# MATH7502 Group project Topic3: Signal processing¶

## VMLS 7.4 Convolution

### Vectors Convolution¶ ¶

In [2]:

```
# convolution of an n-vector a and a m-vector b
function vecConv(a,b)
    n = length(a)
    m = length(b)
    c = zeros(n+m-1)
    for k in 1:n+m-1
        for i in 1:n
            for j in 1:m
                if i+j == k+1
                    c[k] = c[k]+a[i]*b[j]
                end
            end
        end
    end
    c
end
```

Out[2]:

```
vecConv (generic function with 1 method)
```

In [3]:

```
veca = (1,0,-1)
vecb = (2,1,-1)
vecConv(veca,vecb)
```

Out[3]:

```
5-element Array{Float64,1}:
  2.0
  1.0
 -3.0
 -1.0
  1.0
```

### Properties of convolution

In [4]:

```
# Symmetric
using Random
Random.seed!(1)
coma = rand(20)
comb = rand(15)
convab = vecConv(coma,comb)
convba = vecConv(comb,coma)
sum(convab - convba)
```

Out[4]:

```
1.6653345369377348e-15
```

In [5]:

```
# Associative
comc = rand(18)
convbc = vecConv(comb,comc)
convabnc = vecConv(convab,comc)
convanbc = vecConv(coma,convbc)
sum(convabnc - convanbc)
```

Out[5]:

```
-2.8033131371785203e-14
```

In [6]:

```
# For fixed a, the convolution a * b is a linear function of b
n = length(coma)
m = length(comb)
matrixb = zeros(n+m-1,n)
for p in 1:n
    matrixb[p:p+m-1,p] = comb
end
linearab = matrixb*coma
sum(convab - linearab)
```

Out[6]:

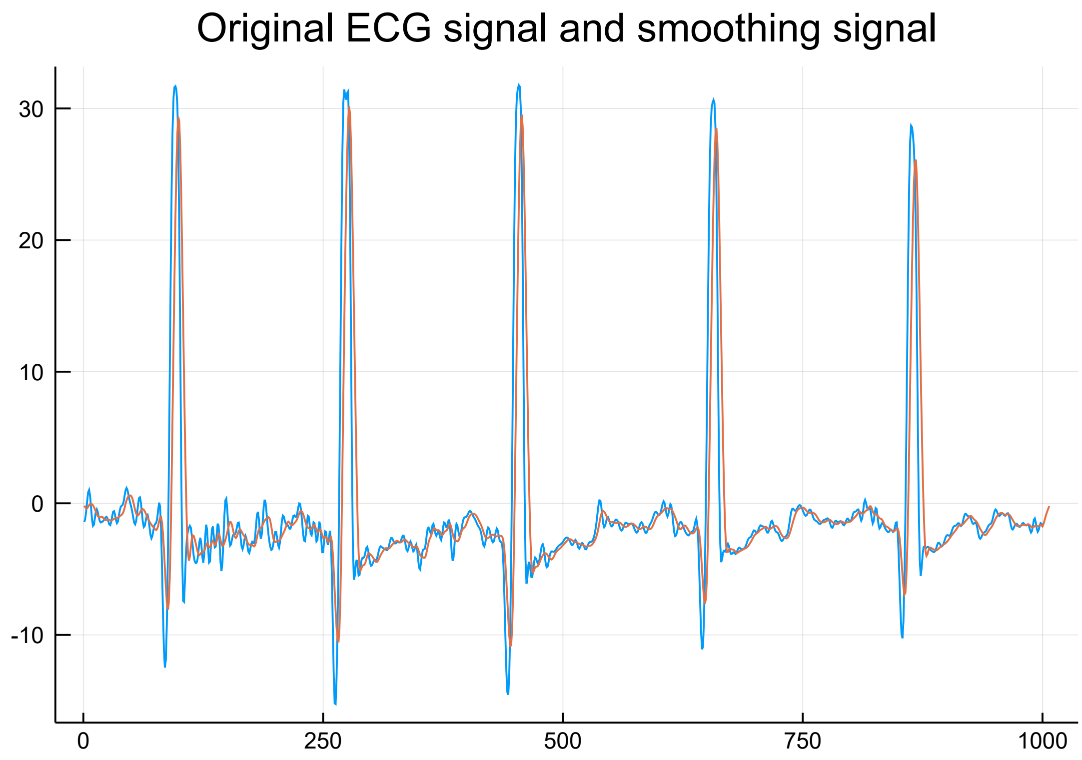
```
-7.494005416219807e-16
```

## Application: timeseries smoothing

In [8]:

```
using Plots
using DelimitedFiles
ECGsample = readdlm("ECG.txt")
ECGsample = ECGsample'
plot(ECGsample,title="Original ECG signal and smoothing signal",legend = false)
kernal = (1/8,1/8,1/8,1/8,1/8,1/8,1/8,1/8)
smooth = vecConv(ECGsample,kernal)
plot!(smooth)
```

Out[8]:




However, this is just an example. The average filter is not suitable to most of biosignals. Bandpass filter is commonly used on biosignals smoothing.

