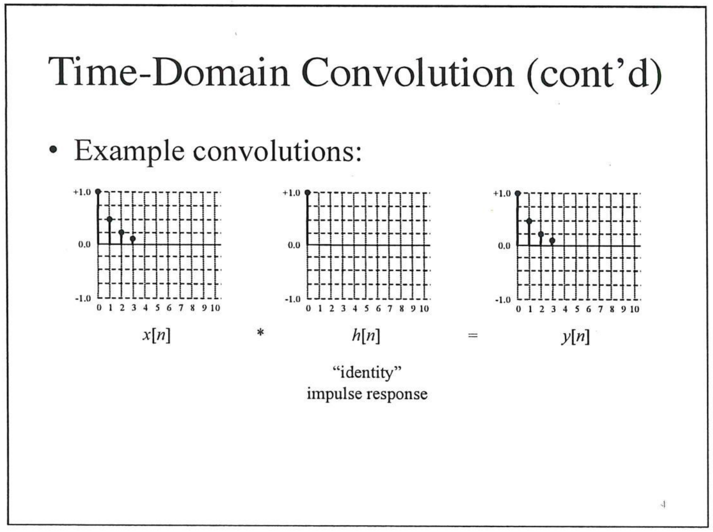
```c
/*  Include files  */
#include <stdio.h>


/*  Function prototypes  */
void convolve(float x[], int N, float h[], int M, float y[], int P);
void print_vector(char *title, float x[], int N);


/*****************************************************************************
*
*    Function:     main
*
*    Description:  Tests the convolve function with various input signals
*
*****************************************************************************/

int main(void)
{
  float input_signal[100], impulse_response[20], output_signal[120];
  int input_size, impulse_size, output_size;

  /*  Create an example input signal  */
  input_signal[0] = 1.0;
  input_signal[1] = 0.5;
  input_signal[2] = 0.25;
  input_signal[3] = 0.125;
  input_size = 4;

  /*  Print out the input signal to the screen  */
  print_vector("Original input signal", input_signal, input_size);

  /*  Create an "identity" impulse response.  The output should be
      the same as the input when convolved with this  */
  impulse_response[0] = 1.0;
  impulse_size = 1;

  /*  Set the expected size of the output signal  */
  output_size = input_size + impulse_size - 1;

  /*  Do the convolution, and print the output signal  */
  convolve(input_signal, input_size, impulse_response, impulse_size,
           output_signal, output_size);
  print_vector("Output signal using identity IR", output_signal, output_size);


  /*  Create an "inverse" impulse response.  The output should be
      inverted when convolved with this  */
  impulse_response[0] = -1.0;
  impulse_size = 1;

  /*  Set the expected size of the output signal  */
  output_size = input_size + impulse_size - 1;

  /*  Do the convolution, and print the output signal  */
  convolve(input_signal, input_size, impulse_response, impulse_size,
           output_signal, output_size);
  print_vector("Output signal using inverse IR", output_signal, output_size);


  /*  Create a "scaling" impulse response.  The output should be
      1/2 the amplitude when convolved with this  */
  impulse_response[0] = 0.5;
  impulse_size = 1;

  /*  Set the expected size of the output signal  */
  output_size = input_size + impulse_size - 1;

  /*  Do the convolution, and print the output signal  */
  convolve(input_signal, input_size, impulse_response, impulse_size,
           output_signal, output_size);
  print_vector("Output signal scaled by 1/2", output_signal, output_size);


  /*  Create a "delay" impulse response.  The output should be
      delayed by 2 samples  */
  impulse_response[0] = 0.0;
  impulse_response[1] = 0.0;
  impulse_response[2] = 1.0;
  impulse_size = 3;

  /*  Set the expected size of the output signal  */
  output_size = input_size + impulse_size - 1;

  /*  Do the convolution, and print the output signal  */
  convolve(input_signal, input_size, impulse_response, impulse_size,
           output_signal, output_size);
  print_vector("Output delayed 2 samples", output_signal, output_size);


  /*  Create a "delay and scaling" impulse response.  The output should be
      delayed by 2 samples and be 1/2 the amplitude  */
  impulse_response[0] = 0.0;
  impulse_response[1] = 0.0;
  impulse_response[2] = 0.5;
  impulse_size = 3;

  /*  Set the expected size of the output signal  */
  output_size = input_size + impulse_size - 1;

  /*  Do the convolution, and print the output signal  */
  convolve(input_signal, input_size, impulse_response, impulse_size,
           output_signal, output_size);
  print_vector("Delayed 2 samples, 1/2 amplitude", output_signal, output_size);


  /*  Create an "echo effect".  The output will contain the original signal
      plus a copy delayed by 2 samples and 1/2 the amplitude.  The original
      and copy will overlap starting at the 3rd sample  */
  impulse_response[0] = 1.0;
  impulse_response[1] = 0.0;
  impulse_response[2] = 0.5;
  impulse_size = 3;

  /*  Set the expected size of the output signal  */
  output_size = input_size + impulse_size - 1;

  /*  Do the convolution, and print the output signal  */
  convolve(input_signal, input_size, impulse_response, impulse_size,
           output_signal, output_size);
  print_vector("Overlapping echo", output_signal, output_size);


  /*  Create an "echo effect" that doesn't overlap.  The output will
```

```
      contain the original signal  plus a copy delayed by 5 samples
      and 1/2 the amplitude.  */
  impulse_response[0] = 1.0;
  impulse_response[1] = 0.0;
  impulse_response[2] = 0.0;
  impulse_response[3] = 0.0;
  impulse_response[4] = 0.0;
  impulse_response[5] = 0.5;
  impulse_size = 6;

  /*  Set the expected size of the output signal  */
  output_size = input_size + impulse_size - 1;

  /*  Do the convolution, and print the output signal  */
  convolve(input_signal, input_size, impulse_response, impulse_size,
           output_signal, output_size);
  print_vector("Non-overlapping echo", output_signal, output_size);


  /*  Interchange the input signal and impulse response.  Since
      convolution is commutative, you should get the same output  */
  convolve(impulse_response, impulse_size, input_signal, input_size,
           output_signal, output_size);
  print_vector("Same as above, but with interchanged h[] and x[]",
               output_signal, output_size);


  /*  End of program  */
  return 0;
}



/*****************************************************************************
*
*    Function:     convolve
*
*    Description:  Convolves two signals, producing an output signal.
*                  The convolution is done in the time domain using the
*                  "Input Side Algorithm" (see Smith, p. 112-115).
*
*    Parameters:   x[] is the signal to be convolved
*                  N is the number of samples in the vector x[]
*                  h[] is the impulse response, which is convolved with x[]
*                  M is the number of samples in the vector h[]
*                  y[] is the output signal, the result of the convolution
*                  P is the number of samples in the vector y[].  P must
*                       equal N + M - 1
*
*****************************************************************************/

void convolve(float x[], int N, float h[], int M, float y[], int P)
{
  int n, m;

  /*  Make sure the output buffer is the right size: P = N + M - 1  */
  if (P != (N + M - 1)) {
    printf("Output signal vector is the wrong size\n");
    printf("It is %-d, but should be %-d\n", P, (N + M - 1));
    printf("Aborting convolution\n");
    return;
  }
```

```

  /*  Clear the output buffer y[] to all zero values  */
  for (n = 0; n < P; n++)
    y[n] = 0.0;

  /*  Do the convolution  */
  /*  Outer loop:  process each input value x[n] in turn  */
  for (n = 0; n < N; n++) {
    /*  Inner loop:  process x[n] with each sample of h[]  */
    for (m = 0; m < M; m++)
      y[n+m] += x[n] * h[m];
  }
}



/*****************************************************************************
*
*    Function:     print_vector
*
*    Description:  Prints the vector out to the screen
*
*    Parameters:   title is a string naming the vector
*                  x[] is the vector to be printed out
*                  N is the number of samples in the vector x[]
*
*****************************************************************************/

void print_vector(char *title, float x[], int N)
{
  int i;

  printf("\n%s\n", title);
  printf("Vector size:  %-d\n", N);
  printf("Sample Number \tSample Value\n");
  for (i = 0; i < N; i++)
    printf("%-d\t\t%f\n", i, x[i]);
}
```



