머리말

천 번째 소수는 무엇일까? 숫자 985564500을 소인수분해 하면 어떻게 될까? 영화 [박사가 사랑한 수식]을 보면 친화수(Amicable number, 우애수)가 나오는데, 직접 프로그램을 작성해서 확인할 수 있을까? 이렇게 직접 손으로 계산하기가 힘든 수론에 관한 문제들을 컴퓨터 프로그램을 이용하여 계산하고 자신만의 알고리즘을 작성하는 방법을 알아보자.

키워드 : 최소공배수, 소인수분해, 에라토스테네스의 체, 친화수, 쌍둥이 소수, 골드바흐의 추측

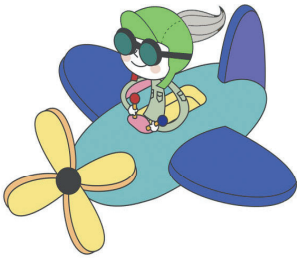
차 례

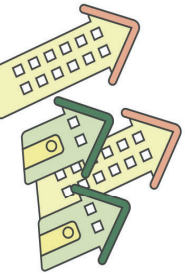
Contents

목표: 소수와 약수에 관련된 다양한 문제들을 컴퓨터 코딩으로 해결하기

프로그램의 특성상 업그레이드 될 경우에는 프로그램 설치 및 명령문이 변경될 경우도 있습니다. 책의 내용에 대해 질문이나 조언이 있는 경우, 블로그 (http://blog.naver.com/cfdkim)에 문의해주시면 감사하겠습니다.

이 책 뒷부분에 있는 박경미 교수님과 정기철 교수님의 참고문헌을 참조 하면 더욱 풍부한 내용을 학습할 수 있습니다.





코딩수학

Chapter 1

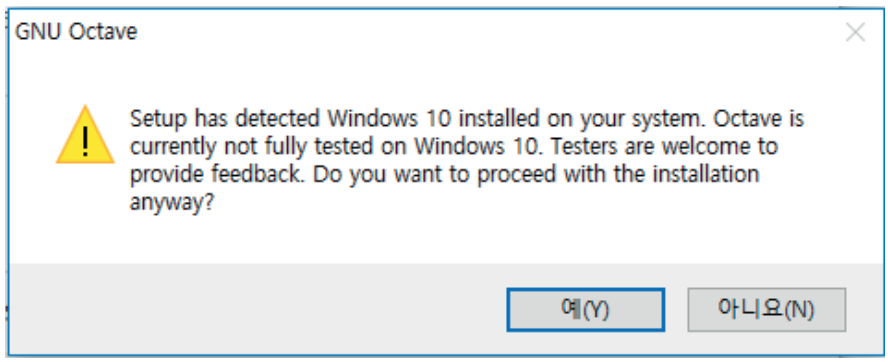
옥타브 설치 및 시작 방법

프로그램을 다운받아보자

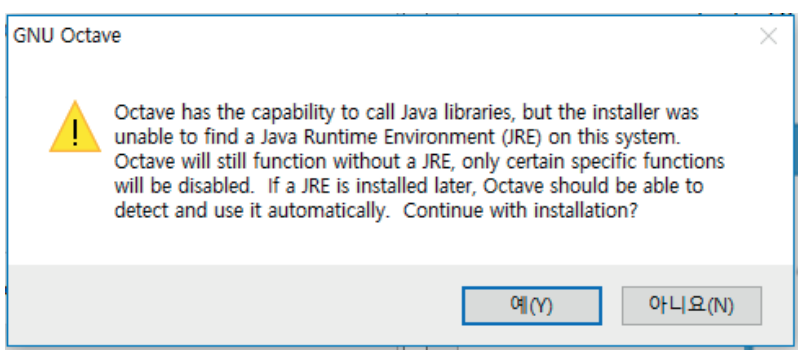
1. 프로그램 다운로드

1.1 옥타브(Octave) 홈페이지에
https://www.gnu.org/software/octave/
접속하여 Download버튼을 클릭한다.

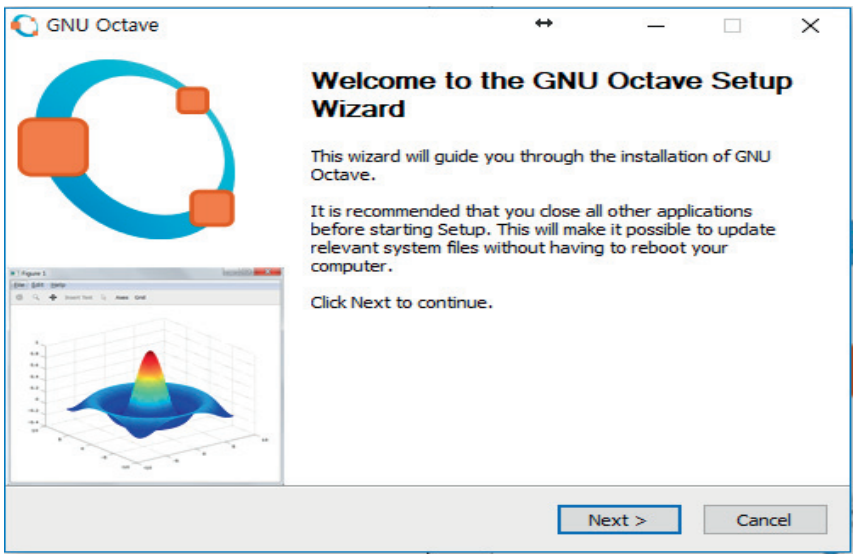
1.2 Download를 클릭하면 기본 화면이 GNU/Linux로 세팅이 되어있다.