## 2-D convolution

In [9]:

```
# convolution of a m*n matrix A and a p*q matrix B
function matCov(A,B)
    mn = size(A)
    pq = size(B)
    m = mn[1]
    n = mn[2]
    p = pq[1]
    q = pq[2]
    C = zeros(m+p-1,n+q-1)
    for r in 1:m+p-1
        for s in 1:n+q-1
            for i in 1:m
                for j in 1:n
                    for k in 1:p
                        for l in 1:q
                            if i+k == r+1 && j+l == s+1
                                C[r,s] = C[r,s]+A[i,j]*B[k,l]
                            end
                        end
                    end
                end
            end
        end
    end
    C
end
```

Out[9]:

```
matCov (generic function with 1 method)
```

In [10]:

```
matA = [1 -1 3;4 -2 6;5 -1 7]
matB = [14 5 6 7;4 -3 66 54;12 5 -34 7]
matCov(matA,matB)
```

Out[10]:

```
5×6 Array{Float64,2}:
 14.0   -9.0   43.0   16.0    11.0   21.0
 60.0  -15.0  179.0   25.0   166.0  204.0
 98.0  -16.0  414.0  186.0   214.0  394.0
 68.0  -23.0  287.0  309.0   190.0  420.0
 60.0   13.0  -91.0  104.0  -245.0   49.0
```

## Application: image blurring

In [11]:

```
using Images
image = load("man.jpg")
```

Out[11]:

In [12]:

```
kernalB = ones(5,5)/25
blurimage = matCov(image,kernalB)
Gray.(blurimage)
```

Out[12]:

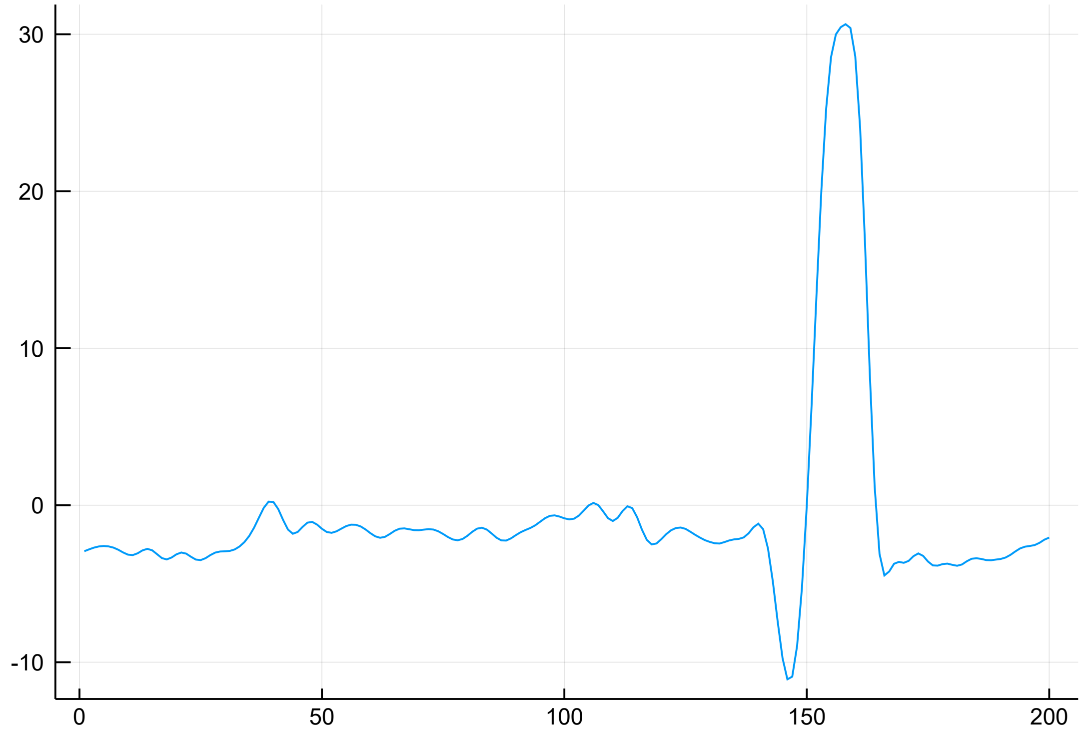
# LALFD 4.1 Fourier Transforms:Discrete and Continuous

