Answer on Question #42871, Programming, Mat LAB | Mathematica | MathCAD | Maple

a) To accept two numbers from the user; Display all prime numbers between these two numbers.

Solution:

A prime number is a natural number which has exactly two distinct natural number divisors: 1 and itself.

To find all the prime numbers less than or equal to a given integer n by Eratosthenes' method:

1. Create a list of consecutive integers from 2 through n: (2, 3, 4, ..., n).
2. Initially, let p equal 2, the first prime number.
3. Starting from p, enumerate its multiples by counting to n in increments of p, and mark them in the list (these will be 2p, 3p, 4p, etc.; the p itself should not be marked).
4. Find the first number greater than p in the list that is not marked. If there was no such number, stop. Otherwise, let p now equal this new number (which is the next prime), and repeat from step 3.

Here is the pseudocode

```
fill your array with numbers from 2 to N

p=2
while p<=n
    for i=2*p, while i<=N, increment i=i+p
        mark element as 0
    end for
    increment p by 1
end while

print all array elements that are not 0
```

The implementation in Matlab is

```
disp('Enter two numbers(intervals)');
a = input('Enter minimum integer number > 1: ');
b = input('Enter maximum integer number: ');
% Eratosthenes' method
%1.Create a list of consecutive integers from 2 through b
primes = 2:b;
% 2.Initially, let p equal 2, the first prime number.
p=2;
%3. Starting from p, enumerate its multiples by counting to b in increments
% of p, and mark them by '0' in the list (these will be 2p, 3p, 4p, etc.; the p
% itself should not be marked).
```

```
while (p <= b)
   for i = 2*p:p:b
      primes(i - 1) = 0;
   end;
   p = p + 1;
end

fprintf('Prime numbers between %d and %d are: ', a, b);

for j = a-1:b-1
if primes(j)~=0
   fprintf('%d, ', primes(j))
end
end
```

b) To accept two numbers from the user and display perfect numbers between these two numbers.

Solution:

In number theory, a perfect number is a positive integer that is equal to the sum of its proper positive divisors, that is, the sum of its positive divisors excluding the number itself (also known as its aliquot sum).

Example: 6 is a perfect number: 1 + 2 + 3 = 6 == 6

8 is not 1 + 2 + 4 = 7 ~= 8

The function perfect checks to see if the given number is "perfect":

```
function perfect = isperfect(test_value)
 % this will keep track of the sum of all integer divisors
 sum = 0;

 % we check all possible divisors
 for (divisor = 1 : test_value - 1)
  div_result = test_value / divisor;

  if (div_result == floor(div_result))
     % this is an integer divisor, so add it to the sum
     sum = sum + divisor;
  end

 end

 % now, does the sum equal the test_value?
 if (sum == test_value)
    perfect = 1;
 else
    perfect = 0;
 end
```

The program is

```
disp('Perfect numbers!');
disp('Enter two numbers(intervals)');
a = input('Enter minimum integer number > 1: ');
b = input('Enter maximum integer number: ');

fprintf('Perfect numbers between %d and %d are: ', a, b);

for i = a:b
if isperfect(i)~=0
   fprintf('%d, ',i)
end
end
```

c) To display Armstrong numbers from 100 to 999 inclusive.

Solution:

An Armstrong number (sometimes called also narcissistic numbers) of three digits is an integer such that the sum of the cubes of its digits equals the number itself.

For example, 407 is an Armstrong number since

$$4^3 + 0^3 + 7^3 = 407$$

We are going to take three digits and iterate them to get all the possible combinations to achieve integers from 100 to 999. (Those first digits are obviously hundreds, tens and units). Then, we perform the convenient computations to find the number itself and the possible Armstrong number. If they match, we display the results.

Thus, program is

```
fprintf('Armstrong numbers between 100 and 999 are: \n');
% We use a as hundreds
for a = 1 : 9
   % We use b as tens
   for b = 0 : 9
      % We use c as units
      for c = 0 : 9
         % n is the resulting number
         n = a*100 + b*10 + c;
         % an is the Armstrong number
         an = a^3 + b^3 + c^3;
         % We compare the number with its AN
         if n == an
            % and display if they're the same
            fprintf('%d, ',an)
         end
      end
   end
end
```