# CSP Lab | Assignment 1

**Aim :** To C Code fundamental operations for signal processing, which include Convolution, Correlation, Downsampling (M=2, 3) & Upsampling (L=2, 3). Also to create their header files for further uses.

**Understanding & Results :**

1. Convolution

$$y[n] = \sum x(k)\, h(n - k)$$

Convolution operation gives the output response of a system ( with response, h[n]) upon excitation of a system with signal, x[n]. It involves shifting and addition operations.
Implemented the convolution operations using dynamic memory allocation based on standard rules of convolution. Below are the results:

**Given Signals:**

x[n] = {0.5377, 1.8339, -2.2588, 0.8622, 0.3188, -1.3077, -0.4336, 0.3426, 3.5784, 2.7694, -1.3499, 3.0349, 0.7254, -0.0631, 0.7147, -0.2050, -0.1241, 1.4897, 1.4090, 1.4172}

h[n] = {0.6715, -1.2075, 0.7172, 1.6302, 0.4889, 1.0347, 0.7269, -0.3034, 0.2939, -0.7873, 0.8884, -1.1471, -1.0689, -0.8095, -2.9443}

**Library Convolution Output:**
{0.361066, 0.582191, -3.345580, 5.498300, 0.805462, -2.874046, 4.106151, -0.410274, -1.445901, -1.804175, -4.350492, 13.203415, -2.715714, 4.821051, 10.269640, -4.427006, 9.755986, 2.903170, -0.592847, 4.635051, -7.384201, 0.336801, -9.399403, -8.545025, 3.613228, -9.319174, -3.912467, 0.599374, -3.378534, -1.245523, -3.972275, -7.041554, -5.295742, -4.172662 }

2. Correlation

$$Rxy[k] = \sum x(n)\, y(n - k)$$

Correlation gives the measure of similarity between two signals x[n] & y[n]. This is computed by shifting a signal in time and element wise multiplication with other or same signal.
Implemented the correlation operation using dynamic memory allocation based on standard rules. It uses a basic convolution skeleton with one signal flipped along the y-axis. Below are the results:

**Given Signals:**

x[n] = {0.5377, 1.8339, -2.2588, 0.8622, 0.3188, -1.3077, -0.4336, 0.3426, 3.5784, 2.7694, -1.3499, 3.0349, 0.7254, -0.0631, 0.7147, -0.2050, -0.1241, 1.4897, 1.4090, 1.4172}

h[n] = {0.6715, -1.2075, 0.7172, 1.6302, 0.4889, 1.0347, 0.7269, -0.3034, 0.2939, -0.7873, 0.8884, -1.1471, -1.0689, -0.8095, -2.9443}

**Library Correlation Output:**

{ -1.583150, -5.834819, 4.591295, -3.287128, -0.848136, 6.467562, -2.287080, 3.294564, -10.074680, -9.504092, -1.490325, -14.680519, 0.247015, -5.437126, -12.322000, 7.005692, -4.859549, 0.220403, 0.664786, 0.141289, 5.301993, -4.103833, 2.326034, 8.033679, -5.260244, 3.125120, 2.816635, 1.961664, 5.222200, 4.070427, 1.438708, 0.315382, -0.765125, 0.951650 }

3. Downsampling

$$y[n] = x[Mn]$$

Downsampling is reducing the number of samples in the signal by a factor of M. For discrete time cases, we have to take care of non-integral time indexes (by discarding them simply) and keep only integral indexes which will constitute our final signal.

**Given Signal :**

x[n] = {0.3252, -0.7549, 1.3703, -1.7115, -0.1022, -0.2414, 0.3192, 0.3129, -0.8649, -0.0301, -0.1649, 0.6277, 1.0933, 1.1093, -0.8637, 0.0774, -1.2141, -1.1135, -0.0068, 1.5326, -0.7697, 0.3714, -0.2256, 1.1174, -1.0891 0.0326, 0.5525, 1.1006, 1.5442, 0.0859, -1.4916, -0.7423, -1.0616, 2.3505, -0.6156, 0.7481}

**Library Downsampled Output :**

**Downsampled (M = 2) Output:**
{ 0.325200, 1.370300, -0.102200, 0.319200, -0.864900, -0.164900, 1.093300, -0.863700, -1.214100, -0.006800, -0.769700, -0.225600, -1.089100, 0.552500, 1.544200, -1.491600, -1.061600, -0.615600 }

**Downsampled (M = 3) Output:**
{ 0.325200, -1.711500, 0.319200, -0.030100, 1.093300, 0.077400, -0.006800, 0.371400, -1.089100, 1.100600, -1.491600, 2.350500 }

4. Upsampling

$$y[n] = \begin{cases} x[n/L], & \text{if n is a multiple of L} \\ 0, & \text{otherwise} \end{cases}$$

Performed zero interpolation as part of the upsampling process. For discrete time cases, between two samples we pad (L-1) zeroes to upsample the signal by factor of L.

**Given Signal:**

x[n] = {0.3252, 1.3703, -0.1022, 0.3192, -0.8649, -0.1649, 1.0933, -0.8637, -1.2141, -0.0068, -0.7697, -0.2256, -1.0891, 0.5525, 1.5442, -1.4916, -1.0616, -0.6156}

**Library Upsampled Output:**

**Upsampled (L = 2) Output:**

{ 0.325200, 0.000000, 1.370300, 0.000000, -0.102200, 0.000000, 0.319200, 0.000000, -0.864900, 0.000000, -0.164900, 0.000000, 1.093300, 0.000000, -0.863700, 0.000000, -1.214100, 0.000000, -0.006800, 0.000000, -0.769700, 0.000000, -0.225600, 0.000000, -1.089100, 0.000000, 0.552500, 0.000000, 1.544200, 0.000000, -1.491600, 0.000000, -1.061600, 0.000000, -0.615600, 0.000000 }

**Upsampled (L = 3) Output:**

{ 0.325200, 0.000000, 0.000000, 1.370300, 0.000000, 0.000000, -0.102200, 0.000000, 0.000000, 0.319200, 0.000000, 0.000000, -0.864900, 0.000000, 0.000000, -0.164900, 0.000000, 0.000000, 1.093300, 0.000000, 0.000000, -0.863700, 0.000000, 0.000000, -1.214100, 0.000000, 0.000000, -0.006800, 0.000000, 0.000000, -0.769700, 0.000000, 0.000000, -0.225600, 0.000000, 0.000000, -1.089100, 0.000000, 0.000000, 0.552500, 0.000000, 0.000000, 1.544200, 0.000000, 0.000000, -1.491600, 0.000000, 0.000000, -1.061600, 0.000000, 0.000000, -0.615600, 0.000000, 0.000000 }

**Code Execution:**

Put all the three files (main.c, common_functions.c, common_functions.h) in the same directory and execute below commands.

// create header file's executable
**gcc -o common_functions.o -c common_functions.c**

// compile main C Code with header file attached
**gcc -o main main.c common_functions.o**

// run output
**./main**