## Load a discrete series f

In [13]:

```
# Intercept a signal of one period
f = ECGsample[500:699]
plot(f, legend = false)
```

Out[13]:




## Fourier Matrix F and DFT Matrix Ω

N×N Fourier matrix F contains powers of w = exp(2πi/N)

In [14]:

```
# Function to build a Fourier matrix
function Fmat(N)
    Fmat = ones(ComplexF64, N, N)
    w = exp(2im*pi/N)
    for i in 1:N
        for j in 1:N
            m = (i-1)*(j-1)
            Fmat[i, j] = w^m
        end
    end
    Fmat
end
```

Out[14]:

```
Fmat (generic function with 1 method)
```

N×N DFT matrix Ω contains powers of ω = exp(-2πi/N)

In [15]:

```
# Function to build a DFT matrix
function DFTmat(N)
    Dmat = ones(ComplexF64, N, N)
    w = exp(-2im*pi/N)
    for i in 1:N
        for j in 1:N
            m = (i-1)*(j-1)
            Dmat[i,j] = w^m
        end
    end
    Dmat
end
```

Out[15]:

```
DFTmat (generic function with 1 method)
```

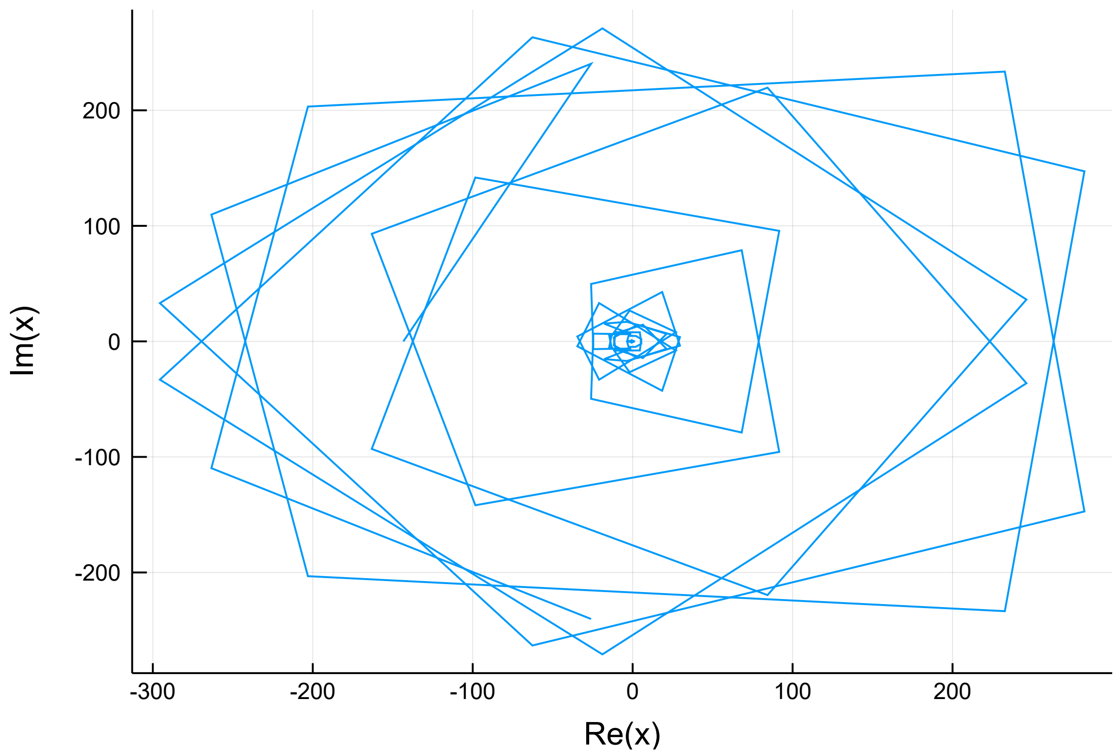
## DFT and inverse-DFT

In [16]:

```
# generate the Fourier matrix and DFT matrix
Ω = DFTmat(200)
F = Fmat(200)

# Ω times f produces discrete Fourier coefficients c
# That is also the discrete Fourier Transform
c = Ω * f
plot(c,legend = false)
```

