Filter C Programming

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Based on

Introduction to Signal Processing

S. J. Ofranidis

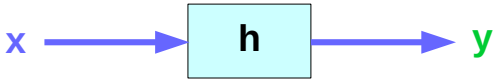
conv

```
#include <stdlib.h>      // to use max, min
/* conv.c - convolution of x[n] with h[n], resulting in y[n] */
/* h : filter array, M : filter order */
/* x : input array, L : input length */
/* y : output array  with length of L+M */

void conv(int M, double *h, int L, double *x, double *y)
{
      int n, m;

      for (n = 0; n < L+M-1; n++)
            for (y[n] = 0, m = max(0, n-L+1); m <= min(n, M-1); m++)
                  y[n] += h[m] * x[n-m];
}
```

Index Variable Contraints

h[0..M-1] : filter array, M : filter order
x[0..L-1] : input array, L : input length
y[0..L+M-2] : output array, M+L-1: output length

$$y[n] \mathrel{+}= \sum_{m=0}^{M+L-1} h[m] * x[n-m]^{L};$$



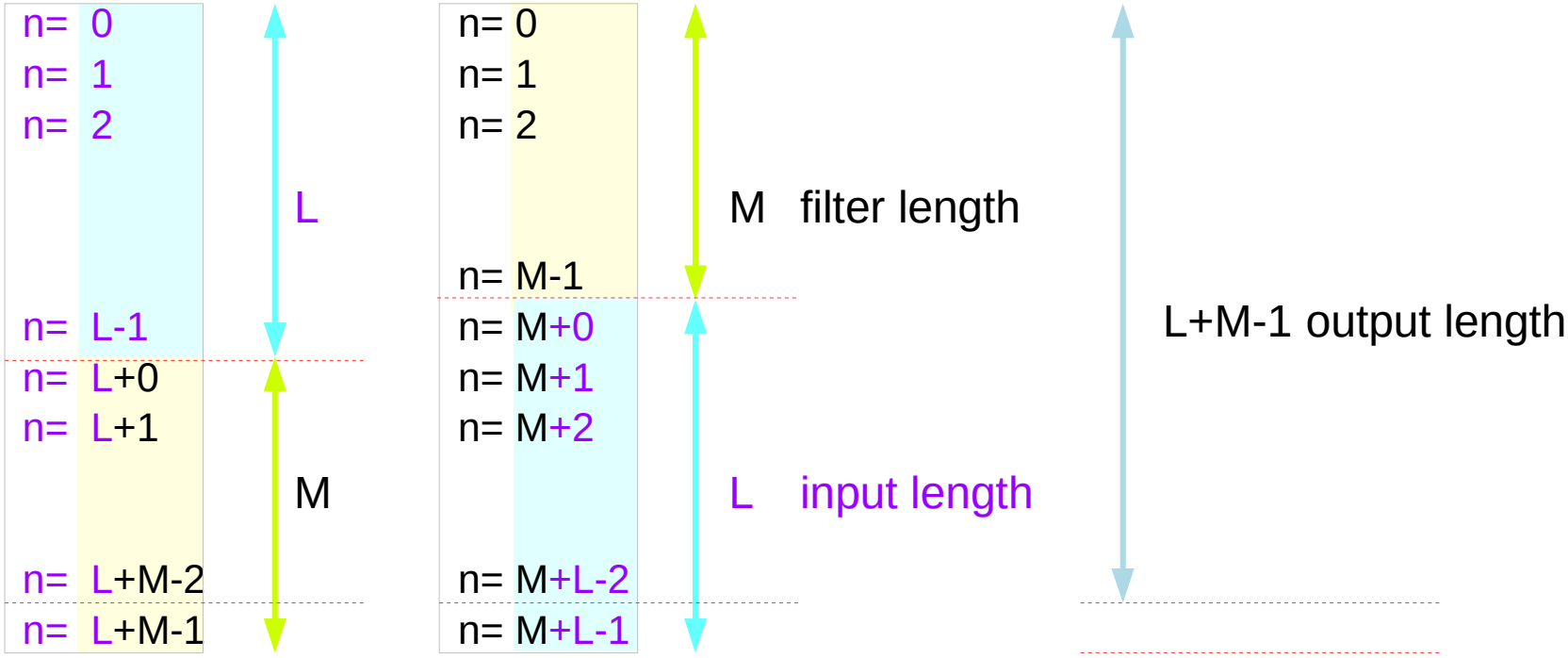
Constraint 1 : **n** ∈ **[0, L+M-2]**
y[ ] : array with size of **L+M-1**

