LIBCONV

An SGI Optimized Library for Convolution and Correlation

Jean-Pierre Panziera

LIBCONV is an optimized library available on Silicon Graphics computers. It features Convolution for Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters, together with Correlations. The library modules take 1D and 2D inputs, and are available in single and double precision for real and complex arithmetic. The 2D subprograms have been parallelized and take fully advantage of SGI parallel architecture.

Section 1 presents the definitions used for Convolution and Correlation, 1D and 2D. LIBCONV implementation and conventions are explained in section 2. Examples of performance achieved by LIBCONV modules are presented in section 3.

In Appendix three short programs provide examples of practical usage of LIBCONV modules.

1 Definitions: Convolutions and Correlations

This section summarizes the mathematical aspects and conventions regarding Convolution and Correlation. Only the topics relevant to the current implementation of SGI's LIBCONV will be discussed here. It is assumed that the user is familiar with Signal Processing techniques. The reference books listed at the end of this note are a good starting point for an unexperienced reader.

1.1 1D Convolution and Correlation.

Consider an input signal F and a Linear Time-Invariant Filter G, the output of the linear system will be a signal H. Mathematically speaking, H is the result of the **Convolution** of Input signal F by Filter G. The sign "*" is used here to define convolution: H = F*G.

In case F, G and H are continuous functions of time (analog signals), these functions are related as follow:

$$\mathbf{H}(\mathbf{t}) = \mathbf{F(t)*G(t)} = \int_{-\infty}^{\infty} \mathbf{F}(\mathbf{t}-\mathbf{u}) \times \mathbf{G}(\mathbf{u})\,\mathbf{du}$$

One important property of the Convolution operator is commutativity, F*G=G*F.

When F, G and H are digitized signals they are represented by their discrete samples. F, for example is defined by the infinite sequence: ..., F(-n), ..., F(-2), F(-1), F(0), F(1), F(2), ..., F(n), ... and the Convolution definition becomes:

$$\mathbf{H}(n) = \sum_{-\infty}^{\infty} \mathbf{F}(n-k) \times \mathbf{G}(k) = \sum_{-\infty}^{\infty} \mathbf{G}(n-k) \times \mathbf{F}(k)$$

Among all signals the Impulse sequence I(k) needs a particular mention. All the samples I(k) are null, except for I(0) = 1. One can check that F*I = I*F = F.

<u>Correlation</u> is a mathematical operator measuring similitudes between two signals F and G. Here we'll use "(*)" to define Correlation: H = F (*) G.

In case F, G and H are continuous function of time (analog signals) F(t), G(t) and H(t) are related as follow:

$$\mathbf{H}(\mathbf{t}) = \mathbf{F(t)(*)G(t)} = \int_{-\infty}^{\infty} \mathbf{F}(\mathbf{t}+\mathbf{u}) \times \mathbf{G}(\mathbf{u})\,\mathbf{du}$$

If you consider the flipped function G', defined by G'(t) = G(-t), then:

$$\mathbf{H}(\mathbf{t}) = \mathbf{F(t)(*)G(t)} = \int_{-\infty}^{\infty} \mathbf{F}(\mathbf{t}-\mathbf{u}) \times \mathbf{G}'(\mathbf{u})\,\mathbf{du} = \mathbf{F(t)*G'(t)}$$

So the correlation F (*) G = F * G' (convolution with flipped G).

Although they are closely related, Correlation is more difficult to manipulate than Convolution, among other restrictions, Correlation is NOT commutative.

When F, G and H are digitized signals the Correlation operator is defined as follow:

$$\mathbf{H}(n) = \sum_{-\infty}^{\infty} \mathbf{F}(n+k) \times \mathbf{G}(k)$$

So far Convolution and Correlation have been defined for infinite length signals. But computer representations are always finite, and in practice signals are always truncated (finite length signals).

<u>1.2 1D Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters</u>

<u>FIR filters:</u> A filter G[n1:n2] with sequence samples Zero except between times n1 and n2 is called a Finite Impulse Response (FIR) filter. Since FIR filters can be represented by a limited sequence, they are really convenient for Digital Signal Processing.

Note that the Convolution of two FIR signals F[m1:m2] and G[n1:n2] is another FIR signal H[(m1+n1):(m2+n2)].

IIR Filters: Given a FIR filter G[0:m], we define filter G~ such that G * G~ = I. This filter G~ has an Infinite Impulse Response (IIR), for example if G is the two sample sequence (1, 1/2), then G~ will be the infinite sequence: 1, -1/2, 1/4, -1/8, 1/16, ..., $(-1/2)^n$, ...