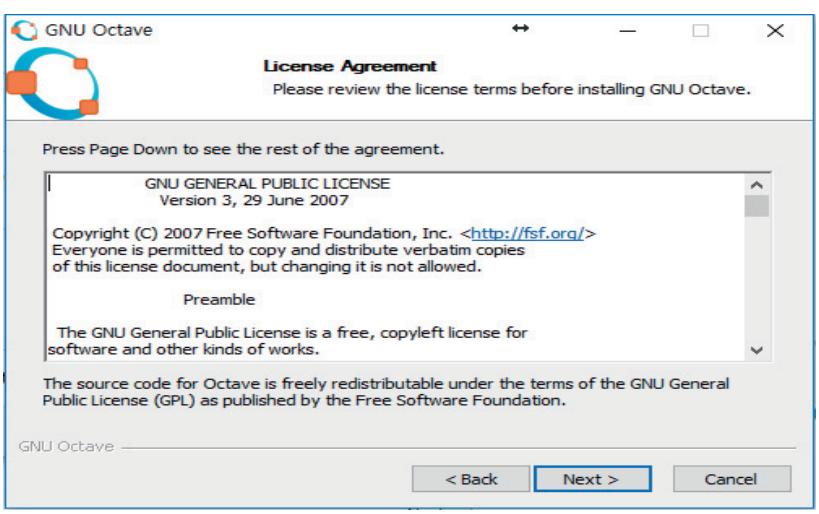
다음 경고 메시지는 java Runtime Environment가 기존에 설치되지 않았다는 것이다. '예(Y)'를 클릭하여 다음 단계로 넘어가자.



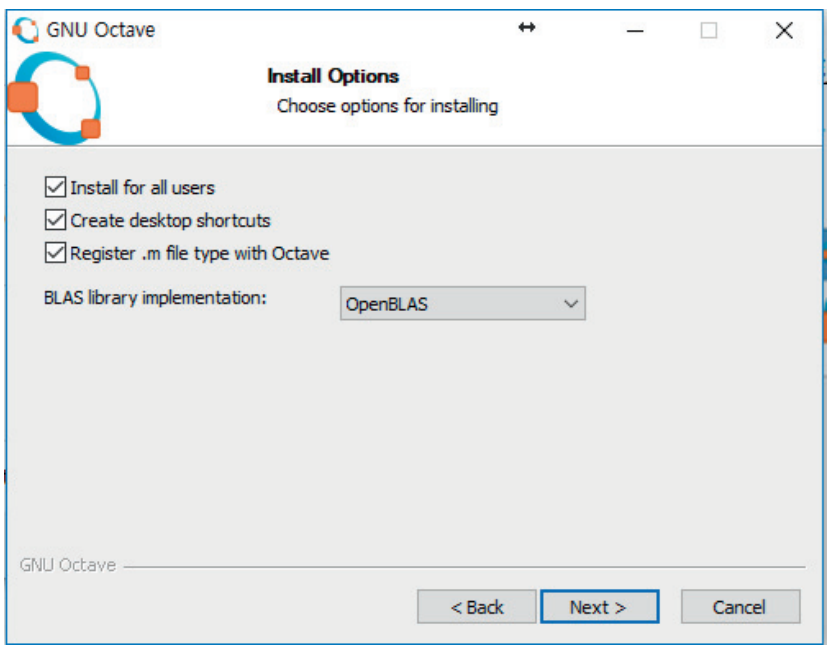
2.3 이제 본격적으로 Octave 설치가 된다. 'Next >'를 클릭하여 다음 단계로 넘어가자.



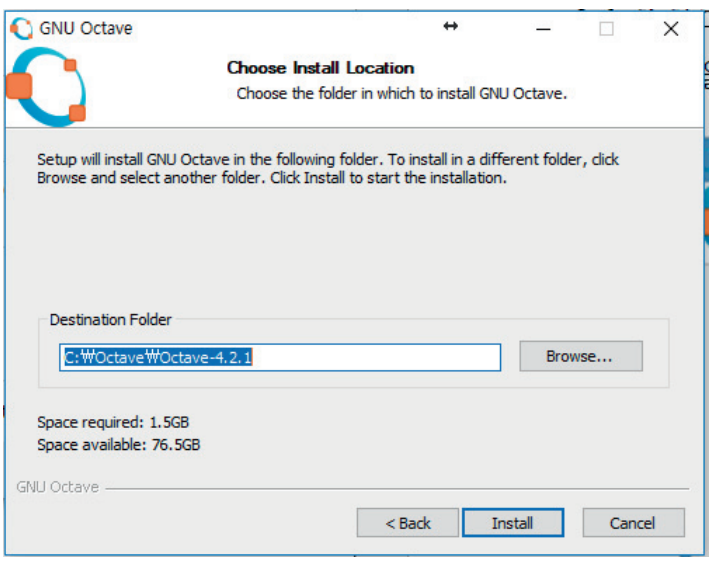
2.4 프로그램 라이센스에 관한 내용이다. ‘Next >’를 클릭하여 다음 단계로 넘어가자.

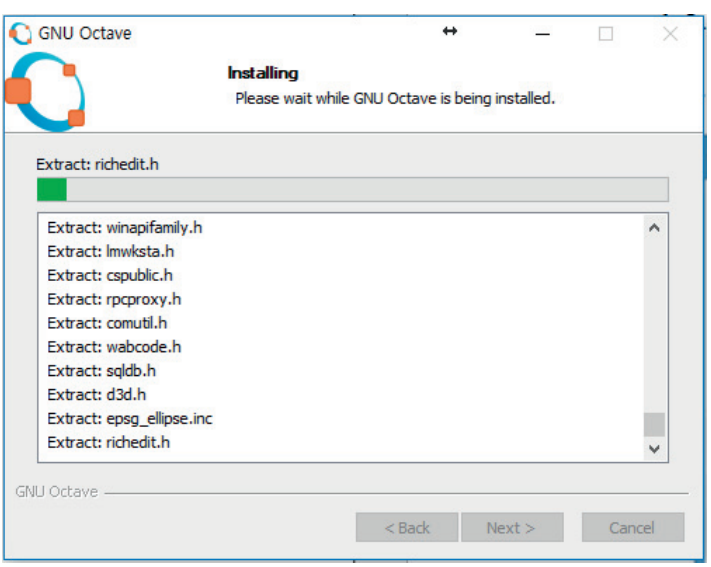


2.5 프로그램 설치에 관한 옵션선택이다. 기본 값으로 두고, 'Next >'를 클릭하여 다음 단계로 넘어가자.