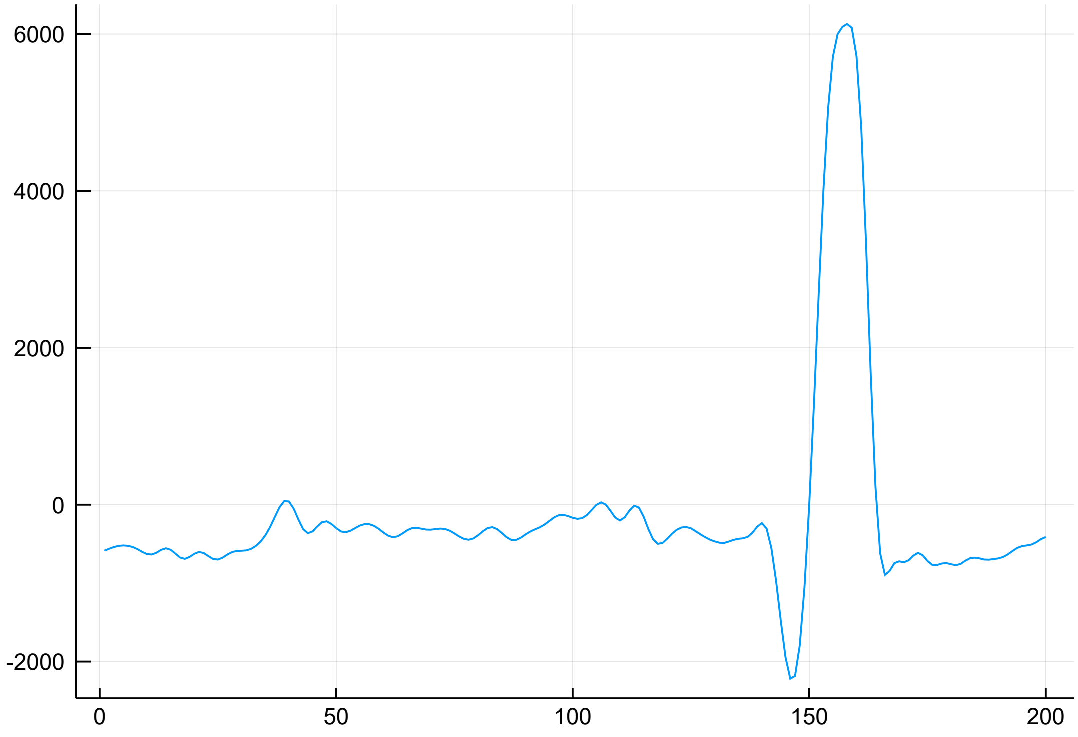
Out[16]:



In [17]:

```
# F times c bring back f
# That is also inverse Fourier transform
a = F*c
plot(real(round.(a,digits = 5)),legend = false)
```

Out[17]:



