TSL Note-2002-53

F-Tools, Signal Processing on the Command Line

V. Ziemann
The Svedberg Laboratory, S-75121 Uppsala, Sweden

April 25, 2002

Abstract

We describe a set of tools that communicate via the standard Unix pipe concept to perform various digital signal processing algorithms.

1 Introduction

In this report we describe a set of programs that perform various digital signal processing tasks such as filtering, digitizing, mixing and also the generation of the filters and data visualization. All programs communicate via standard input and standard output, i.e. they read from the keyboard and write to the screen. Thus making it possible to use the Unix feature of input redirection and piping to string small programs together on the command line to build more complex algorithms. The individual small programs communicate via Unix-pipes, just like other Unix tools such as `awk`, `grep`, or `sed` do. Having defined this very simple interface it is very easy to extend the program suite and one can do it in any programming language or shell scripting language making the approach very flexible. Furthermore, interfacing to other data generating source such as data acquisition devices or simulation programs is very easy. All that the data generating sources need to do is write their data stream to standard output and then all tools of this program suite are available for use.

This approach using small simple programs that communicate via standard in- and output can be contrasted to the use of large stand-alone programs like MATLAB [1] that provide a convenient environment developed

and analyzed signal processing algorithms. They provide a shell to do ones tasks, but a toolbox of small, well communicating, programs provides a more flexible way of handling a large variety of data processing tasks.

In the next section we will present the small programs that constitute the `f_tools` program suite. That section serves as a reference guide to the programs and explains the command line options. It is followed by a tutorial showing some examples what one can do with the tools and how to combine them. Some more technical aspects like how to obtain and how to install the programs are deferred to appendices.

2 The Programs

Here we describe the individual programs, their features and parameters. All programs follow the rule that if no command line parameters are specified, a short usage note is displayed and the program exits. We will use the convention to use `<int>` and `<float>` to describe an integer or float parameter, respectively. A file name as input parameter will be denoted by `<file>`.

2.1 f_sine

This program creates a single sine wave where amplitude, frequency and number of data points can be specified as arguments:

- `-am <float>`
  The float value `<float>` denotes the amplitude of the sine wave.

- `-fr <float>`
  The float value `<float>` denotes the frequency in units of the sampling frequency. It only makes sense to use values in the base band between 0 and 0.5, all others will be aliased thither.

- `-le <int>`
  Number of data points written to output.

- `-pipe`
  Reads samples from standard input and adds to it.

2.2 f_wobble

This program creates a modulated sine wave where amplitude, frequency as well as wobble amplitude and frequency can be specified

- `-am <float>`
  The float value `<float>` denotes the amplitude of the carrier sine wave.
- `-fr <float>`
  The float value `<float>` denotes the frequency of the carrier in units of the sampling frequency. It only makes sense to use values in the base band between 0 and 0.5, all others will be aliased thither.
- `-da <float>`
  Wobble amplitude.
- `-df <int>`
  Wobble frequency.
- `-pipe`
  Reads samples from standard input and adds to it.

2.3 f_noise

This program can add random numbers or noise to the the data stream read from standard input or create a stream of random numbers that are printed to standard output if the `-source` option is specified.

- `-am <float>`
  Amplitude of the noise added or created.
- `-source`
  Will not read from standard input but just create an output stream of random numbers.
- `-seed <int>`
  A new seed for the pseudo-random number generator.
- `-nch <int>`
  The number of channels (or columns) that are written to standard output.

- `-trunc <float>`
  Truncate the gaussian at this number of standatd deviations.

- `-if <file>`
  In this way one can specify to read from file `<file>` instead from standard input.

2.4 f_ping

This program writes a single "1" to standard output followed by a number of zeroes. It can be used to test e.g. the impulse response of digital filters.

- `-le <int>`
  The total number of output data written to standard output.

2.5 f_pulse

A simple pulse generator that produces rectangular pulses where the user can specify the number of samples at a given amplitude.

- `-am1 <float>`
  The amplitude at time slot 1 (default=1)

- `-am2 <float>`
  The amplitude at time slot 2 (default=0)

- `-t1 <int>`
  The length in samples of time slot 1 (default=10);

- `-t2 <int>`
  The length in samples of time slot 2 (default=10);

2.6 f_mix

This program reads a data stream from standard input and multiplies it with a self-generated sine wave whose amplitude and frequency can be specified on the command line. Electrical engineers call such a thing a mixer.