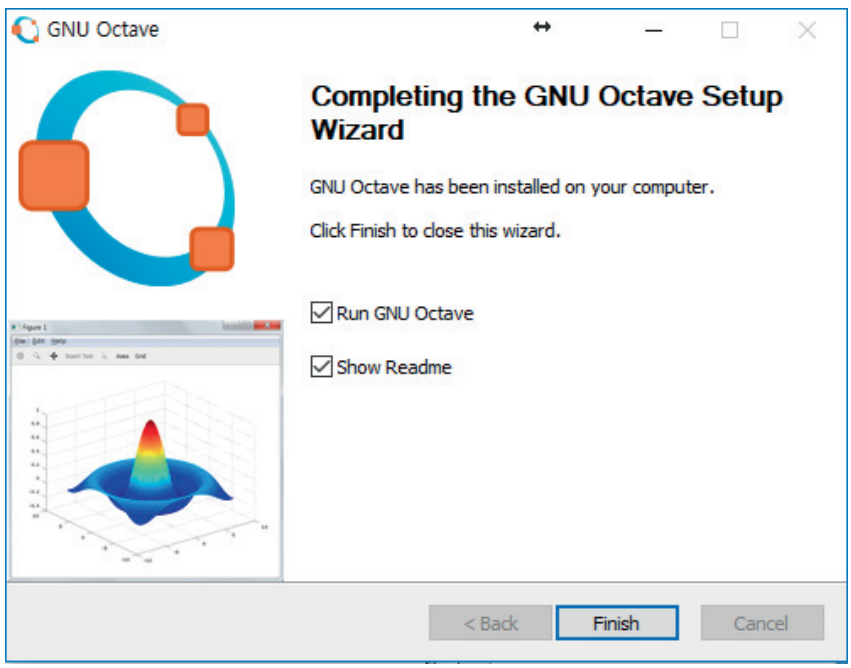


2.6 프로그램 설치위치를 정하는 창이다. 기본 설정으로 두고 'Install'을 클릭하여 설치하자.

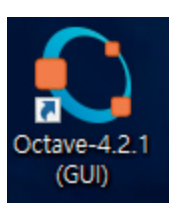




2.7 다음과 같은 화면이 나올 경우, 정상적으로 설치가 완료된 것이다. “Finish”를 클릭하여 프로그램 설치를 종료하자.

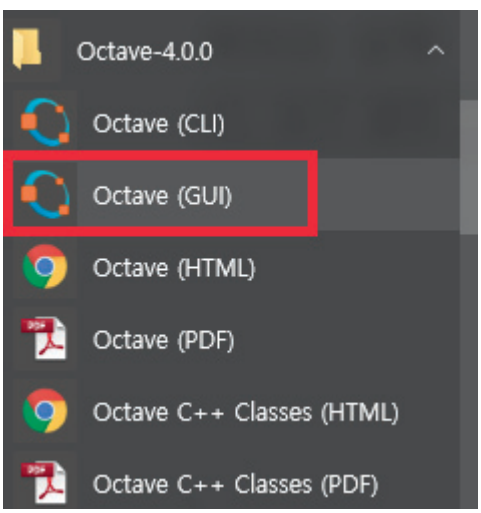


2.8 프로그램 설치를 마치면 Octave 프로그램이 실행되지만 종료하고 다시 시작하자. 바탕화면을 보면 다음과 같은 Octave GUI 아이콘이 있다. 더블클릭하여 프로그램을 실행하자.