Constraint 2 : **n-m** ∈ **[0, L-1]**
x[ ] : array with size of **L**

Constrain 3 : **m** ∈ **[0, M-1]**
h[ ] : array with size of **M**

Constraint 1 – counting **n**

Constraint 1 :**n** ∈ **[0, L+M-2]**



M < L is assumed

$$y[n] \mathrel{+}= \sum_{m=0}^{M-1} h[m] * x[n-m];$$

Constraint 1 & 2 – min **m** and max **n-m**

Constraint 2 : **n-m** ∈ **[0, L-1]** → n – m : **L-1** ... 0 (max, min) (0, n+1-L)

for **max(n-m)** values
m should be least possible

0 ≤ max(n-m) ≤ L-1

Case A) n ≤ L-1
m can be **zero**
for **max(n-m)** = **n**

Case B) n ≥ L
m can be **n-(L-1)**
for **max(n-m) = L-1**

min(m) = max(0, n-(L-1))

min **m** ← max **n-m** max(0, n+1-L)

M+L-1 M L

$$y[n] \mathrel{+}= h[m] * x[n-m];$$

Constraint 1 & 3 – max **m** and min **n-m**

Constrain 3 : **m** ∈ **[0, M-1]** → n – m : L-1 ... **0** (n, M-1)

max **min**

for **min(n-m)** values
m should be greatest possible

0 ≤ min(n-m) ≤ L-1

Case A) n ≤ M-1
m can be **n**
for **min(n-m)** = **0**

Case B) n ≥ M
m only can be **M-1**
for **min**(**n-m**) **= n-(M-1)**

max(m) = min(n, M-1)

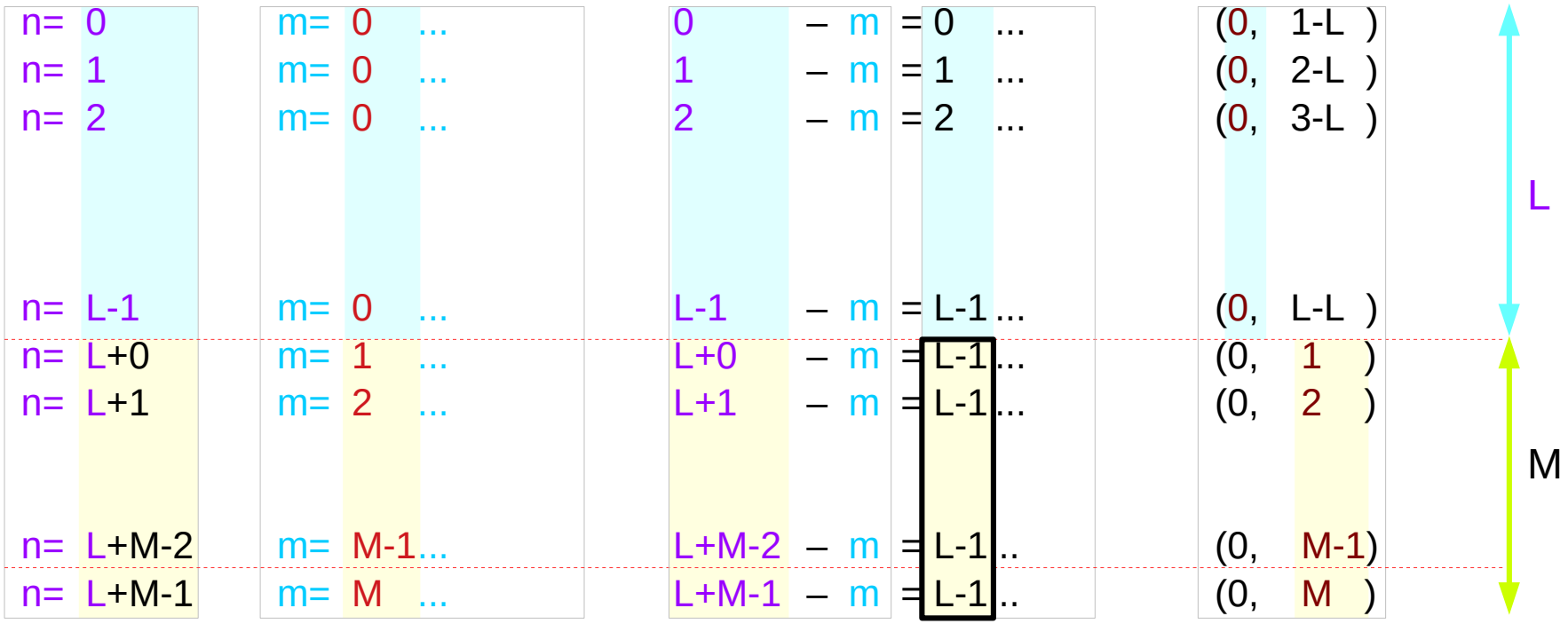
max **m** → min **n-m** min(n, M-1)

