# 신호 처리를 위한 행렬 계산 Programming Assignment 4

20180490 이재헌

**[Code Implementation]**

1. Toeplitz.m

$$T_n = \begin{pmatrix} 1 & r_1 & \cdots & r_{n-1} \\ r_1 & 1 & \cdots & r_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n-1} & r_{n-2} & \cdots & 1 \end{pmatrix}$$

```
function T = Toeplitz(c)

 n = length(c);
 T = zeros(n);

 T(1,:) = c.';
 for i=2:n
     for j=1:n
         T(i,j) = T(1,abs(i-j)+1);
     end
 end
```

첫 번째 행에 c를 넣어주고, 첫번째 행의 요소의 순서를 바꾸어 다른 행을 채웠다.

2. Durbin.m

**Function** $[y]$ = Solve_Yule_Walker_Eq($r$)

$y(1) = -r(1);\ \beta = 1;\ \alpha = -r(1)$

**for** $k = 1{:}n-1$

$\boxed{\beta = (1-\alpha^2)\beta}$

$\alpha = -(r(k+1) + r(k{:}-1{:}1)^T y(1{:}k))/\beta$

$z(1{:}k) = y(1{:}k) + \alpha\ y(k{:}-1{:}1)$

$y(1{:}k+1) = \begin{pmatrix} z(1{:}k) \\ \alpha \end{pmatrix}$

**end**

$\mathbf{2n^2}$ [flops]

```
function y = Durbin(r)

 n = length(r);
 y = zeros(n,1);
 y(1) = -r(1);
 a = -r(1);
 b = 1;
 z = zeros(n-1,1);

for k=1:n-1
    b = (1-a^2) * b;
    a = -(r(k+1) + r(k:-1:1).' * y(1:k))/b;
    z(1:k) = y(1:k) + a * y(k:-1:1);
    y(1:k+1) = [z(1:k); a];
end
```

강의 자료에 나온 알고리즘에 $\alpha$ 대신 a를, $\beta$ 대신 b를 사용하였다.

3. Levinson.m

**Function** $[x]$ = Solve_Toeplitz_System $(r, b)$

$y(1) = -r(1);\quad \beta = 1;\quad \alpha = -r(1);\quad x(1) = b(1);$

for $k = 1 : n-1$

$\beta = (1-\alpha^2)\beta;$

$\mu = (b(k+1) - r(1:k)^T x(k:-1:1)/\beta$

$v(1:k) = x(1:k) + \mu\, y(k:-1:1);\quad \boxed{x(1:k+1) = \begin{pmatrix} v(1:k) \\ \mu \end{pmatrix}}$

if $k < n-1$

$\alpha = \left(-r(k+1) + r(1:k)^T y(k:-1:1)\right)/\beta$

$z(1:k) = y(1:k) + \alpha\, y(k:-1:1);\quad \boxed{y(1:k+1) = \begin{pmatrix} z(1:k) \\ \alpha \end{pmatrix}}$

end

end

$4n^2$ [flops]

```
function x = Levinson(r,b)

 n = length(r);
 y = zeros(n,1);
 y(1) = -r(1);
 a = -r(1);
 beta = 1;
 x = zeros(n,1);
 x(1) = b(1);
 z = zeros(n-1,1);
 nu = zeros(n-1,1);
```

```
for k=1:n-1
    beta = (1-a^2) * beta;
    mu = (b(k+1) - r(1:k).' * x(k:-1:1))/beta;
    nu(1:k) = x(1:k) + mu * y(k:-1:1);
    x(1:k+1) = [nu(1:k); mu];

    if k < n-1
        a = -(r(k+1) + r(k:-1:1).' * y(1:k))/beta;
        z(1:k) = y(1:k) + a * y(k:-1:1);
        y(1:k+1) = [z(1:k); a];
    end
end
```

강의 자료에 나와있는 알고리즘을 코드로 구현하였다.

4. Trench.m

**Function** [$T\_inv$] = Get_Inverse_of_Toeplitz_Matrix ($T$)

$\gamma = 1/(1+r(1:n-1)^T y(1:n-1))$
$v(1:n-1) = \gamma\, y(n-1:-1:1)$
$B(1,1) = \gamma$
$B(1,2:n) = v(n-1:-1:1)^T$
for $\quad i = 2:\text{floor}((n-1)/2)+1$
  for $\quad j = i:n-i+1$
    $B(i,j) = B(i-1,j-1)+(v(n+1-j)v(n+1-i)-v(i-1)v(j-1))/\gamma$
  end
end

$13n^2/4$ [flops]

```
function B = Trench(r)

n = length(r);
y = Durbin(r(1:n-1));
B = zeros(n);

gamma = 1/(1 + r(1:n-1).' * y(1:n-1));
nu(1:n-1) = gamma * y(n-1:-1:1);
B(1,1) = gamma;
B(1,2:n) = nu(n-1:-1:1).';

for i=2:floor((n-1)/2)+1
    for j=i:n-i+1
        B(i,j) = B(i-1,j-1) + (nu(n+1-j)*nu(n+1-i) - nu(i-1)*nu(j-1))/gamma;
    end
end
```

여기까지가 강의자료에 나온 알고리즘이다. 하지만 이렇게 구현하면 다음과 같이 일부만 채워진 행렬이 나온다.

```
0.1748   -0.5353   -0.2779   -0.0369   -0.3863
     0    0.9600    0.2340   -0.7790         0
     0         0    1.3940         0         0
     0         0         0         0         0
     0         0         0         0         0
```