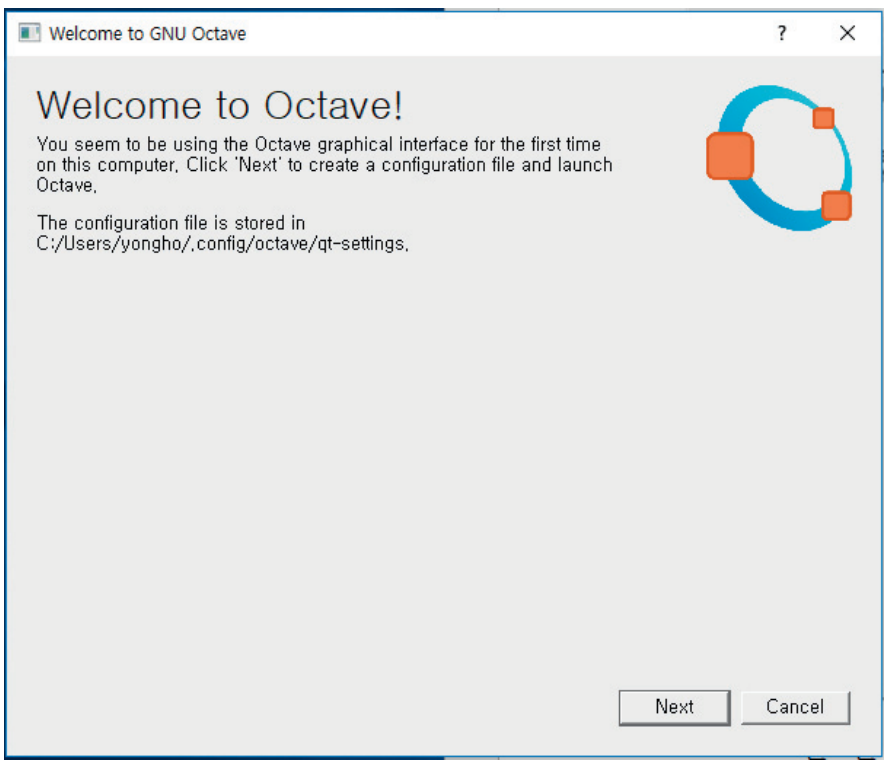


2.9 초기 설정

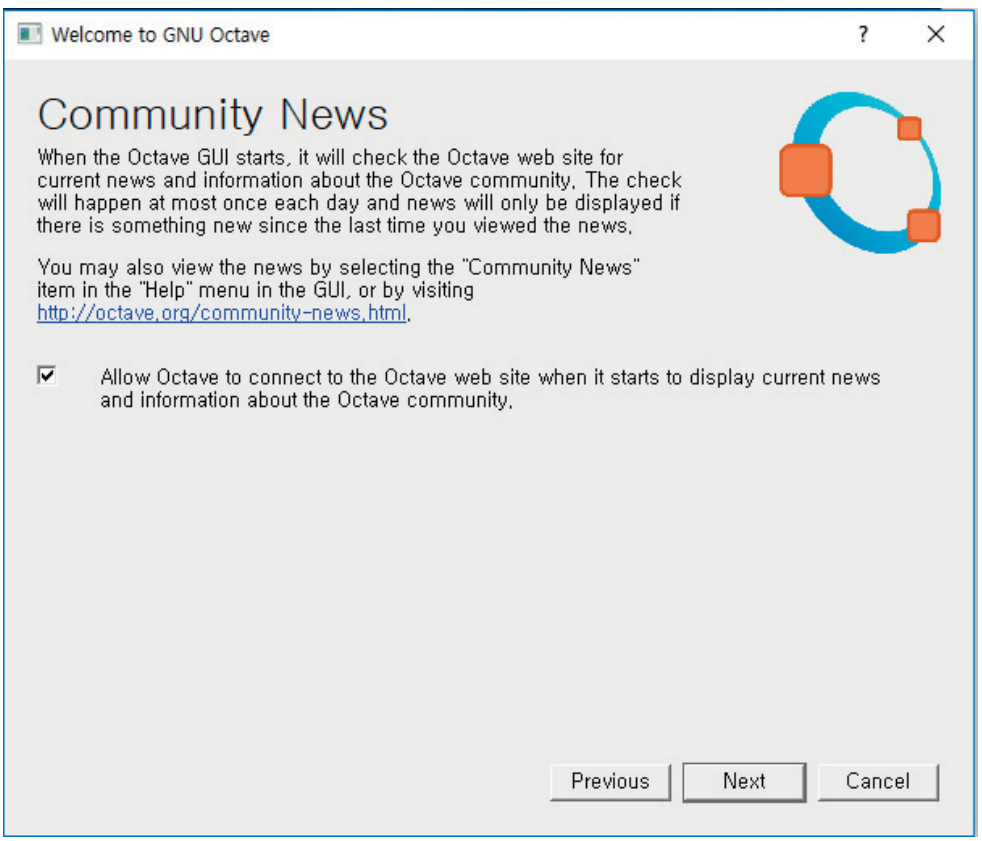
설치를 완료하고 프로그램 목록에 있는 Octave (GUI)를 실행한다. 초기 설정은 최초에 한번만 실행하게 된다.



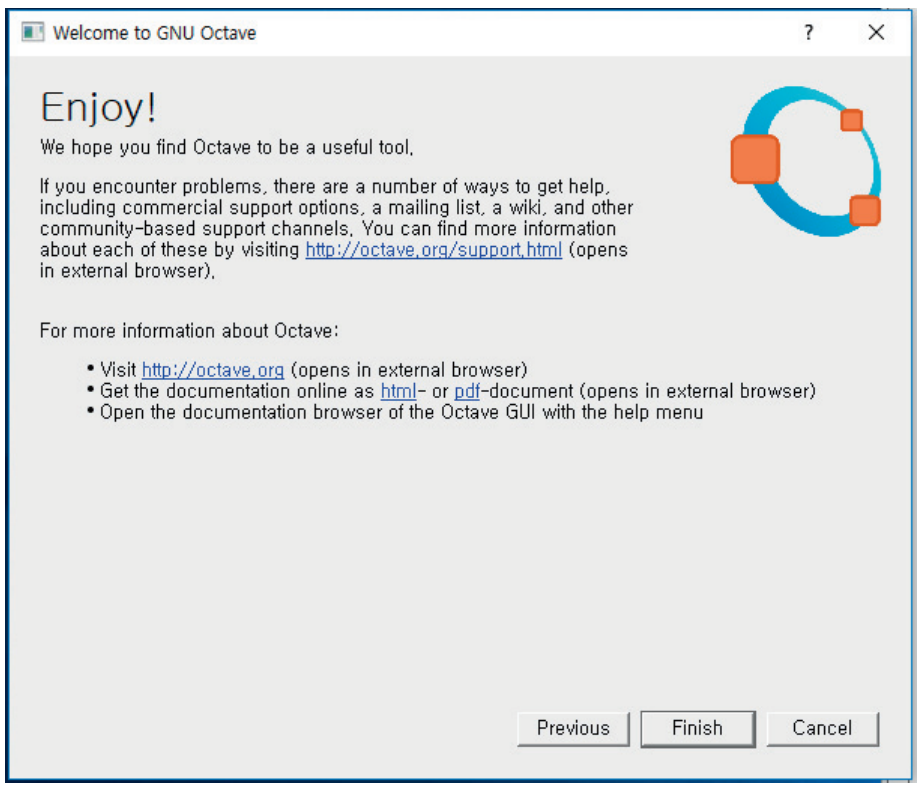
‘Next >’를 클릭하여 다음 단계로 넘어가자.

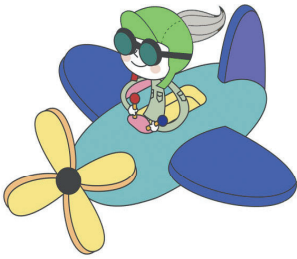


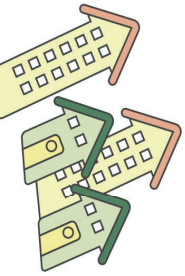
'Next >'를 클릭하여 다음 단계로 넘어가자.



‘Finish’를 클릭하여 기본설정을 완료한다.