- `-am <float>`
  Amplitude of the "local oscillator".

- `-fr <float>`
  Frequency of the "local oscillator".

- `-if <file>`
  Instead of reading from standard input the program can read from a file specified by the `-if` command line argument.

2.7 f_fir

This program filters the input stream read from standard input by convoluting it with a FIR filter, whose coefficients are specified in a file.

- `-ff <file>`
  The file that contains the filter coefficients. Note that the number of coefficients of the filter is determined auto-magically.

- `-if <file>`
  Read from file instead of reading from standard input.

2.8 f_iir

This program filters the input stream read from standard input by convoluting it with a IIR filter, whose coefficients are specified in a file.

- `-ff <file>`
  The file that contains the filter coefficients. Note that the number of coefficients of the filter is determined auto-magically.

The format of the input file is very simple. A hash-mark specifies comment lines, first the a-coefficients are specified one per line and terminated by a single line that contains an ampersand (&). Then the b-coefficients follow one per line. Note that we use the convention that the b-coefficients *always* begin with $b(0)$ which typically is unity and the other coefficients are positive in the transfer-function. In other words, we completely specify all polynomial coefficients of the transfer-function. Note that this implies that we need to use a minus-sign in the code of the IIR-filter as shown here

$$y(n) = \sum_{i=0}^{n} a(i)x(n-i) - \sum_{i=1}^{m} b(i)y(n-i) \ . \tag{1}$$

This is also the convention followed in [2].

2.9 f_dec

This program is a close relative to `f_fir`. Beyond doing the same FIR filtering described there it can discard a specifiable number of data points between those written to output. It can be used to implement multi-rate filters.

- `-ff <file>`
  The file containing the filter coefficients.

- `-dec <int>`
  The decimation ratio. If the specified integer is for example 5 this means that only every fifth output data point is passed to standard output. In this way a reduction of the sample rate is achieved.

- `-if <file>`
  Instead of reading from standard input the program can read from a file specified by the `-if` command line argument.

2.10 f_fft

This program performs a Fast Fourier Transform on the data read from a file and writes the amplitude data to standard output such that it can be displayed in any normal display program, such as `xmgr` or friends. Note that this program works on any length input data. After determining how many data points are read the next bigger power of two is determined and the unspecified data points padded by zeroes.

- `-f <file>`
  Specifies the input file to use that contains the time domain input data.

- `-r`
  Rescales the output by $2/n$.

- `-x`
  Scales the x-axis to extend from 0 to 0.5.

- `-dB`
  display the output in decibel. Good for filter characterization.

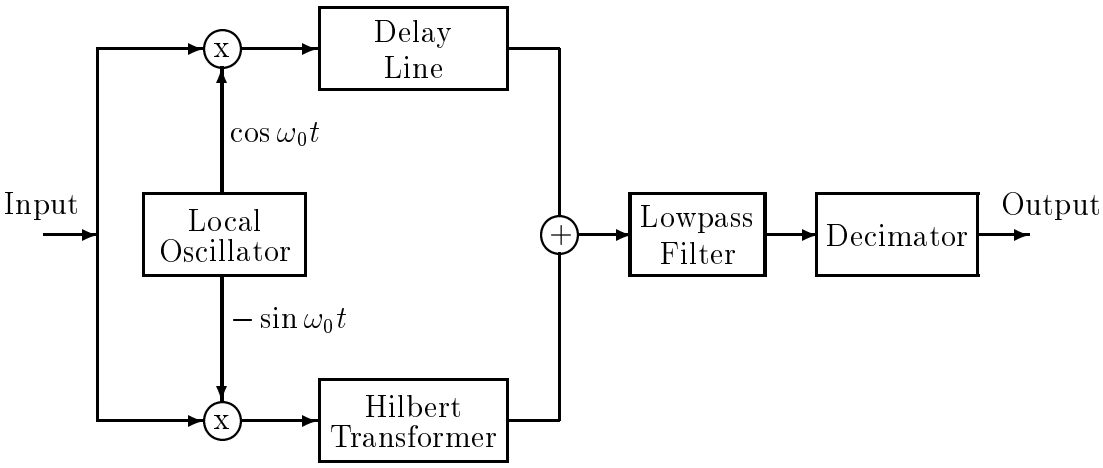

Figure 1: *An image reject mixer.*

2.11 `f_undersample`

This program can be used to reduce the sampling rate by discarding a fixed number of data points from the input stream between those passed on to output.