One can consider G~ as the "inverse" filter of G, and for simplicity we use the symbol "/":

H = F * G~ = F / G

The general formula for an IIR filter is:

$$\mathbf{H}(n) = \sum_{k=0}^{\infty} \mathbf{F}(n-k) \times \mathbf{G\sim}(k) = \left(\mathbf{F}(n) - \sum_{k=1}^{m} \mathbf{H}(n-k) \times \mathbf{G}(k)\right) / (\mathbf{G}(0))$$

Note that IIR filters are recursive since H(n) depends on H(n-k).

The IIR filter defined by G is stable only when G is a Minimum Phase Filter, like in the previous example G(1,1/2). In case the IIR filter is stable, the amplitude of the output will be negligeable after a certain time. So although they have an infinite length response, it is still possible to manipulate IIR filters using signals of limited length.

IIR filters are of great interest for Digital Signal Processing, since they can be defined by a short sequence. The combination of both a FIR (G1) and an IIR (G2) filter enables the optimal design for a filter, G = G1 / G2.

1.3 2 Dimension Convolution and Correlation

2D Convolution and Correlation definitions are the extensions from the 1D case.

The **2D Convolution** of continuous function is defined as:

$$\mathbf{H}(x,y) = \mathbf{F(x,y)*G(x,y)} = \int\int \mathbf{F(x-u,y-v).G(u,v)dudv}$$

For a **2D FIR filter**, the discretized formula is:

$$\mathbf{H}(\mathbf{m},\mathbf{n}) = \sum_{\mathbf{k}}\sum_{\mathbf{l}} \mathbf{F}(\mathbf{m}-\mathbf{k},\mathbf{n}-\mathbf{l}) \times \mathbf{G}(\mathbf{k},\mathbf{l})$$

The computations for a **2D IIR filter** are:

$$\mathbf{H}(\mathbf{m},\mathbf{n}) = \left(\mathbf{F}(\mathbf{m},\mathbf{n}) - \sum_{\mathbf{k}}\sum_{\mathbf{l}} \mathbf{H}(\mathbf{m}-\mathbf{k},\mathbf{n}-\mathbf{l}) \times \mathbf{G}(\mathbf{k},\mathbf{l})\right) / (\mathbf{G}(\mathbf{0},\mathbf{0}))$$

The definition for **2D Correlation:**

$$\mathbf{H}(x,y) = \mathbf{F(x,y)(*)G(x,y)} = \int\int \mathbf{F(x+u,y+v).G(u,v)dudv}$$

is discretized as:

$$\mathbf{H}(\mathbf{m},\mathbf{n}) = \sum_{\mathbf{k}}\sum_{\mathbf{l}} \mathbf{F}(\mathbf{m}+\mathbf{k},\mathbf{n}+\mathbf{l}) \times \mathbf{G}(\mathbf{k},\mathbf{l})$$

1.4 Convolution & Fast Fourier Transform (FFT) - Flops count

The relationships between operators in the Fourier (frequency) domain and operators in space (time) domain are well known. Consider two sequences F and G, and their Fourier transform Φ and Γ. The convolution F*G has a Fourier transform equal to the product ΦxΓ. The convolution is equivalent to a product in the Fourier domain.

Therefore it is possible to apply a filter G to a signal F, by first computing their FFTs, then multiplying the FFTs and finally taking the inverse transform of the product. To compare the direct convolution and the method using the FFTs, one needs to estimate the number of floating point operations involved.

The number of floating point operations (Flops) necessary for the convolution of two 1D signals of respective lengths M and N is 2*M*N.

Typically for a signal of length N, the FFT computation requires 5*N*log(N) flops, where "log()" is the logarithm base 2. For the transform of a 1000 samples sequence, 5*1,000*10=50,000 flops are needed. Given the efficiency of the Fast Fourier Transform (FFT) algorithms, O(N*log(n)), one may consider using systematically the FFT approach. If we assume Γ the FFT of filter G is already available, the whole convolution cycle requires:

- 5*N*log(N) flops for the direct transform F -> Φ,
- 8*N/2 for the products of the FFTs Φ and Γ,
- 5*N*log(N) flops for the inverse transform of the result.

This represents a total of N*(10xlog(N)+4) flops for the FFT approach versus 2*M*N flops for the plain convolution. For a sequence of 1000 samples, the FFT approach is advantageous only when the size M of the filter is larger than 50. In practice Digital Signal Processing filters are often shorter than 50 samples so the direct convolution is usually more attractive.