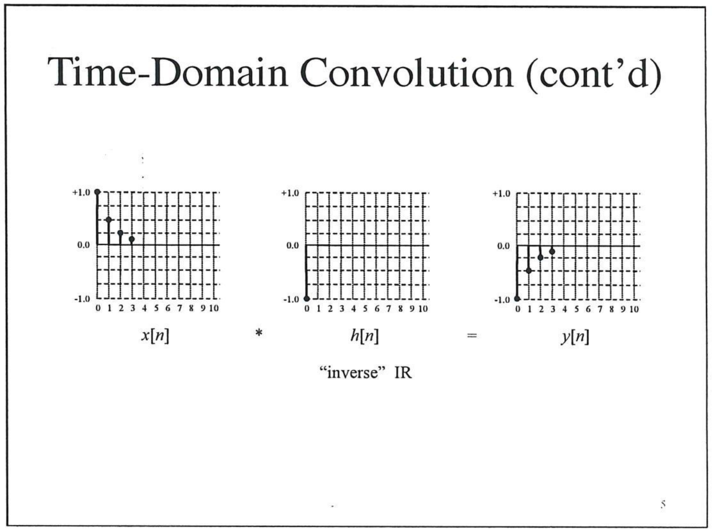
Time-Domain Convolution (cont’d)