M+L-1 M L

$$y[n] \mathrel{+}= h[m] * x[n-m];$$

Constraint 1 & 2 – min **m** and max **n-m**

Constraint 2 : **n-m** ∈ **[0, L-1]** ➡ n – m : L-1 ... 0 (0, n+1-L)



min **m** ⬅ max **n-m** max(0, n+1-L)

m = max(0, n-L+1) .. min(n, M-1)

$$y[n] \mathrel{+}= \sum_{m=0}^{M+L-1} h[m] * x[n-m]; \qquad (M,\ L)$$

Constraint 1 & 3 – max **m** and min **n-m**

Constrain 3 : **m** ∈ **[0, M-1]** ➡ n – m : L-1 ... 0 (n, M-1)

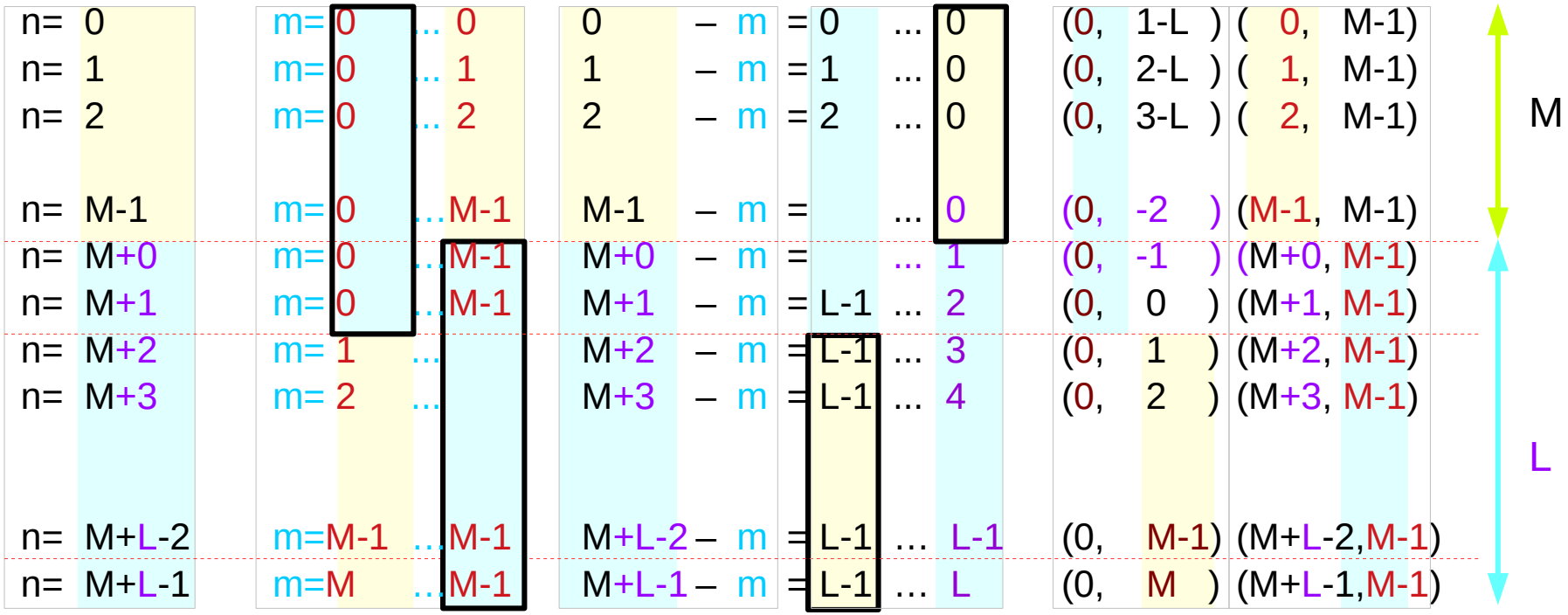
n=	0	m=	...	0	0	– m =	...	0	(0,	M-1)	
n=	1	m=	...	1	1	– m =	...	0	(1,	M-1)	
n=	2	m=	...	2	2	– m =	...	0	(2,	M-1)	M
n=	M-1	m=	...	M-1	M-1	– m =	...	0	(M-1,	M-1)	
n=	M+0	m=	...	M-1	M+0	– m =	...	1	(M+0,	M-1)	
n=	M+1	m=	...	M-1	M+1	– m =	...	2	(M+1,	M-1)	
					M+2	– m =	...	3			L
n=	M+L-2	m=	...	M-1	M+L-2	– m =	...	L-1	(M+L-2,	M-1)	
n=	M+L-1	m=	...	M-1	M+L-1	– m =	...	L	(M+L-1,	M-1)	

