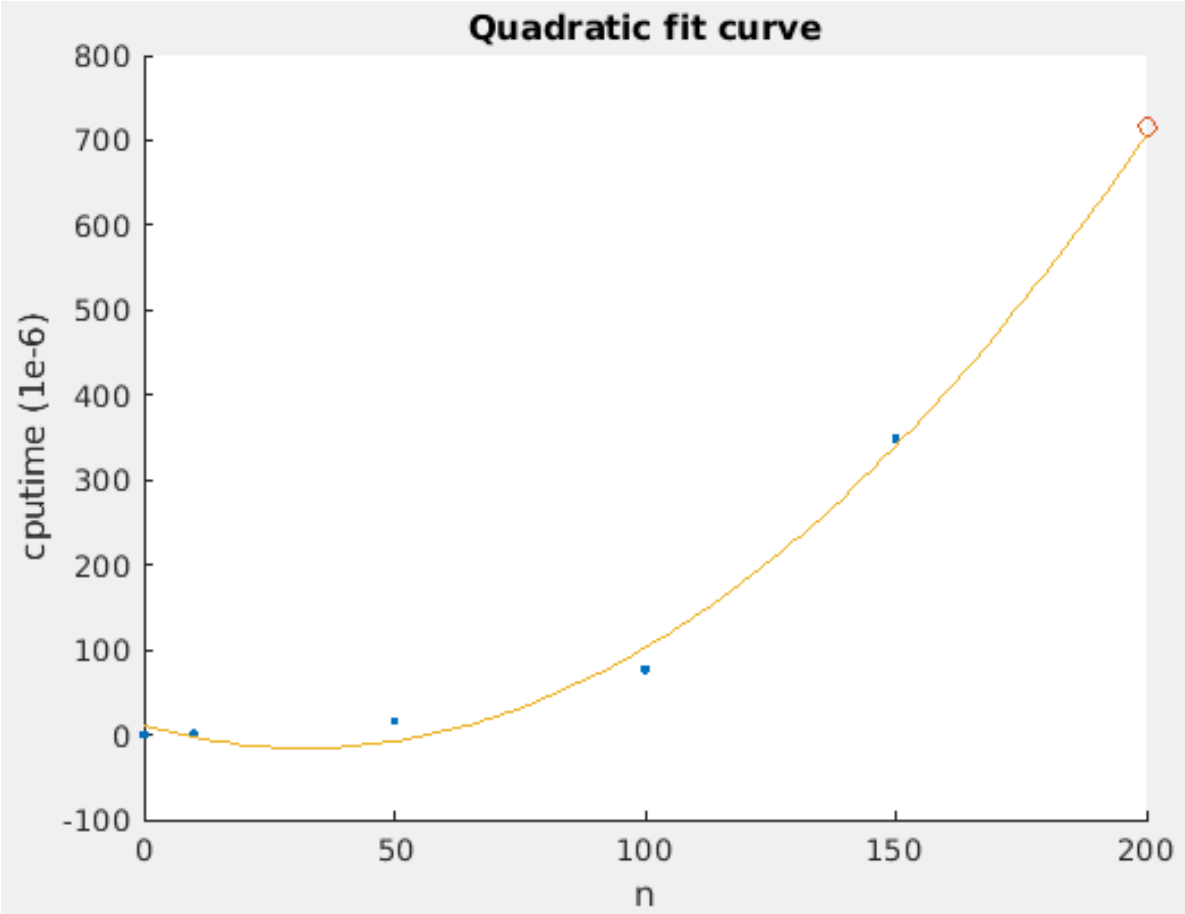
Using the quadratic fit, it is estimated that permutations can be calculated under 10 seconds for a maximum size of array of $n = \mathbf{1712}$, although there might be memory limitations in doing so.