- Example convolutions:

$x[n]$ * $h[n]$ = $y[n]$

“identity”
impulse response

4

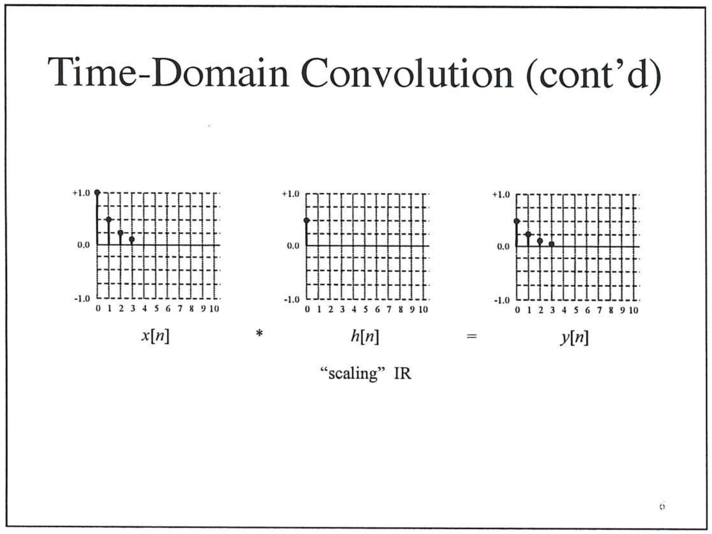


Time-Domain Convolution (cont’d)

$x[n]$ * $h[n]$ = $y[n]$

“inverse” IR

5

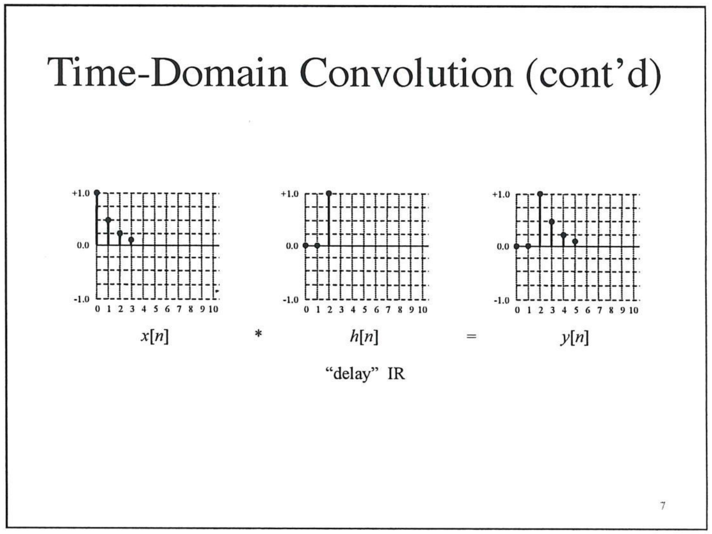


Time-Domain Convolution (cont’d)

$x[n]$ * $h[n]$ = $y[n]$

“scaling” IR

6

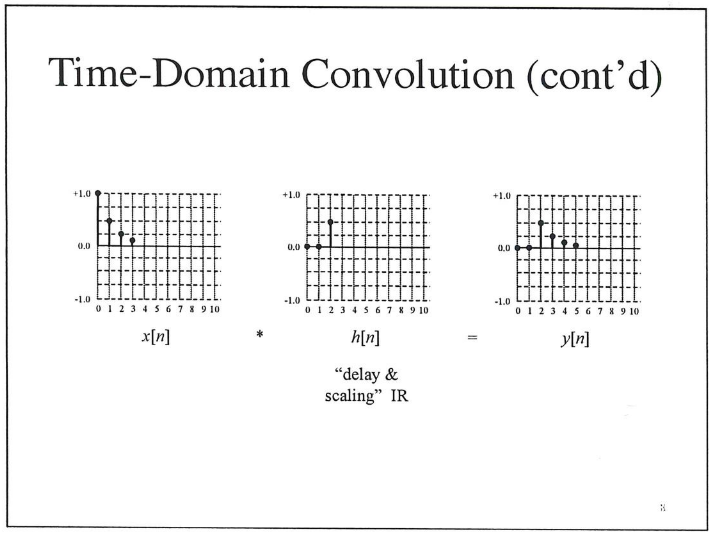


Time-Domain Convolution (cont’d)