There are a few problems one should approach with care when using FFTs. The Fourier theory assumes the input sequence to be cyclic (to repeat itself indefinitely). This implicit "cyclic" assumption may produce unpleasant wrapping effects. Padding of the input signal fixes these side effects but add to the FFT cost since the new sequence is longer, sometimes twice as long. Finally FFT performance may vary a lot depending on the number N of samples in the transform. It is optimal when N is a power of two, extremely slow if N is prime. In contrast there is no such restriction for the direct convolution, which is extremely flexible.

In conclusion there is no method systematically superior to the other. The direct convolution approach is better when the filter is much shorter than the input sequence, while the FFT is more efficient when both inputs have similar length.

2 LIBCONV implementation

The two main concerns when writing the LIBCONV library have been Performance and Generality. The intent was to provide well tuned modules usable in most Convolution and Correlation instances. Although these two objectives may be contradictory, LIBCONV manages to provide the maximum flexibility with a high performance level.

To achieve maximum flexibility, 1D sequence are defined by 3 parameters (section 2.2), while 6 parameters are necessary for 2D inputs (section 2.4). One drawback of this versatility may come from the (very) long calling sequences, up to 23 parameters for **sfir2d**. To help the user, systematic conventions were used for modules names, calling parameters and sequence definition.

2.1 LIBCONV naming conventions

A systematic naming convention is used for all LIBCONV modules. Names are 6 or 7 letters *X.YYY.MN*.D where:

- *X* represents the precision and can be **"s"** for real single precision, **"d"** for double precision, **"c"** for complex and **"z"** for double complex.
- *YYY* is either **"fir"** for FIR convolution, "**iir"** for IIR filtering, and **"cor"** for Correlation.
- *MN* is the dimension, *MN* can be a 1 or 2 character string: "**1**" or "**2**" for 1D and 2D modules and "**M1**" for "multiple 1D" subprograms.

The following table 1 shows all the modules available in LIBCONV:

Table 1: LIBCONV naming convention

	1 Dimension			2 Dimension		
	FIR filter	IIR filter	Correlation	FIR filter	IIR filter	Correlation
Real Single Precision	**sfir1d**	**siir1d**	**scor1d**	**sfir2d** **sfirm1d**	**siir2d** **siirm1d**	**scor2d** **scorm1d**
Real Single Precision	**dfir1d**	**diir1d**	**dcor1d**	**dfir2d** **dfirm1d**	**diir2d** **diirm1d**	**dcor2d** **dcorm1d**
Single Complex	**cfir1d**	**ciir1d**	**ccor1d**	**cfir2d** **cfirm1d**	**ciir2d** **ciirm1d**	**ccor2d** **ccorm1d**
Double Complex	**zfir1d**	**ziir1d**	**zcor1d**	**zfir2d** **cfirm1d**	**ziir2d** **ziirm1d**	**zcor2d** **zcorm1d**

For example:

- **sfir1d** applies a FIR filter to single precision real 1D signals,
- **diir1d** performs a IIR filtering on double precision 1D data,
- **cfirm1d** applies a 1D FIR filter for each column of a 2D single complex array, and
- **zcor2d** computes the correlation of 2D double complex sequences.

For a complete description of each module's parameters, please refer to the man pages.

2.2 LIBCONV 1D sequence

To insure that LIBCONV's definition of a signal F (1D sequence) is general enough, three parameters are required e.g. F(Incf,f0,Nf):

- IncF represents the **increment** between two samples in the array holding F,
- f0 is the **time of the first sample** of the array containing the sequence F,
- Nf is the **number of samples** defined for F.

Therefore a signal F[0:N] sampled between time 0 and time N with all its samples stored consecutively will be defined as F(0,N+1,1).

2.3 1D FIR, IIR Filters and Correlation

FIR filters: in LIBCONV, FIR filters modules such as **sfir1d** compute:

$$\mathbf{H(IncH,h0,Nh)} \leftarrow \beta \cdot \mathbf{H(IncH,h0,Nh)} + \alpha \cdot \mathbf{F(Incf,f0,Nf)*G(IncG,g0,Ng)}$$

For example, the complete convolution of signal F[0:m] and the FIR filter G[0:n] resulting in output H[0:(m+n)], can be performed in LIBCONV as:

$$\mathbf{H(1,0,(m+n+1))} \leftarrow \mathbf{0} \cdot \mathbf{H(1,0,(m+n+1))} + \mathbf{1} \cdot \mathbf{F(1,0,m+1)*G(1,0,n+1)}$$

by setting parameters α to 1; β to 0; f0, g0 and h0 to 0; Nf, Ng, Nh to m+1, n+1 and m+n+1 respectively.

Note that the parameters h0 and Nh allows to compute only the interesting part of the Convolution which can be limited, rather than the whole output.