우리가 사용하는 Toeplitz matrix가 symmetric & persymmetric이기 때문에 이 행렬의 역행렬도 역시 symmetric & persymmetric 이다. 이 성질을 이용하여 아래의 코드를 추가해 행렬을 완성할 수 있다.

```
for i=1:floor((n-1)/2)
    for j=i:n-i+1
        B(j,i) = B(i,j);
        B(n+1-j,n+1-i) = B(i,j);
        B(n+1-i,n+1-j) = B(i,j);
    end
end
```

완성 후 행렬 모습은 아래와 같다.

```
 0.1748   -0.5353   -0.2779   -0.0369   -0.3863
-0.5353    0.9600    0.2340   -0.7790   -0.0369
-0.2779    0.2340    1.3940    0.2340   -0.2779
-0.0369   -0.7790    0.2340    0.9600   -0.5353
-0.3863   -0.0369   -0.2779   -0.5353    0.1748
```

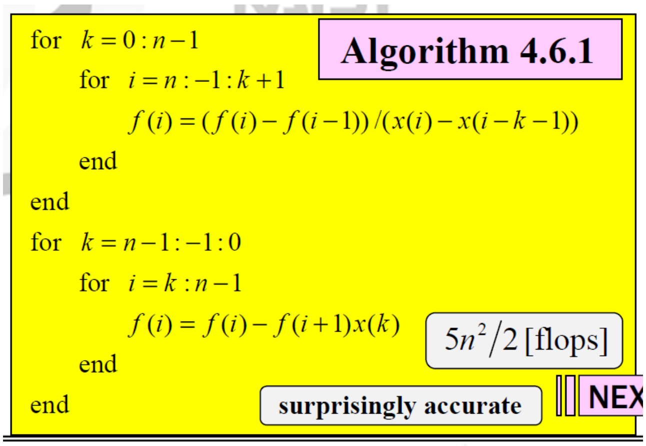
5. Vandermonde.m

$$V = V(x_0, x_1, \cdots, x_n) = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_0 & x_1 & \cdots & x_n \\ \vdots & \vdots & & \vdots \\ x_0^n & x_1^n & \cdots & x_n^n \end{pmatrix}$$

```
function V = Vandermonde(x)

 n = length(x);
 V = zeros(n);
 for i=1:n
     for j=1:n
         V(i,j) = x(j)^(i-1);
     end
 end
```

6. VTsolve.m

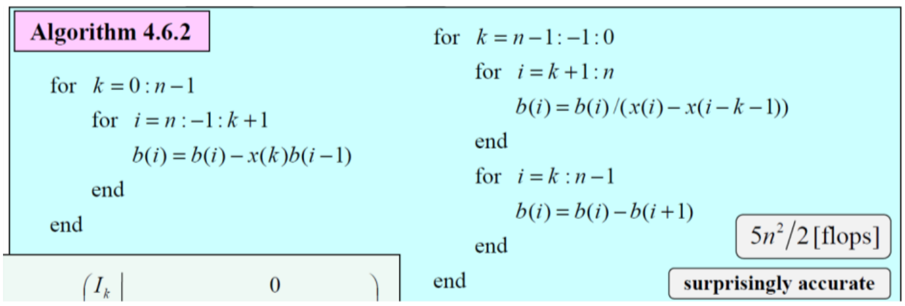



```
function a = VTsolve(x, f)

n = length(x);
a = f;
for k=1:n-1
    for i=n:-1:k+1
        a(i) = (a(i)-a(i-1))/(x(i)-x(i-k));
    end
end
for k=n-1:-1:1
    for i=k:n-1
        a(i) = a(i) - a(i+1) * x(k);
    end
end
```

교과서에 나와있는 알고리즘을 f 대신 a를 사용해서 구현하였다.

7. Vsolve.m




```
function z = Vsolve(x,b)

n = length(x);
z = b;
```

```
    for k=1:n-1
        for i=n:-1:k+1
            z(i) = z(i) - x(k) * z(i-1);
        end
    end
    for k=n-1:-1:1
        for i=k+1:n
            z(i) = z(i)/(x(i) - x(i-k));
        end
        for i=k:n-1
            z(i) = z(i) - z(i+1);
        end
    end
```

교과서에 나와있는 알고리즘을 b 대신 z를 사용해서 구현하였다.

**[Test.m Screenshot]**

```
Toeplitz matrix T =
    1.0000   -0.5702    0.2287   -1.2553   -1.3903
   -0.5702    1.0000   -0.5702    0.2287   -1.2553
    0.2287   -0.5702    1.0000   -0.5702    0.2287
   -1.2553    0.2287   -0.5702    1.0000   -0.5702
   -1.3903   -1.2553    0.2287   -0.5702    1.0000

||Durbin(r)  - T₩(-r)|| =  1.522e-15

||Levinson(r,b) - T₩b|| =  7.044e-16

||Trench(r) - inv(T)|| =  4.488e-16
```

```
Vandermonde matrix V =
    1.0000    1.0000    1.0000    1.0000    1.0000
    0.0030    0.4571    0.9217   -0.1973    0.0755
    0.0000    0.2090    0.8496    0.0389    0.0057
    0.0000    0.0955    0.7831   -0.0077    0.0004
    0.0000    0.0437    0.7218    0.0015    0.0000

||VTSolve(x,f) - V'₩f|| =  3.235e-13

||VSolve(x,b) - V₩b||    =  1.636e-13
```

Toeplitz 행렬의 알고리즘의 오차는 $10^{-16}$의 order로 나왔고, Vandermonde 행렬의 알고리즘의 오차는 $10^{-13}$의 order로 나왔다. 결과가 매우 정확하게 나왔음을 알 수 있다. 강의 자료에 나온 알고리즘을 사용하면 계산량을 줄이면서도 매우 정확한 결과를 얻을 수 있음을 확인하였다.