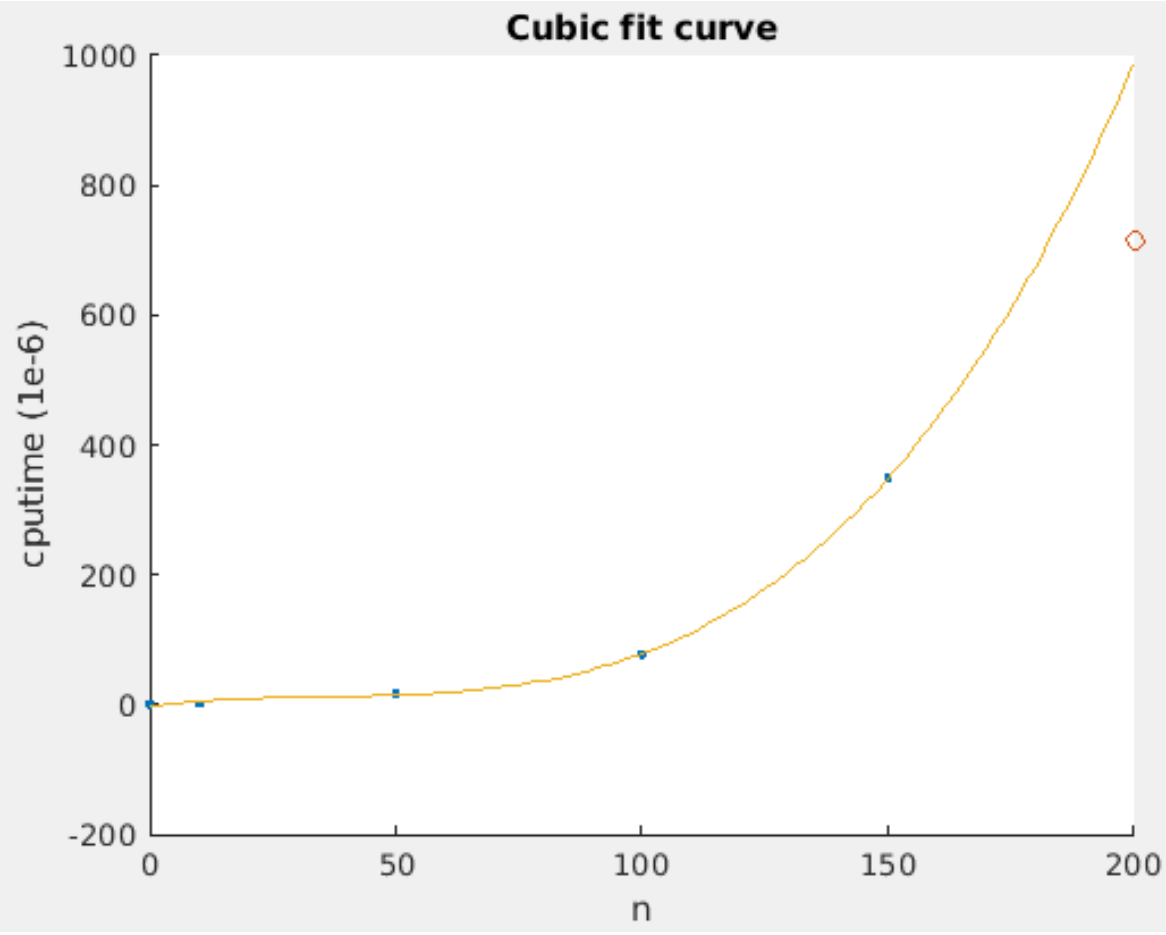
Appendix

A Exercise 1 : Corrected Code

```
function primes = listofprime(N)
%   listofprimes(N) Generates a list of prime numbers from 2 to N.
%
%   Input: scalar value N
%   Output: a row vector of prime numbers from 2 to N.

% check if input is a scalar
if length(N) ~= 1
  error('N must be a scalar');
end

% check if input is greater or equal to 2; else return empty array
if N < 2,
  primes = [];
end

% create an array for the numbers from 2 to N
primes = 2:N;

% loop through all integers from 2 to sqrt(N)
for k = 2:sqrt(N)
  % setting a value of the array to 0 indicates the value is crossed out
  % check if the number k is not crossed out

  % since array starts from 2 and indexing starts from 1
  % number in array is not equal to the index

  if primes(k-1) ~= 0
    % cross out all multiples of k starting from k^2, i.e.
    % cross out k^2, k^2+k, k^2+2*k, ...
    primes(((k*k):k:N)-1) = 0;
  end
end
disp(primes)
% return all values that are not crossed out
primes = primes(primes>0);
end
```

B Exercise 2 : Best Fit Curves





C Exercise 4 : Best Fit Curves