IIR filters: in LIBCONV, IIR filters modules such as **diir1d** compute:

$$\mathbf{H(IncH,h0,Nh)} \leftarrow \mathbf{F(Incf,f0,Nf)} / \mathbf{G(IncG,g0,Ng)}$$

The first program example (see Appendix 1) illustrates the use of 1D FIR and IIR filters.

Correlation: in LIBCONV, Correlation modules such as **ccor1d** compute:

$$\mathbf{H(IncH,h0,Nh)} \leftarrow \mathbf{F(Incf,f0,Nf)\,(*)\,G(IncG,g0,Ng)}$$

In the current implementation, the Correlation modules call the FIR Convolution since (see Section 1.1) Correlation is equal to the Convolution by the flipped sequence G:

H(IncH,h0,Nh) = F(IncF,f0,Nf) (*) G(IncG,g0,Ng) = F(IncF,f0,Nf) * G(-IncG,-g0,Ng)

The relationship between Convolution and Correlation is checked in the second code example (Appendix 2).

2.4 LIBCONV 2D Sequence

As for 1 D, LIBCONV 2D modules provide a great flexibility. 2D arrays are now defined with 6 parameters. The sequence F is fully specified by F(IncF,ldF,fx0,Nfx,fy0,Nfy) where:

- IncF the increment between two successive elements in the X direction,
- LdF the increment between two successive elements in the Y direction,
- fx0 represents the value of first index in the X direction,
- Nfx the number of samples in the X direction,
- fy0 represents the value of first index in the Y direction,
- Nfy the number of samples in the Y direction.

A given sample F(x,y) of the 2D sequence will be at an offset of (x*IncF+y*Ldf) elements from sample F(0,0).

Often users choose IncF=1 and LdF=Nfx, but it is sometimes useful to set LdF larger than Nfx. For example if Nfx is a power of 2 (e.g. 512), it is better to use a parameter LdF larger if one is to access the array in the Y direction. Choosing Ldf=514 or 516 will ensure a better cache usage, hence better performance when working in the Y direction.

In general, in order to use the transpose of a 2D sequence, one would need to copy all the sequence samples into another array. This is not necessary with the LIBCONV modules since the transposed of sequence F(IncF,ldF,fx0,Nfx,fy0,Nfy) can be specified by F(ldF,IncF,fy0,Nfy,fx0,Nfx), just exchanging X and Y parameters.

2.5 2D and "multiple 1D" modules

LIBCONV has modules to perform 2D FIR Convolution (such as **zfir2d**), 2D IIR Convolution (such as **siir2d**) and 2D Correlation (such as **dcor2d**).

Together with these "plain 2D" modules that operate with two 2D arrays F and G, "multiple 1D" modules (such as **cfirm1d**) are also available. The "multiple 1D" FIR convolution module take a 2D array F and a 1D array G1, and performs 1D FIR convolution in the X direction for each Y. By exchanging the X and Y parameters in the F definition, it is possible to work in the Y direction for each X. Appendix 3 gives examples of the two approaches for a 2D derivative computation.

Multiple 1D filters may save lots of computation when the 2D filter to be applied can be decomposed into the combination of a filter in the X direction and a filter in the Y direction.

Parallel Implementation: 2D FIR Convolution and Correlation computations are independent for each X and each Y sample. H(m,n) does not depend on other samples of H. Similarly "multiple 1D" modules treat each Y independently. So a natural parallel implementation will process concurrently the Y directions, this only requires to parallelize the outer loop on Y and is quite efficient.

The only problem arises with 2D IIR filters which are recursive filters, H(m,n) depends on H(m-k,n-l) (see section 1.3). After further analysis one can notice that H(m,n) depends only on the H(u,v) where (u<m AND v<n). For example H(m,n) and H(m-1,n+1) are independent, and they could be computed concurrently. Although it is not immediate, a parallel implementation is possible for 2D IIR filters. It requires an external loop over a variable K, and at each iteration, the elements H(m,n) such that "(m+n) = k" (elements along a diagonal) are computed in parallel.