코딩수학

Chapter 2

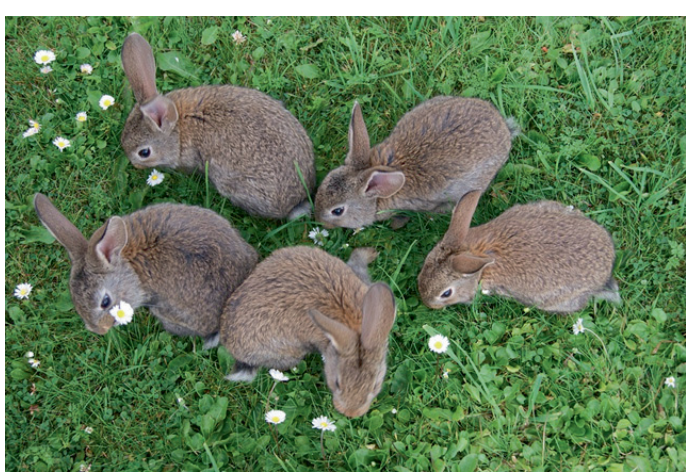
옥타브를 이용하여 소수와 약수 관련 문제들을 풀어보자

피보나치 수열

피보나치 수열은 첫 번째 항과 두 번째 항의 값이 1이고 이후의 항들은 이전의 두 항을 더한 값으로 정의되는 수열이다. 이 수열은 12세기 말 이탈리아 수학자 레오나르도 피보나치가 제안한 수열로 항 쌍의 토끼가 새끼를 계속 낳을 경우 토기의 수를 나타낸 것이다. 토끼 쌍을 숫자로 나타내면 다음과 같다. 1, 1, 2, 3, 5, 8, 13, 21, 34, ... 파보나치 수열은 아래의 점화식으로 정의되는 수열이다.

$$F_1 = F_2 = 1, \qquad \text{일 때}$$
$$F_n = F_{n-1} + F_{n-2}, \quad n \geq 3$$

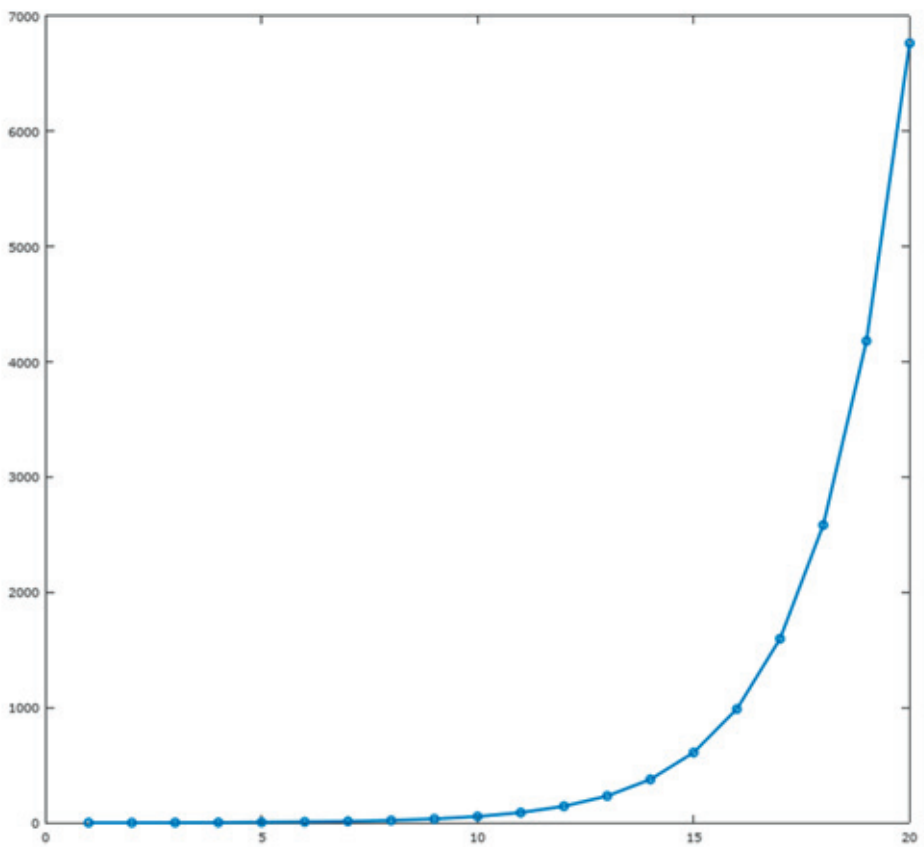
이 수열의 20항까지를 프로그램을 작성하여 구해보자.



코드설명

```
clear;
% 메모리 초기화
F(1)=1;
% 피보나치 수열의 첫째항 값
F(2)=1;
% 피보나치 수열의 둘째항 값
n=20;
% 전체 수열의 항수
fprintf('F(1)=%d ₩n',F(1));
% 명령문 창에 첫째항 출력
fprintf('F(2)=%d ₩n',F(2));
% 명령문 창에 둘째항 출력
for i=3:n
   F(i)=F(i-1)+F(i-2);
% 피보나치 수열의 점화식으로 F(3), F(4), ... , F(n)을 구함
   fprintf('F(%d)=%d ₩n',i,F(i));
% i 번째 수열 F(i)를 출력
end
plot(F,'o-')
% 피보나치 수열을 그림으로 플롯, o-는 심볼로 그림선을 나타낸다.
```

그래프 결과



피보나치 수열을 첫째항부터 20번째 항까지 플롯

명령문 창 결과

```
〉〉 Fibonacci
F(1)=1
F(2)=1
F(3)=2
F(4)=3
F(5)=5
F(6)=8
F(7)=13
F(8)=21
F(9)=34
F(10)=55
F(11)=89
F(12)=144
F(13)=233
F(14)=377
F(15)=610
F(16)=987
F(17)=1597
F(18)=2584
F(19)=4181
F(20)=6765
```

공약수

공약수(common divisor)는 두 정수의 공통된 약수이다.

코드명: CommonDivisor.m

```
clear;
x=30816
y=81696
for divisor=1:min(x,y)
 if mod(x,divisor)==0 && mod(y,divisor)==0
    divisor
 end
end
```

명령문 창 결과

```
〉〉 CommonDivisor
x =  30816
y =  81696
divisor =  1
divisor =  2
divisor =  3
divisor =  4
divisor =  6
divisor =  8
divisor =  12
divisor =  16
divisor =  24
divisor =  32
divisor =  48
divisor =  96
```

명령문 창 결과

```
〉〉 LCommonMultiple
x =   30816
y =   81696
ans =   26224416
```

에라토스테네스의 체

에라토스테네스의 체 (Sieve of Eratosthenes)는 소수를 찾는 간단한 방법이다. 고대 그리스 수학자 에라토스테네스가 발견하였다. 이 방법은 마치 체로 치듯이 수를 걸러낸다고 하여 '에라토스테네스의 체'라고 부른다.