In [18]:

```
# FΩ = NI
round.(F * Ω,digits = 5)
```

Out[18]:

```
200×200 Array{Complex{Float64},2}:
 200.0+0.0im     0.0+0.0im    -0.0+0.0im  …     0.0-0.0im     0.0-0.0im
   0.0-0.0im   200.0+0.0im    -0.0+0.0im        0.0-0.0im     0.0-0.0im
  -0.0-0.0im    -0.0-0.0im   200.0+0.0im        0.0-0.0im     0.0-0.0im
  -0.0-0.0im    -0.0-0.0im    -0.0-0.0im       -0.0-0.0im    -0.0-0.0im
  -0.0-0.0im    -0.0-0.0im    -0.0-0.0im       -0.0-0.0im    -0.0-0.0im
  -0.0-0.0im    -0.0-0.0im    -0.0-0.0im  …    -0.0-0.0im    -0.0-0.0im
  -0.0-0.0im    -0.0-0.0im    -0.0-0.0im       -0.0-0.0im    -0.0-0.0im
  -0.0-0.0im    -0.0-0.0im    -0.0-0.0im       -0.0-0.0im    -0.0-0.0im
  -0.0-0.0im    -0.0-0.0im    -0.0-0.0im       -0.0-0.0im    -0.0-0.0im
  -0.0-0.0im    -0.0-0.0im    -0.0-0.0im       -0.0-0.0im    -0.0-0.0im
  -0.0-0.0im    -0.0-0.0im    -0.0-0.0im  …    -0.0-0.0im    -0.0-0.0im
  -0.0-0.0im    -0.0-0.0im    -0.0-0.0im       -0.0-0.0im    -0.0-0.0im
  -0.0-0.0im    -0.0-0.0im    -0.0-0.0im       -0.0-0.0im    -0.0-0.0im
      ⋮                                   ⋱
  -0.0+0.0im    -0.0+0.0im    -0.0+0.0im       -0.0+0.0im    -0.0+0.0im
  -0.0+0.0im    -0.0+0.0im    -0.0+0.0im       -0.0+0.0im    -0.0+0.0im
  -0.0+0.0im    -0.0+0.0im    -0.0+0.0im  …    -0.0+0.0im    -0.0+0.0im
  -0.0+0.0im    -0.0+0.0im    -0.0+0.0im       -0.0+0.0im    -0.0+0.0im
  -0.0+0.0im    -0.0+0.0im    -0.0+0.0im       -0.0+0.0im    -0.0+0.0im
  -0.0+0.0im    -0.0+0.0im    -0.0+0.0im       -0.0+0.0im    -0.0+0.0im
  -0.0+0.0im    -0.0+0.0im    -0.0+0.0im       -0.0+0.0im    -0.0+0.0im
  -0.0+0.0im    -0.0+0.0im    -0.0+0.0im  …    -0.0+0.0im    -0.0+0.0im
  -0.0+0.0im    -0.0+0.0im    -0.0+0.0im       -0.0+0.0im    -0.0+0.0im
   0.0+0.0im    -0.0+0.0im    -0.0+0.0im       -0.0+0.0im    -0.0+0.0im
   0.0+0.0im     0.0+0.0im     0.0+0.0im      200.0+0.0im    -0.0+0.0im
   0.0+0.0im     0.0+0.0im     0.0+0.0im       -0.0-0.0im   200.0+0.0im
```

## The DFT Matrix Ω is a Permutation of the Fourier Matrix F

In [19]:

```
# Function to build a Permutation matrix
function Pmat(N)
    P = zeros(N,N)
    P[1,1] = 1
    for i in 2:N
            P[N+2-i,i] = 1
    end
    P
end
```

Out[19]:

```
Pmat (generic function with 1 method)
```

In [20]:

```
P = Pmat(200)

# FP = Ω
round.(F*P-Ω,digits = 5)
```

Out[20]:

```
200×200 Array{Complex{Float64},2}:
 0.0+0.0im  0.0+0.0im  0.0+0.0im  …   0.0+0.0im   0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im  …  -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im  …  -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
    ⋮                             ⋱
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im  …  -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im  …  -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0-0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0+0.0im  -0.0-0.0im  -0.0-0.0im
```

In [21]:

```
# PΩ = F
round.(Ω*P-F,digits = 5)
```

Out[21]:

```
200×200 Array{Complex{Float64},2}:
 0.0+0.0im  0.0+0.0im  0.0+0.0im  …   0.0+0.0im   0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im  …  -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im  …  -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  -0.0+0.0im  -0.0+0.0im
    ⋮                             ⋱                                     
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im  …  -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im  …  -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0-0.0im
 0.0+0.0im  0.0+0.0im  0.0-0.0im     -0.0-0.0im  -0.0-0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0-0.0im  -0.0+0.0im  -0.0+0.0im
```

## Fast Fourier Transform

In [22]:

```
# Combine matrix - C
using LinearAlgebra
function Comat(N)
    n = floor(Int64,N/2)
    hI = Matrix{Float64}(I, n, n)

    DDmat = zeros(ComplexF64,n,n)
    w = exp(2im*pi/N)
    for i in 1:n
        DDmat[i,i] = w^(i-1)
    end
    Cm1 = hcat(hI,DDmat)
    Cm2 = hcat(hI,-DDmat)
    Cm = vcat(Cm1,Cm2)
end
```

Out[22]:

```
Comat (generic function with 1 method)
```

In [23]:

```
# Half-size transform matrix - H
function HalfF(N)
    n = floor(Int64,N/2)
    hF = Fmat(n)
    zm = zeros(n,n)
    HF1 = hcat(hF,zm)
    HF2 = hcat(zm,hF)
    HF = vcat(HF1,HF2)
end
```