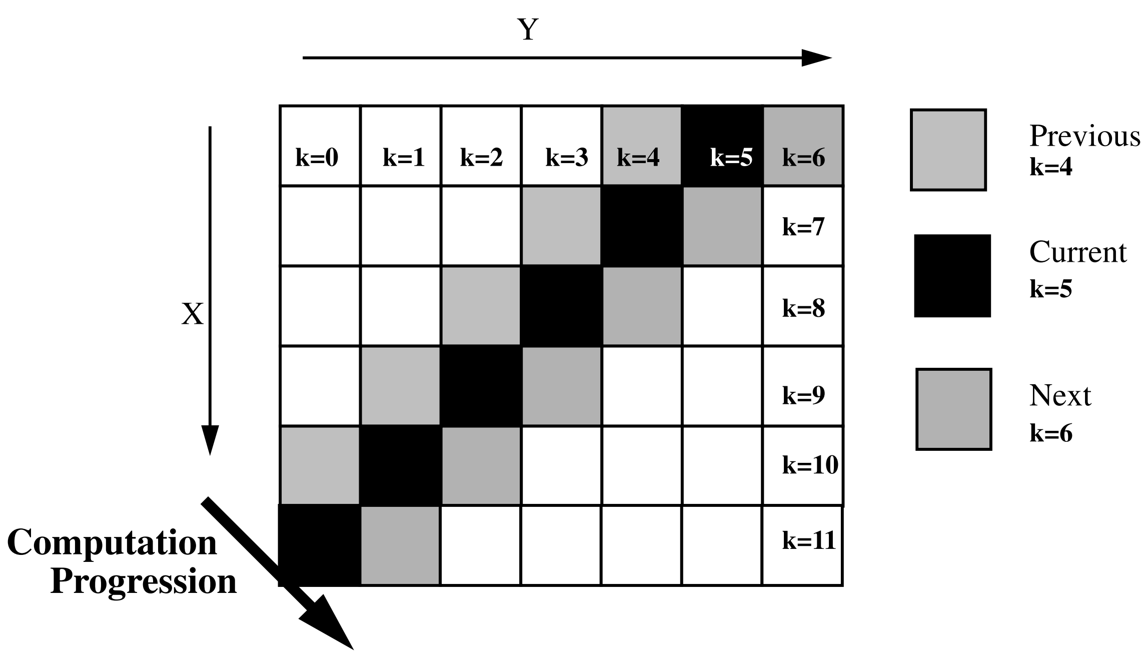

Figure 1 : The darkened samples computations are independent and are computed in parallel

3 LIBCONV performances

LIBCONV performance come from a optimized utilization of SGI's RISC Parallel architecture. To better use the RISC CPUs "nested loop optimization" techniques sometimes referred as "outer loop unrolling" are used. The Parallel strategy was explained in section 2.5.

Figure 2 summarizes performance for the 1D real single precision modules **sfir1d** for a 4D/Crimson workstation (1 CPU R4000 @ 50 MHz). A steady 35 Mflops can be achieved once the input sequence is larger than 150 samples.

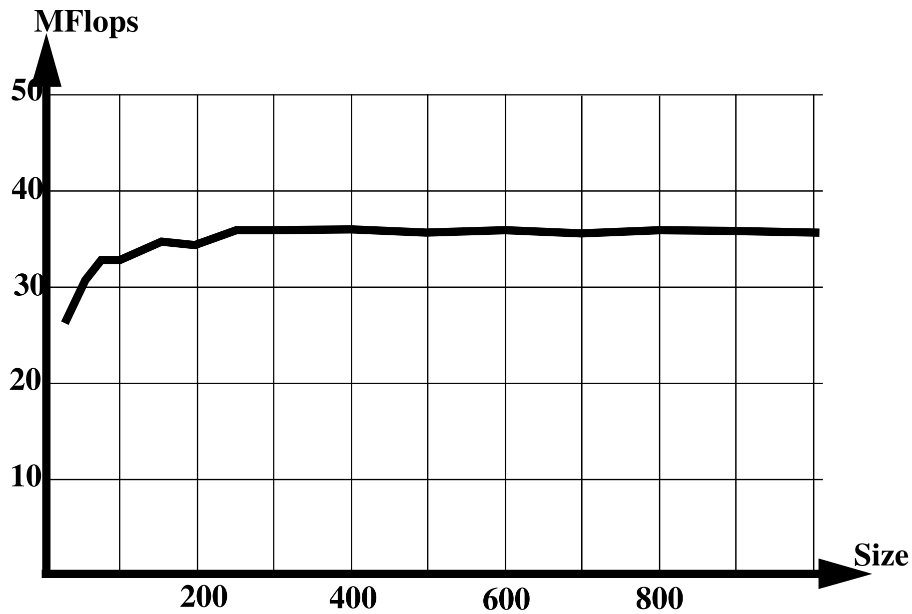

Figure 2 :sfir1d performance on a 4D/Crimson for varying sequence length

Figure 3 shows Parallel Speedups for the 2D real single precision modules **sfir2d** and **siir2d** for a 4D/380 server (8 CPUs R3000 @ 33 MHz). **sfir2d** speedup is very close to the linear optimum while **siir2d** has a larger Parallel overhead which impact on the Parallel speedup.