예를 들어, 아래와 같이 1부터 100까지의 자연수가 있다고 하자. 1은 소수가 아니므로 체크하고 2부터 시작하여 2보다 큰 2의 배수는 모두 체크를 한다. 그런 다음 3보다 큰 3의 배수를 모두 체크를 한다. 이런 방법으로 100보다 1작은 99까지 하고 나서 체크가 안 된 수들이 소수이다. 아래의 수들로 직접 해보자.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

코드설명

```
clear;
n=100;
% 1부터 100 까지의 숫자 중에 소수를 찾고 총 개수를 구한다.
x=ones(n,1);
% x는 크기가 n인 벡터이고 모든 원소의 값이 1이다.
x(1)=0;
% 1은 소수가 아니므로 0으로 세팅
for i=2:n-1
% 나누는 수로 2부터 n-1을 사용한다.
  for j=i+1:n
% 체크할 대상 수로 i보다 1큰 수부터 n까지로 한다.
   if mod(j,i)==0
% j가 i로 나누어떨어지는지 체크한다.
   x(j)=0;
% 나누어떨어지는 경우 x(j)의 값을 0으로 세팅
   end
  end
end
for i=1:n
```

```
prime =  41
prime =  43
prime =  47
prime =  53
prime =  59
prime =  61
prime =  67
prime =  71
prime =  73
prime =  79
prime =  83
prime =  89
prime =  97
TotalNumber =  25
```

n이 작은 경우에는 위의 알고리즘이 괜찮으나 n이 큰 경우에는 계산속도가 느리다. 좀 더 빠른 알고리즘은 다음 부분을

```
for i=2:n-1
% 나누는 수로 2부터 n-1을 사용한다.
```

아래와 같이 변경하는 것이다.

```
for i=2:[√n]
% 나누는 수로 2부터 [√n]을 사용한다.
```

새로운 방법이 왜 되는지에 대해서는 곰곰이 생각해보자. 여기서 [m] 기호는 m을 넘지 않는 가장 큰 정수이다. 옥타브에서는 floor() 명령어를 사용한다.

코드설명

```
clear;
q=234613;
% 임의의 자연수
flag=1;
% q가 소수인지 아닌지를 알려주는 수
for i=2:q-1
% q를 나눌 수
  if (mod(q,i)==0)
% q가 i로 나누어떨어지는지를 판별
    printf('%d is not a prime number ₩n',q);
% q가 소수가 아니라고 출력
    flag=0;
% q가 소수가 아니라고 저장
    break;
% q가 소수가 아니므로 루프를 빠져나옴
  end
end
if flag==1
% q가 2부터 q-1까지의 수로 나누어떨어지지 않으면
flag=1로 그대로 유지가 되었으므로 q는 소수
```

```
    printf('%d is a prime number ₩n',q);
% q가 소수라고 출력
end
```

명령문 창 결과

```
〉〉 PrimeOrNot
234613 is a prime number
```

소인수 분해

소인수 분해(prime factorization)는 합성수를 소수의 곱으로 나타내는 방법을 말한다. 합성수(合成數, composite number)는 1과 자기 자신이 아닌 다른 자연수의 곱으로 나타낼 수 있는 자연수를 의미한다. 1보다 큰 모든 정수는 소수이거나 합성수이다.

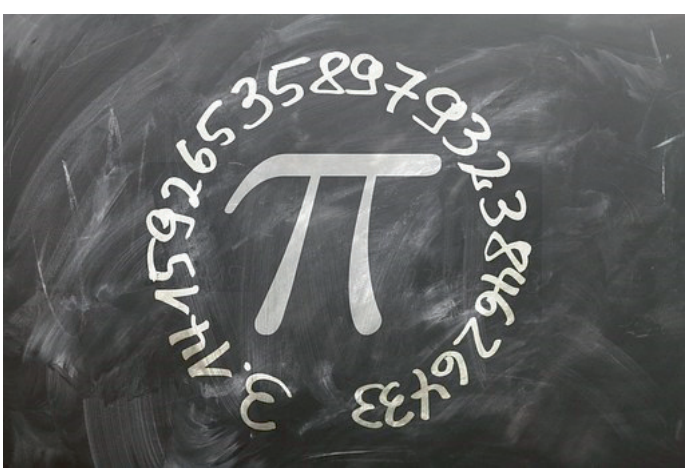
코드명: PrimeFactorization.m

```
clear;
format long
n=985564500
q=2;
count=1;
while q <= n
   if mod(n,q)==0
      n=n/q;
      x(count)=q;
      count=count+1;
   else
      q=q+1;
```

```
s
% s의 값이 주어진 수 n과 일치함을 체크할 수 있다.
```

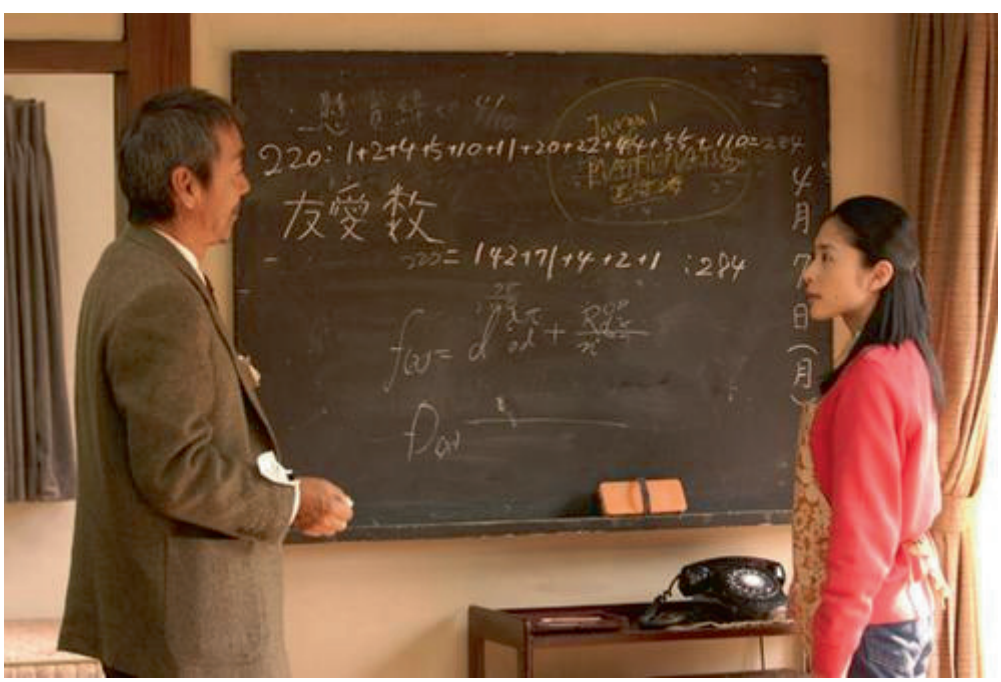
명령문 창 결과

```
〉〉 PrimeFactorization
n =   985564500
x =   2    2    3     5    5     5    79    8317
s =   985564500
```