max **m** ➡ min **n-m** min(n, M-1)

m = max(0, n-L+1) .. min(n, M-1)

$$y[n] \mathrel{+}= \sum_{m}^{M+L-1} h[m] * x[n-m];$$

Constrain 3 : **m ∈ [0, M-1]** → n – m : L-1 ... 0 (n, M-1)

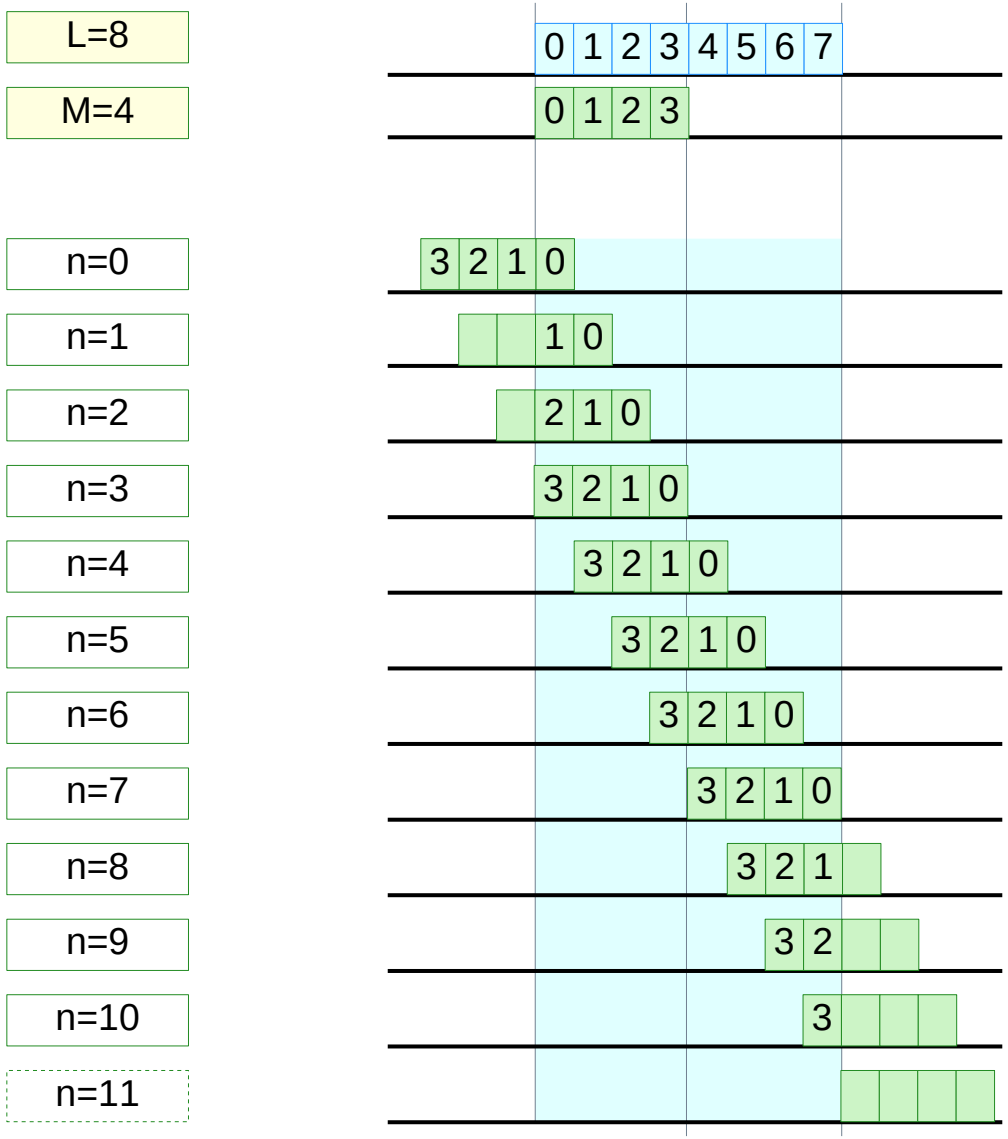


max **m** → min **n-m** min(n, M-1)

m = max(0, n-L+1) .. min(n, M-1)

$$y[n] \mathrel{+}= \sum_{m=0}^{M+L-1} h[m] * x[n-m]; \quad (m \le M,\ n-m \le L)$$