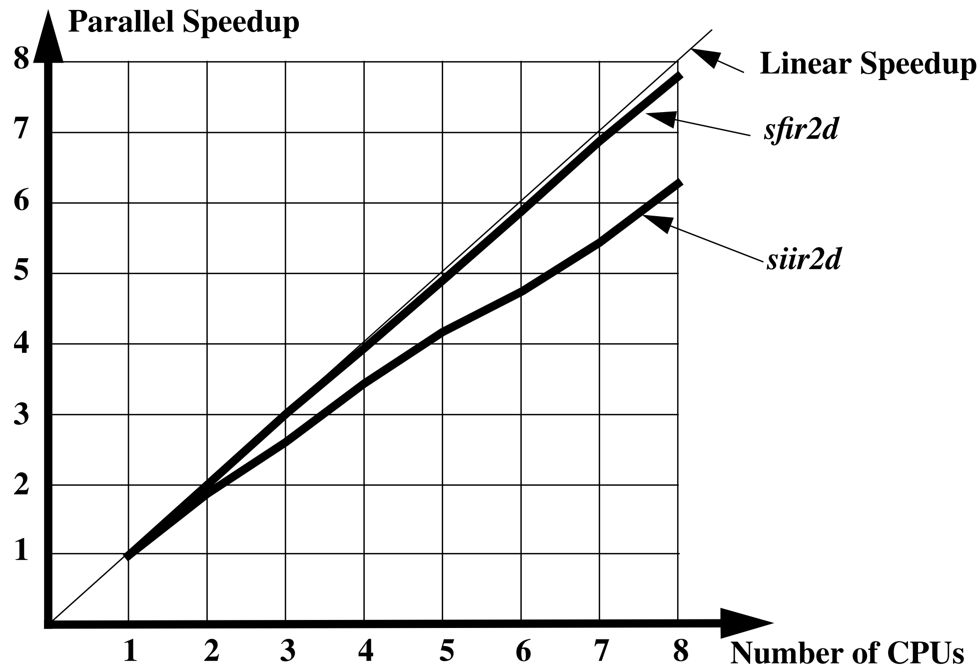

Figure 3 : sfir2d and siir2d Parallel Speedup on a 4D/380.

Appendix 1:1D FIR and IIR filters

The following Fortran program illustrates how FIR and IIR filters operate, and how one is the inverse of the other:

```
       Program ex1
******************************************************************************
*     This test program illustrates 1D Finite Impulse Response (FIR) and
*     Infinite Impulse Response (IIR) filters.
*     Double precision versions DFIR1D and DIIR1D are used
******************************************************************************
*     We first apply the IIR filter to an Impulse input
*     We use then the same filter as a FIR filter...
*     The final result: an Impulse signal !
******************************************************************************
      parameter (MAX=7)
      double precision impulse(0:MAX), filter(0:1)
      double precision out1(0:MAX), out2(0:MAX)
      impulse(0) = 1.0d0
      do i = 1, MAX
       impulse(i) = 0.0d0
      end do
      filter(0) = 1.0d0
      filter(1) = 0.3d0
*  IIR filter:
      call diir1d( impulse,1,0,MAX+1,filter,1,0,2,out1,1,0,MAX+1)
*  FIR filter:
      call dfir1d(out1,1,0,MAX+1,filter,1,0,2,out2,1,0,MAX+1,1.d0,0.d0)
      write(6,11)
      write(6,12) impulse(0:MAX)
      write(6,13)
      write(6,12) out1(0:MAX)
      write(6,14)
      write(6,12) out2(0:MAX)
 11   format('Input is an Impulse: ')
 12   format(8f9.5)
 13   format(/'After application of IIR filter:')
 14   format(/'After a final FIR filter, output is an Impulse again:')
      stop
      end
```

This example is compiled and run with the following commands:

```
% f77 -o ex1 ex1.f ../libconv.a
% ex1
Input is an Impulse:
  1.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
After application of IIR filter:
  1.00000 -0.30000  0.09000 -0.02700  0.00810 -0.00243  0.00073 -0.00022
After a final FIR filter, output is an Impulse again:
  1.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
```