친화수

친화수(親和數, Amicable numbers, 친애수)는 두 수의 쌍이 있어, 어느 한 수의 진약수(약수 중 자기 자신을 제외한 것)를 모두 더하면 다른 수가 되는 것을 말한다. 예를 들어 220과 284는 친화수이다. 220의 진약수는 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110이고 모두 더하면 284가 된다. 반대로 284의 모든 진약수 1, 2, 4, 71, 142를 모두 더하면 220이 된다.



영화 [박사가 사랑한 수식]의 스틸컷

코드설명

```
clear;
n=3000;
% 3000까지의 친화수를 구한다.
for i=2:n
% 2부터 n까지의 수들 중에서 친화수를 체크한다.
  x=0;
% i의 진약수들을 전부 더하기 위해서 0으로 초기화
  for j=1:i-1
% i의 진약수를 구하기 위해서 1부터 i-1까지 검토
    if mod(i,j)==0
% j가 i의 약수인지를 검토한다.
      x=x+j;
% 약수를 누적해서 합해줌
    end
  end
  if x>1
% x가 1이면 i는 소수임으로 친화수가 될 수 없다. i가
소수가 아니라면 계속진행
    y=0;
% x의 상대 친화수를 체크하기 위해서 y를 0으로 세팅
```

```
    for j=1:x-1
% x의 진약수를 구하기 위해서 1부터 x-1까지 검토
      if mod(x,j)==0
% j가 x의 약수인지를 검토한다.
        y=y+j;
% 약수를 누적해서 합해줌
      end
    end
    if i==y && x>i
% i와 i의 진약수의 합인 x의 진약수의 합 y가 같고 x가 i보다 크다면 친화수 쌍을 출력
      [i x]
    end
  end
end
```

명령문 창 결과

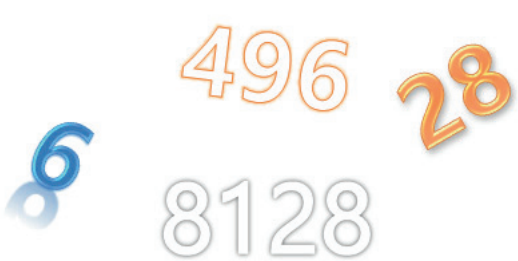
```
>> AmicableNumber
ans =   220    284
ans =  1184   1210
ans =  2620   2924
```

코드설명

```
clear;
n=10000;
% 10000까지 수들 중에서 완전수를 찾기
for i=2:n
% 2부터 n까지의 수들 중에서 완전수를 체크한다.
  x=0;
% i의 진약수들을 전부 더하기 위해서 0으로 초기화
for j=1:i-1
% j가 i의 약수인지를 검토한다.
  if mod(i,j)==0
    x=x+j;
% 약수를 누적해서 합해줌
  end
end
  if i==x
% 만약 i와 i의 진약수들을 모수 합한 값이 같으면
    i
% 진약수 i를 출력
  end
end
```

명령문 창 결과

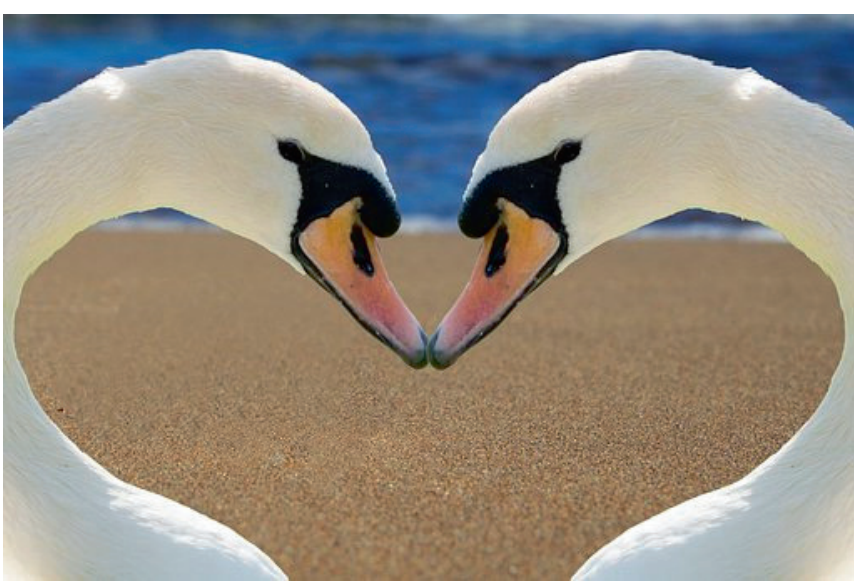
```
>> PerfectNumber
i =   6
i =   28
i =   496
i =   8128
```



부부수

부부수(夫婦數, Betrothed numbers)는 두 수의 쌍이 있어, 어느 한 수의 1을 제외한 진약수를 모두 더하면 다른 수가 되는 것을 말한다.