Conv.c



M+L-1 M L

$$y[n] \mathrel{+}= h[m] * x[n-m];$$

```
n=0 m=0
----------
n=1 m=0
n=1 m=1
----------
n=2 m=0
n=2 m=1
n=2 m=2
----------
n=3 m=0
n=3 m=1
n=3 m=2
n=3 m=3
----------
n=4 m=0
n=4 m=1
n=4 m=2
n=4 m=3
----------
n=5 m=0
n=5 m=1
n=5 m=2
n=5 m=3
----------
n=6 m=0
n=6 m=1
n=6 m=2
n=6 m=3
----------
n=7 m=0
n=7 m=1
n=7 m=2
n=7 m=3
----------
n=8 m=1
n=8 m=2
n=8 m=3
----------
n=9 m=2
n=9 m=3
----------
n=10 m=3
----------
```

Conv.c

```
#include <stdio.h>

#define MIN(a,b) (((a)<(b))?(a):(b))
#define MAX(a,b) (((a)>(b))?(a):(b))

#define L 8
#define M 4


int main(void)
{
  int n, m;


  for (n = 0; n < L+M-1; n++) {
    for (m = MAX(0, n-L+1); m <= MIN(n, M-1); m++)
      printf("n=%d m=%d \n", n, m);
    printf("----------------\n");
  }

}
```

```
n=0 m=0
----------------
n=1 m=0
n=1 m=1
----------------
n=2 m=0
n=2 m=1
n=2 m=2
----------------
n=3 m=0
n=3 m=1
n=3 m=2
n=3 m=3
----------------
n=4 m=0
n=4 m=1
n=4 m=2
n=4 m=3
----------------
n=5 m=0
n=5 m=1
n=5 m=2
n=5 m=3
----------------
n=6 m=0
n=6 m=1
n=6 m=2
n=6 m=3
----------------
n=7 m=0
n=7 m=1
n=7 m=2
n=7 m=3
----------------
n=8 m=1
n=8 m=2
n=8 m=3
----------------
n=9 m=2
n=9 m=3
----------------
n=10 m=3
----------------
```

Conv.c

```
#include <stdio.h>

#define MIN(a,b) (((a)<(b))?(a):(b))
#define MAX(a,b) (((a)>(b))?(a):(b))

#define L 8
#define M 4


int main(void)
{
  int n, m;


  for (n = 0; n < L+M; n++) {
    for (m = MAX(0, n-L+1); m <= MIN(n, M); m++)
      printf("n=%d m=%d \n", n, m);
    printf("----------------\n");
  }

}
```

```
n=0 m=0
----------------
n=1 m=0
n=1 m=1
----------------
n=2 m=0
n=2 m=1
n=2 m=2
----------------
n=3 m=0
n=3 m=1
n=3 m=2
n=3 m=3
----------------
n=4 m=0
n=4 m=1
n=4 m=2
n=4 m=3
n=4 m=4
----------------
n=5 m=0
n=5 m=1
n=5 m=2
n=5 m=3
n=5 m=4
----------------
n=6 m=0
n=6 m=1
n=6 m=2
n=6 m=3
n=6 m=4
----------------
n=7 m=0
n=7 m=1
n=7 m=2
n=7 m=3
n=7 m=4
----------------
n=8 m=1
n=8 m=2
n=8 m=3
n=8 m=4
----------------
n=9 m=2
n=9 m=3
n=9 m=4
----------------
n=10 m=3
n=10 m=4
----------------
n=11 m=4
----------------
```

Tap.c

```
/* tap.c - i-th tap of circular delay-line buffer */
/* usage: si = tap(D, w, p, i); */
/* p passed by value */
/* i=0,1,..., D */

double tap(int D, double *w, double p, int i)
{
        return w[(p - w + i) % (D + 1)];
}
```

delay.c

```
/* delay.c - delay by D time samples */
/* w[0] = input, w[D] = output */
/* reverse-order updating */

void delay(int D, double *w)
{
        int i;

        for (i=D; i>=1; i--)
                w[i] = w[i-1];

}
```

References

[1] S. J. Ofranidis , Introduction to Signal Processing