Out[23]:

```
HalfF (generic function with 1 method)
```

In [24]:

```
# Even-Odd permutation matrix P - PP
function PEOmat(N)
    P = zeros(N,N)
    n = floor(Int64,N/2)
    for i in 1:n
            P[i,2*i-1] = 1
    end
    for i in n+1:N
        P[i,2*(i-n)] = 1
    end
    P
end
```

Out[24]:

```
PEOmat (generic function with 1 method)
```

In [25]:

```
# F = C*H*PP
C = Comat(1024)
H = HalfF(1024)
PP = PEOmat(1024)
FFT = C*H*PP
F1024 = Fmat(1024)
round.(FFT - F1024)
```

Out[25]:

```
1024×1024 Array{Complex{Float64},2}:
 0.0+0.0im  0.0+0.0im  0.0+0.0im  …   0.0+0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  0.0+0.0im   0.0-0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im      0.0-0.0im  0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im      0.0+0.0im  0.0+0.0im  -0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im      0.0-0.0im  0.0+0.0im   0.0-0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im  …   0.0+0.0im  0.0+0.0im   0.0-0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im      0.0-0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im      0.0+0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im      0.0+0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     -0.0+0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im  …  -0.0-0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im      0.0+0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im      0.0-0.0im  0.0+0.0im   0.0-0.0im
    ⋮                             ⋱
 0.0+0.0im  0.0-0.0im  0.0-0.0im      0.0+0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im      0.0+0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im      0.0+0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im  …   0.0+0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im      0.0+0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im      0.0+0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im      0.0+0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im      0.0+0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0-0.0im  …   0.0+0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0-0.0im      0.0+0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im      0.0+0.0im  0.0+0.0im   0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im      0.0+0.0im  0.0+0.0im   0.0+0.0im
```

# Full FFT by Recursion

In [26]:

```
# Funtion for full FFT
function fFFT(N)
    n = floor(Int64,N/2)
    C = Comat(N)
    PP = PEOmat(N)

    if N>4
        hf = fFFT(n)
    else
        hf = Fmat(2)
    end

    Z = zeros(n,n)
    Hf1 = hcat(hf,Z)
    Hf2 = hcat(Z,hf)
    H = vcat(Hf1,Hf2)

    FFT = C*H*PP
end
```

Out[26]:

```
fFFT (generic function with 1 method)
```

In [27]:

```
# test
fFFT(1024)
round.(fFFT(1024) - F1024)
```

Out[27]:

```
1024×1024 Array{Complex{Float64},2}:
 0.0+0.0im  0.0+0.0im  0.0+0.0im  …  0.0+0.0im  0.0+0.0im  0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     0.0+0.0im  0.0+0.0im  0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     0.0-0.0im  0.0+0.0im  0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     0.0-0.0im  0.0-0.0im  0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     0.0-0.0im  0.0-0.0im  0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im  …  0.0-0.0im  0.0-0.0im  0.0-0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     0.0-0.0im  0.0-0.0im  0.0-0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     0.0-0.0im  0.0-0.0im  0.0-0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     0.0-0.0im  0.0-0.0im  0.0-0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     0.0-0.0im  0.0-0.0im  0.0-0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im  …  0.0-0.0im  0.0-0.0im  0.0-0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     0.0-0.0im  0.0-0.0im  0.0-0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     0.0-0.0im  0.0-0.0im  0.0-0.0im
    ⋮                             ⋱
 0.0+0.0im  0.0-0.0im  0.0-0.0im     0.0+0.0im  0.0+0.0im  0.0+0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     0.0+0.0im  0.0+0.0im  0.0+0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     0.0+0.0im  0.0+0.0im  0.0+0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im  …  0.0+0.0im  0.0+0.0im  0.0+0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     0.0+0.0im  0.0+0.0im  0.0+0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     0.0+0.0im  0.0+0.0im  0.0+0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     0.0+0.0im  0.0+0.0im  0.0+0.0im
 0.0+0.0im  0.0-0.0im  0.0-0.0im     0.0+0.0im  0.0+0.0im  0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0-0.0im  …  0.0+0.0im  0.0+0.0im  0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0-0.0im     0.0+0.0im  0.0+0.0im  0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     0.0+0.0im  0.0+0.0im  0.0+0.0im
 0.0+0.0im  0.0+0.0im  0.0+0.0im     0.0+0.0im  0.0+0.0im  0.0+0.0im
```