- `-ra <int>`
  The decimation rate.

- `-if <file>`
  Read from file instead of reading from standard input.

2.12 `f_irm`

This program implements an *image reject mixer* that only mixes the upper side band down to baseband with the help of a Hilbert transformer and a local oscillator that provides both in-phase and quadrature components. On top of the mixing this program also provides low-pass filtering and decimation. The user can specify the mixing frequency, the filter file and the decimation rate.

- `-fr <float>`
  The mixing frequency (default=0.123).

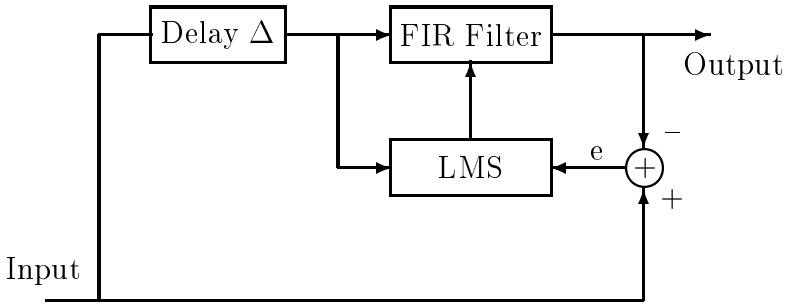

Figure 2: *An adaptive line enhancer*

- `-dec <int>`
  The decimation factor (default=1, no decimation).

- `-le <int>`
  The length or number of taps of the Hilbert transformer used to make the 90 degree phase shift (default=71).

- `-ff <file>`
  The fir filter (typically a low-pass) used to remove the high frequency components that would alias when decimating. This filter must be specified.

2.13 `f_ale`

This program implements an *adaptive line enhancer* that picks up coherent signals in noise and amplifies them using and LMS algorithm that is used to build a fir-filter. The filter coefficients are dumped into a file after every million samples processed. There are he following command line options

- `-nf <int>`
  number of taps of the fir filter (default=71).

- `-de <int>`
  delay used (default=3).

- `-mu <float>`
  learning parameter (default=1e-6).

- `-ga <float>`
  leakage parameter default=1 (no leakage).

- `-init <int>`
  if the specified integer is larger than zero it defines the number of samples after which the filter and working arrays are reinitialized to zero. This is useful if dealing with signals varying with time, because once a filter is found it is very difficult to latch on to another frequency. The specified number of samples should be a few times the number of samples needed for the filter to converge. Try 10 000 for a start, then increase or decrease.

- `-off`
  bypasses the algorithm and passes the input samples straight through to the output. Useful to turn the algorithm off temporarily.

2.14 f_histogram

This program receives a data stream from standard input and sorts the samples in a histogram. Every so often it outputs the resulting histogram to standard output such that it can be viewed in `f_scope`.

- `-min <float>`
  Lower limit of the histogram (default=-5).

- `-max <float>`
  Upper limit of the histogram (default=5).

- `-nbin <int>`
  Number of bins of the histogram (default=64).

- `-npts <int>`
  Dump the cumulative histogram every `npts` samples (default=512).

- `-mult <int>`
  Determines how often each point is printed to standard output (default=8). If `mult` times `nbin` equals `npts` the display on `f_scope` will be stationary.

2.15 f_trigger

This program receives a data stream from standard input until a trigger event that can be specified by amplitude and slope occurs. Then it passes a number of data points to standard output.

- `-am <float>`
  The amplitude at which to trigger.
- `-sl <float>`
  The slack or vertical accuracy to which a data sample has to hit the trigger threshold. Default is 0.05.
- `-pos`
  Requires positive slope at the trigger point. This is the default.
- `-neg`
  Requires negative slope at the trigger point.
- `-npts <int>`
  The number of points passed to standard output after a trigger event occurred. Default is 512.
- `-if <file>`
  Read from file instead of reading from standard input.

2.16 f_make_filter

This program can be used to generate FIR filter coefficients that are written to standard output. The length, bandwidth, and type of filter can be specified on the command line.