Appendix 2:Convolution and Correlation

The following C code illustrates how Correlation and Convolution are related together:

```
/* This program illustrates relation between CONVOLUTION and CORRELATION
   Correlation is equivalent to convolution with Flipped array */
#include <stdio.h>
#include <math.h>
#include "conv.h"
void float_seq_print( float *f_array, int n_pts);
#define NPTS 8
main() {
    int i;
    float seq1[NPTS], seq2[NPTS], cor1[NPTS], cor2[NPTS];
/* Initialize Sequences ... seq2 is seq1 shifted by two samples */
    seq1[0] = seq1[1] = seq2[NPTS-2] = seq2[NPTS-1] = 0.;
    for (i = 2; i < NPTS ; i++)
        seq1[i] = seq2[i-2] = 10. * pow( -.3, (double)i);
/* Correlation */
    scor1d( seq1,1,0,NPTS,seq2,1,0,NPTS,cor1,1,0,NPTS);
/* Convolution with flipped array */
/* We actually just call sfir1d with the flipping parameters */
/* New origin of array is last sample, new increment is -(old increment) */
    sfir1d(seq1,1,0,NPTS,seq2+(NPTS-1),-1,-(NPTS-1),NPTS,cor2,1,0,NPTS,1.,0.);
/* Print out the Input sequences and the Correlation functions */
    printf("Seq # 1 : \n");
      float_seq_print( seq1, NPTS);
    printf("Seq # 2 : \n");
      float_seq_print( seq2, NPTS);
    printf("Correlation using \"scor1d\" : \n");
      float_seq_print( cor1, NPTS);
    printf("Correlation using \"sfir1d\" : \n");
      float_seq_print( cor2, NPTS);
}
void float_seq_print( float *f_array, int n_pts){
    for ( ; n_pts > 0; n_pts--)      printf("%9.5f", *f_array++);
    printf("\n");
}
```

The following commands will compile this program and execute it:

```
% cc -o ex2 ex2.c ../libconv.a -lm
% ex2
Seq # 1 :
  0.00000  0.00000  0.90000 -0.27000  0.08100 -0.02430  0.00729 -0.00219
Seq # 2 :
  0.90000 -0.27000  0.08100 -0.02430  0.00729 -0.00219  0.00000  0.00000
Correlation using "scor1d" :
  0.08010 -0.26703  0.89011 -0.26703  0.08010 -0.02402  0.00715 -0.00197
Correlation using "sfir1d" :
  0.08010 -0.26703  0.89011 -0.26703  0.08010 -0.02402  0.00715 -0.00197
```

Appendix 3:2D filter and Multiple 1D filters

This Fortran example computes the derivative of the input array using 2D filters or multiple 1D filters:

```
      Program ex3
*****************************************************************************
*  This program is to illustrate the use of 2D and Multiple 1D filters       *
*  We use here DFIR2D and DFIRM1D modules from LIBCONV                       *
*****************************************************************************
*  1D filter in the X direction:(1/2,-1.,1/2) Second Derivative approximation
*  1D filter in the Y direction:(-1/2,0.,1/2) First  Derivative Approximation
*  The equivalent 2D filter is:    -1/4   0   +1/4
*                                 +1/2   0   -1/2
*                                 -1/4   0   +1/4
*  This is the approximation of the triple derivative D()/Dxxy               *
*****************************************************************************
      parameter (HX=5, HY=5, NX=(2*HX+1), NY=(2*HY+1))
      double precision input(-HX:HX,-HY:HY), fil2d(-1:1,-1:1)
      double precision filx(-1:1), fily(-1:1), temp(-HX:HX,-HY:HY)
      double precision out1(-HX:HX,-HY:HY), out2(-HX:HX,-HY:HY)
      double precision scale_x, scale_y
      data fil2d/-.25,0.5,-.25,0.0,0.0,0.0,.25,-.5,.25/
      data filx /0.5,-1.,0.5/,        fily/-.5,0.0,0.5/
* Init Input
      scale_x = 180./dfloat(HX)
      scale_y = 180./dfloat(HY)
      do j = -HY,HY
        do i = -HX,HX
          input(i,j) = cosd(i*scale_x)*(1.-dfloat(ABS(i))/dfloat(HX))
     $               * cosd(j*scale_y)*(1.-dfloat(ABS(j))/dfloat(HY))
        end do
      end do
*  2D FIR filter:
      call dfir2d( input,1,NX,-HX,NX,-HY,NY,
     $             fil2d,1,3, -1,  3, -1, 3,
     $             out1, 1,NX,-HX,NX,-HY,NY, 1.0d0, 0.0d0)
*  Multiple 1D FIR filter in the X direction:
      call dfirm1d( input,1,NX,-HX,NX, NY,
     $              filx, 1,   -1, 3,
     $              temp, 1,NX,-HX,NX,       1.0d0, 0.0d0)
*  Multiple 1D FIR filter in the Y direction:
      call dfirm1d( temp,NX,1,-HY,NY, NX,
     $              fily, 1,   -1, 3,
     $              out2,NX,1,-HY,NY,        1.0d0, 0.0d0)
*  Print out the input and the two results
      write(6,11)
      write(6,12) ((input(i,j), j=-HY,HY), i=-HX,HX)
      write(6,13)
      write(6,12) ((out1(i,j), j=-HY,HY), i=-HX,HX)
```

```
      write(6,14)
      write(6,12) ((out2(i,j), j=-HY,HY), i=-HX,HX)
 11   format(‘Input : ‘)
 12   format(11f7.3)
 13   format(/’After application of 2D FIR filter:’)
 14   format(/’After multiple 1D FIR filters in the X and Y Direction:’)
      stop
      end
```