# LALFD 4.2 Shift Matrices and Circulant Matrices

In [28]:

```
# Function to build a shift matrix
function Shiftm(N)
    mat = zeros(N,N)
    for i in 1:N
        if i == 1
            mat[N,i] = 1
        else
            mat[i-1,i] = 1
        end
    end
    mat
end
```

Out[28]:

```
Shiftm (generic function with 1 method)
```

In [29]:

```
Ps = Shiftm(6)
x = [1,2,3,4,5,6]
Ps^2*x
```

Out[29]:

```
6-element Array{Float64,1}:
 3.0
 4.0
 5.0
 6.0
 1.0
 2.0
```

In [30]:

```
# Function to build a Circulant matrix
function Cirmat(a)
    n = length(a)
    P = Shiftm(n)
    Id = Matrix{Float64}(I, n, n)
    mat = a[1]*Id
    for i in 2:n
        mat = mat+a[i]*P^(i-1)
    end
    mat
end
```

Out[30]:

```
Cirmat (generic function with 1 method)
```

In [31]:

```
vecc = [1,2,3,4,5]
C = Cirmat(vecc)
```

Out[31]:

```
5×5 Array{Float64,2}:
 1.0  2.0  3.0  4.0  5.0
 5.0  1.0  2.0  3.0  4.0
 4.0  5.0  1.0  2.0  3.0
 3.0  4.0  5.0  1.0  2.0
 2.0  3.0  4.0  5.0  1.0
```

In [32]:

```
# Product of two Circulant matrices is also circulant
vecd = [9,0,8,6,4]
D = Cirmat(vecd)
C*D
# C*D == D*C
```

Out[32]:

```
5×5 Array{Float64,2}:
 67.0  94.0  81.0  78.0  85.0
 85.0  67.0  94.0  81.0  78.0
 78.0  85.0  67.0  94.0  81.0
 81.0  78.0  85.0  67.0  94.0
 94.0  81.0  78.0  85.0  67.0
```

## Cyclic Convolution

In [33]:

```
function Cconv(a,b)
    A = Cirmat(a)
    B = Cirmat(b)
    m = A*B
    m[1,:]
end
```

Out[33]:

```
Cconv (generic function with 1 method)
```

In [34]:

```
Cconv(vecc,vecd)
```

Out[34]:

```
5-element Array{Float64,1}:
 67.0
 94.0
 81.0
 78.0
 85.0
```

## Eigenvalues and Eigenvectors of P

In [35]:

```
eigvals(Shiftm(4))
```

Out[35]:

```
4-element Array{Complex{Float64},1}:
   -1.0000000000000004 + 0.0im
 8.326672684688674e-17 + 0.9999999999999996im
 8.326672684688674e-17 - 0.9999999999999996im
    0.9999999999999999 + 0.0im
```

In [36]:

```
# The eigenvector matrix for P is the Fourier matrix
eP = eigvecs(Shiftm(4))

# Julia give one eigenvector situation. Eigenvector multiply a scalar is still eigenvector .
eigenP = hcat(eP[:,4]*(-2),eP[:,2]*2im,eP[:,1]*(-2),eP[:,3]*(-2im))
round.(eigenP,digits = 5)
```

Out[36]:

```
4×4 Array{Complex{Float64},2}:
 1.0-0.0im   1.0+0.0im   1.0-0.0im   1.0-0.0im
 1.0-0.0im  -0.0+1.0im  -1.0-0.0im  -0.0-1.0im
 1.0-0.0im  -1.0-0.0im   1.0-0.0im  -1.0+0.0im
 1.0-0.0im  -0.0-1.0im  -1.0-0.0im  -0.0+1.0im
```

In [37]:

```
round.(Fmat(4),digits = 5)
```

Out[37]:

```
4×4 Array{Complex{Float64},2}:
 1.0+0.0im   1.0+0.0im   1.0+0.0im   1.0+0.0im
 1.0+0.0im   0.0+1.0im  -1.0+0.0im  -0.0-1.0im
 1.0+0.0im  -1.0+0.0im   1.0-0.0im  -1.0+0.0im
 1.0+0.0im  -0.0-1.0im  -1.0+0.0im   0.0+1.0im
```

In [38]:

```
Shiftm(4)
```

Out[38]:

```
4×4 Array{Float64,2}:
 0.0  1.0  0.0  0.0
 0.0  0.0  1.0  0.0
 0.0  0.0  0.0  1.0
 1.0  0.0  0.0  0.0
```