예를 들어 48의 진약수 중 1을 제외한 진약수들의 합이 2+3+4+6+8+12+16+24=75이고, 75의 진약수 중 1을 제외한 진약수들의 합이 3+5+15+25=48이 되는데, 이런 경우를 부부수라고 한다.



코드설명

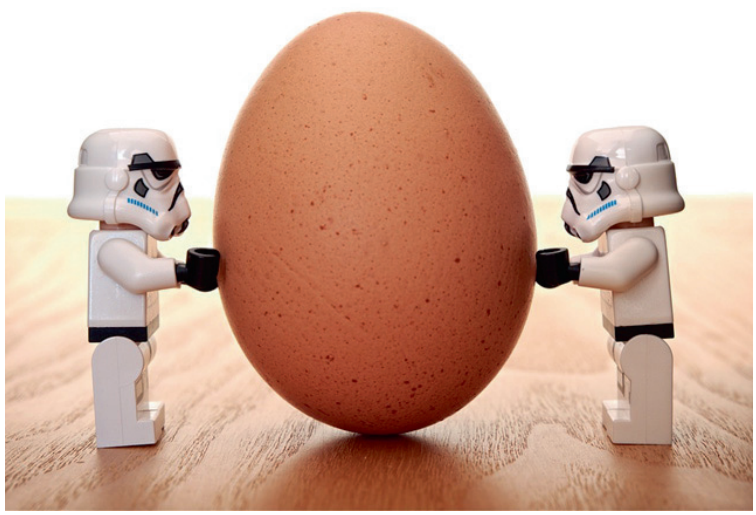
```
clear;
n=2000;
% 2000까지 수들 중에서 부부수를 찾기
for i=2:n
% 2부터 n까지의 수들 중에서 부부수를 체크한다.
  x=0;
% i의 진약수들을 전부 더하기 위해서 0으로 초기화
for j=2:i-1
% i의 1을 제외한 진약수를 구하기 위해서 2부터 i-1까지 검토
  if mod(i,j)==0
% j가 i의 약수인지를 검토한다.
    x=x+j;
% 약수를 누적해서 합해줌
  end
end
if x>0
% x가 0이면 i는 소수임으로 부부수가 될 수 없다. i가 소수가 아니라면 계속진행
  y=0;
```

명령문 창 결과

```
〉〉 BetrothedNumber
ans =    48    75
ans =   140    195
ans =  1050    1925
ans =  1575    1648
```

쌍둥이 소수

쌍둥이 소수(twin prime)는 두 수의 차가 2인 소수의 쌍, 즉 (p, p+2)이다.



코드명: TwinPrime.m

```
clear;
n=100;
y=ones(n,1);
y(1)=0;
for i=2:floor(sqrt(n))
   for j=i+1:n
```

```
      z(count)=i;
      count=count+1;
   end
end
for i=1:length(z)-1
% 쌍둥이 수를 찾기위해 루프를 사용함
   if z(i+1)-z(i)==2
% 두 소수의 차가 2이면 출력
      [z(i) z(i+1)]
   end
end
```

명령문 창 결과

```
>> TwinPrime
ans =    3    5
ans =    5    7
ans =   11   13
ans =   17   19
ans =   29   31
ans =   41   43
ans =   59   61
ans =   71   73
```

사촌소수

사촌 소수(cousin prime)은 두 수의 차가 4인 소수의 쌍, 즉 (p, p+4)이다. 100 이하의 사촌 소수를 구해보자.

코드명: CousinPrime.m

```
clear;
n=100;
y=ones(n,1);
y(1)=0;
for i=2:floor(sqrt(n))
  for j=i+1:n
   if mod(j,i)==0
   y(j)=0;
   end
  end
end
count=1;
for i=1:n
  if y(i)==1
```

```
 i의 배수들은 모두 소수가 아니므로 y를 0으로 세팅
    if mod(j,i)==0
    y(j)=0;
    end
  end
end
count=1;
% 1부터 n까지의 수 중에서 소수들만 저장하기 위해서
count=1로 세팅
for i=1:n
  if y(i)==1
% i번째 수가 소수이면 z에 저장하고 인텍스를 하나 크
게함
     z(count)=i;
     count=count+1;
  end
end
for i=1:length(z)-1
% 쌍둥이 수를 찾기위해 루프를 사용함
  if z(i+1)-z(i)==4
% 두 소수의 차가 4이면 출력
```

```
    [z(i) z(i+1)]
  end
end
```

명령문 창 결과

```
>> CousinPrime
ans =  7    11
ans = 13    17
ans = 19    23
ans = 37    41
ans = 43    47
ans = 67    71
ans = 79    83
```

골드바흐의 추측

골드바흐의 추측 "2보다 큰 모든 짝수는 두 소수의 합으로 나타낼 수 있다"를 확인하는 코드를 작성해 보자.

코드명: GoldbachConjecture.m

```
clear;
n=100;
x=ones(n,1);
x(1)=0;
for  i=2:floor(sqrt(n))
```

코드설명

```
clear;
n=100;
% 두 개의 소수의 합으로 나타낼 수
x=ones(n,1);
% 1부터 n까지 수 중에서 소수이면 1 아니면 0
x(1)=0;
% 1은 소수가 아니므로 0
for i=2:floor(sqrt(n))
% 소수 체크는 floor(sqrt(n))의 배수만 하면 됨
  for j=i+1:n
% i보다 1큰 수부터 n까지 체크
    if mod(j,i)==0
% 만약에 j가 i의 배수이면
    x(j)=0;
% 소수가 아님
    end
  end
end
count=1;
% 소수만 따로 저장하기 위한 인덱스
```

명령문 창 결과

```
>> GoldbachConjecture
ans =  3    97
ans = 11    89
ans = 17    83
ans = 29    71
ans = 41    59
ans = 47    53
```

[참고 문헌]

[1] 박경미의 수학 N, 박경미 지음, 동아시아, 2016.

[2] 박경미의 수학 콘서트 플러스, 박경미 지음, 동아시아, 2013.

[3] 수학비타민 플러스, 박경미 지음, 김영사, 2009.

[4] 중1수학(상) 파이썬 프로그래밍, 정기철 지음, 홍릉과학출판사, 2017.

[5] 위키백과, https://ko.wikipedia.org/wiki