To compile and run:

```
% f77 -o ex3 ex3.f ../libconv.a
% ex3
Input :
  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
  0.000  0.026  0.020 -0.030 -0.105 -0.162 -0.105 -0.030  0.020  0.026  0.000
  0.000  0.020  0.015 -0.023 -0.080 -0.124 -0.080 -0.023  0.015  0.020  0.000
  0.000 -0.030 -0.023  0.034  0.120  0.185  0.120  0.034 -0.023 -0.030  0.000
  0.000 -0.105 -0.080  0.120  0.419  0.647  0.419  0.120 -0.080 -0.105  0.000
  0.000 -0.162 -0.124  0.185  0.647  1.000  0.647  0.185 -0.124 -0.162  0.000
  0.000 -0.105 -0.080  0.120  0.419  0.647  0.419  0.120 -0.080 -0.105  0.000
  0.000 -0.030 -0.023  0.034  0.120  0.185  0.120  0.034 -0.023 -0.030  0.000
  0.000  0.020  0.015 -0.023 -0.080 -0.124 -0.080 -0.023  0.015  0.020  0.000
  0.000  0.026  0.020 -0.030 -0.105 -0.162 -0.105 -0.030  0.020  0.026  0.000
  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000

After application of 2D FIR filter:
 -0.007 -0.005  0.014  0.031  0.033  0.000 -0.033 -0.031 -0.014  0.005  0.007
  0.008  0.006 -0.017 -0.039 -0.041  0.000  0.041  0.039  0.017 -0.006 -0.008
  0.011  0.008 -0.024 -0.052 -0.055  0.000  0.055  0.052  0.024 -0.008 -0.011
  0.006  0.005 -0.013 -0.029 -0.031  0.000  0.031  0.029  0.013 -0.005 -0.006
 -0.004 -0.003  0.009  0.021  0.022  0.000 -0.022 -0.021 -0.009  0.003  0.004
 -0.029 -0.022  0.061  0.136  0.144  0.000 -0.144 -0.136 -0.061  0.022  0.029
 -0.004 -0.003  0.009  0.021  0.022  0.000 -0.022 -0.021 -0.009  0.003  0.004
  0.006  0.005 -0.013 -0.029 -0.031  0.000  0.031  0.029  0.013 -0.005 -0.006
  0.011  0.008 -0.024 -0.052 -0.055  0.000  0.055  0.052  0.024 -0.008 -0.011
  0.008  0.006 -0.017 -0.039 -0.041  0.000  0.041  0.039  0.017 -0.006 -0.008
 -0.007 -0.005  0.014  0.031  0.033  0.000 -0.033 -0.031 -0.014  0.005  0.007

After multiple 1D FIR filters in the X and Y Direction:
 -0.007 -0.005  0.014  0.031  0.033  0.000 -0.033 -0.031 -0.014  0.005  0.007
  0.008  0.006 -0.017 -0.039 -0.041  0.000  0.041  0.039  0.017 -0.006 -0.008
  0.011  0.008 -0.024 -0.052 -0.055  0.000  0.055  0.052  0.024 -0.008 -0.011
  0.006  0.005 -0.013 -0.029 -0.031  0.000  0.031  0.029  0.013 -0.005 -0.006
 -0.004 -0.003  0.009  0.021  0.022  0.000 -0.022 -0.021 -0.009  0.003  0.004
 -0.029 -0.022  0.061  0.136  0.144  0.000 -0.144 -0.136 -0.061  0.022  0.029
 -0.004 -0.003  0.009  0.021  0.022  0.000 -0.022 -0.021 -0.009  0.003  0.004
  0.006  0.005 -0.013 -0.029 -0.031  0.000  0.031  0.029  0.013 -0.005 -0.006
  0.011  0.008 -0.024 -0.052 -0.055  0.000  0.055  0.052  0.024 -0.008 -0.011
  0.008  0.006 -0.017 -0.039 -0.041  0.000  0.041  0.039  0.017 -0.006 -0.008
 -0.007 -0.005  0.014  0.031  0.033  0.000 -0.033 -0.031 -0.014  0.005  0.007
```