## Eigenvalues and Eigenvectors of a Circulant C

In [39]:

```
# Eigenvectors of a circulant matrix
cc = [1,2,3,4]
Cir = Cirmat(cc)
round.(eigvecs(Cir),digits = 5)
```

Out[39]:

```
4×4 Array{Complex{Float64},2}:
 -0.5+0.0im  -0.5+0.0im  -0.5-0.0im  -0.5+0.0im
 -0.5+0.0im  -0.0+0.5im  -0.0-0.5im   0.5+0.0im
 -0.5+0.0im   0.5+0.0im   0.5-0.0im  -0.5+0.0im
 -0.5+0.0im   0.0-0.5im   0.0+0.5im   0.5+0.0im
```

In [40]:

```
# Eigenvector of the permutation
# Julia give one eigenvector situation. Eigenvector multiply a scalar is still eigenvector .
eigenP2 = hcat(eP[:,4],eP[:,3]*im,eP[:,2]*-im,eP[:,1])
round.(eigenP2,digits = 5)
# Eigenvectors of a circulant matrix are the same as the eigenvectors of the permutation
```

Out[40]:

```
4×4 Array{Complex{Float64},2}:
 -0.5+0.0im  -0.5+0.0im  -0.5-0.0im  -0.5+0.0im
 -0.5+0.0im   0.0+0.5im   0.0-0.5im   0.5+0.0im
 -0.5+0.0im   0.5-0.0im   0.5+0.0im  -0.5+0.0im
 -0.5+0.0im   0.0-0.5im   0.0+0.5im   0.5+0.0im
```

In [41]:

```
# Multiply F times the vector c in the top row of C to find the eigenvalues
round.(Fmat(4)*Cir[1,:],digits = 5)
```

Out[41]:

```
4-element Array{Complex{Float64},1}:
 10.0 + 0.0im
 -2.0 - 2.0im
 -2.0 + 0.0im
 -2.0 + 2.0im
```

In [42]:

```
round.(eigvals(Cir),digits = 5)
```

Out[42]:

```
4-element Array{Complex{Float64},1}:
 10.0 + 0.0im
 -2.0 + 2.0im
 -2.0 - 2.0im
 -2.0 + 0.0im
```

In [43]:

```
# The N eogenvalues of C are the components of Fc = inverse Fourier transform of c
round.(Cir[1,:]'*DFTmat(4),digits = 5)
```

Out[43]:

```
1×4 Array{Complex{Float64},2}:
 10.0-0.0im  -2.0+2.0im  -2.0-0.0im  -2.0-2.0im
```

## The Convolution Rule

In [44]:

```
# Two circulant matrices C and D. Their top rows are the vectors c and d
c = C[1,:]
d = D[1,:]
```

Out[44]:

```
5-element Array{Float64,1}:
 9.0
 0.0
 8.0
 6.0
 4.0
```

In [45]:

```
# Top row of CD is the cyclic convolution of vectors c and d
C*D
```

Out[45]:

```
5×5 Array{Float64,2}:
 67.0  94.0  81.0  78.0  85.0
 85.0  67.0  94.0  81.0  78.0
 78.0  85.0  67.0  94.0  81.0
 81.0  78.0  85.0  67.0  94.0
 94.0  81.0  78.0  85.0  67.0
```

In [46]:

```
Cconv(c,d)
```

Out[46]:

```
5-element Array{Float64,1}:
 67.0
 94.0
 81.0
 78.0
 85.0
```

$F(c * d) = (Fc).\times(Fd)$

LHS:Convolve c and d first, then transform by F

RHS:Transform by F first, then multiply Fc times Fd component by component

In [48]:

```
# Convolution Rule
Fmat(5)*Cconv(c,d)
```

Out[48]:

```
5-element Array{Complex{Float64},1}:
               405.0 + 0.0im
 -6.3196601125010545 + 10.3228644035338im
  -28.68033988749893 + 2.43689772174681im
  -28.68033988749893 - 2.4368977217467886im
 -6.3196601125010154 - 10.322864403533785im
```

In [49]:

```
(Fmat(5)*c).*(Fmat(5)*d)
```

Out[49]:

```
5-element Array{Complex{Float64},1}:
               405.0 + 0.0im
  -6.319660112501046 + 10.3228644035338im
  -28.68033988749893 + 2.4368977217467958im
 -28.680339887498935 - 2.4368977217467833im
  -6.319660112501056 - 10.322864403533783im
```