$x[n]$ * $h[n]$ = $y[n]$

“delay” IR

7

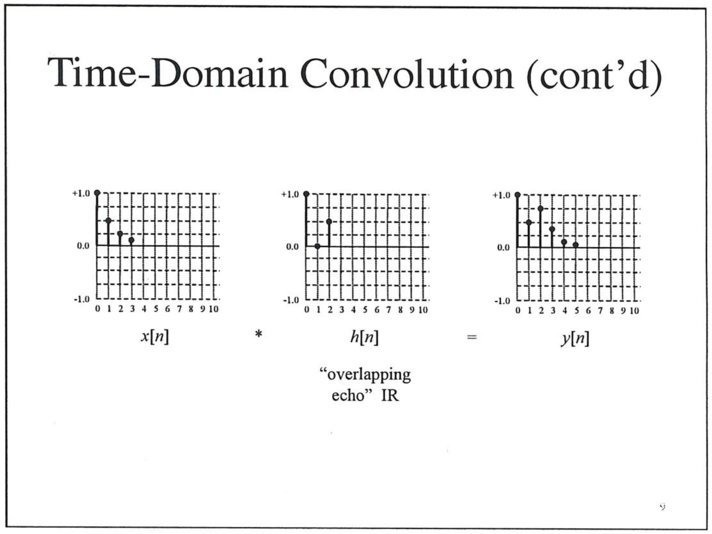


Time-Domain Convolution (cont'd)

$x[n]$ * $h[n]$ = $y[n]$

"delay & scaling" IR

8

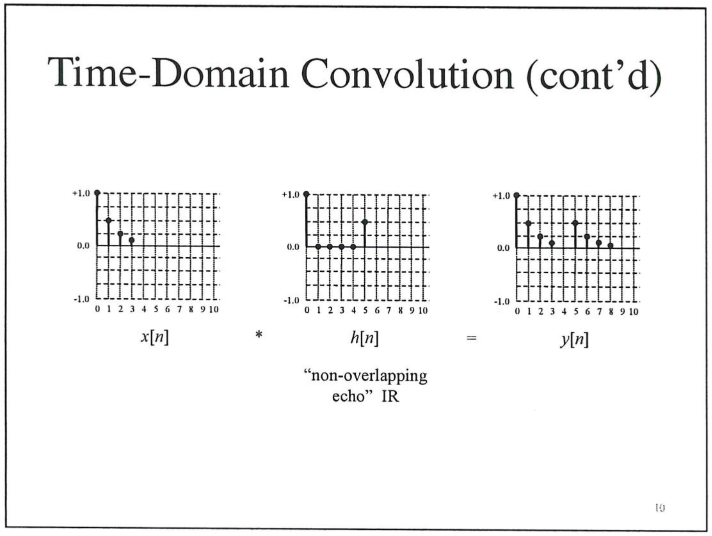


Time-Domain Convolution (cont'd)

$x[n]$ * $h[n]$ = $y[n]$

"overlapping echo" IR

9

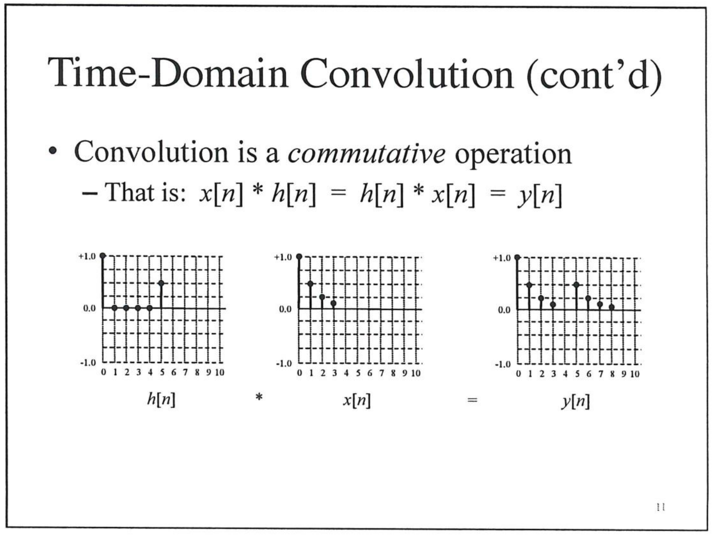


Time-Domain Convolution (cont'd)

$x[n]$ * $h[n]$ = $y[n]$

"non-overlapping echo" IR

10



Time-Domain Convolution (cont'd)

- Convolution is a *commutative* operation
  - That is: $x[n] * h[n] = h[n] * x[n] = y[n]$

$h[n]$ * $x[n]$ = $y[n]$

11