- `-lowpass`
  Make filter coefficients for a low-pass filter.
- `-bandpass`
  Make filter coefficients for a band-pass filter.
- `-bandstop`
  Make filter coefficients for a band-stop filter.

- `-highpass`
  Make filter coefficients for a high-pass filter.

- `-le <int>`
  The length of the filter or number of taps.

- `-ce <float>`
  Center frequency in units of the sample rate. The useful range is between 0 and 0.5, that is, in the baseband.

- `-bw <float>`
  Band width within the range 0 and 0.5.

2.17 f_remez_lowpass

This program calculates the coefficients of the an optimal low-pass filter using the Parks-McClellan algorithm and the remez exchange method. Equal weight is put on optimizing the passband to unity and the stopband to zero. Another program like `f_remez_estimate` has to be used to determine the number of taps that will result in the desired ripple in the pass and stop band. This routine requires the `mkfilter` package from Ref. [5] to be installed.

- `-N <int>`
  Number of coefficients or taps of the FIR filter.

- `-bw <float>`
  The band edge in units of the sampling rate of the optimal low-pass filter.

- `-df <float>`
  The width of the transition band in units of the sampling rate.

2.18 f_remez_estimate

This program is used to estimate the length of an optimal Parks-McClellan-Remez filter for given transition band width and ripple in pass and stop band.

- `-df <float>`
  The width of the transition band in units of the sampling rate.

- `-dp <float>`
  The pass band ripple in linear scale, not dB.

- `-ds <float>`
  The stop band ripple in linear scale, not dB.

2.19 `f_window`

Give an input file with FIR filter coefficients this program will multiply the filter coefficients with a window function. Several window functions can be specified on the command line. The windowed filter coefficients are written to standard output.

- `-ff <file>`
  Specify the input file with the filter coefficients.

- `-mode <int>`
  Specify the type of window in numerical form, 0=none, 1=Bartlett, 2=Hanning, 3=Hamming, 4=Blackman.

- `-help`
  Short help is displayed to standard output explaining the command line options.

- `-none`
  Do not apply a window function. Equivalent to `-mode 0`.

- `-bartlett`
  Bartlett Window, equivalent to `-mode 1`.

- `-hanning`
  Hanning Window, equivalent to `-mode 2`.

- `-hanning`
  Hamming Window, equivalent to `-mode 3`.

- `-blackman`
  Blackman Window, equivalent to `-mode 4`.

2.20 f_make_iir

This program calculates the coefficients of simple IIR filters following Ref. [3].

- `-lowpass1`
  First order low-pass filter.
- `-highpass1`
  First order high-pass filter.
- `-x <float>`
  Design parameter of the first order filters (default=0.8).
- `-bandpass`
  Second order band-pass filter.
- `-bandstop`
  Second order band-stop filter.
- `-ce <float>`
  Center frequency of the second order filters (default=0.2).
- `-bw <float>`
  Bandwidth of the second order filters (default=0.1).

2.21 f_make_chebychev

This program calculates the coefficients of IIR Chebychev-filters following Ref. [3]. The number of poles must be even.

- `-lp`
  Chooses low-pass filter (default)
- `-hp`
  Chooses high-pass filter
- `-pr <float>`
  The passband ripple specified in percent (default=1).
- `-bw <float>`
  The cutoff frequency of the filter (default=0.1).
- `-np <int>`
  The number of poles used in the filter (default=4).

2.22 tf_mult

This program calculates the filter that results in cascading two IIR filters, which is equivalent of multiplying the transfer functions. It takes the two original IIR-filter file-names as command line parameters and returns the result to standard output such that it can be redirected into a file for further use.

2.23 tf_add

This program calculates the filter that results in passing a signal in parallel through two filters. Otherwise it behaves identically to `tf_mult`.

2.24 f_split

This program takes a single data stream and converts it into `nch` identical data streams, possibly all multiplied by the same factor. This programs is useful to multiply a single data stream into three, such that one can use `f_scope` to display the wave form, the spectrum in logarithmic and linear scale at the same time.

- `-nch <int>`
  The number of output data streams (default=1).
- `-am <float>`
  The output stream data are multiplied by this factor (default=1).

2.25 f_scope

This program is the primary viewer of data in this software suite. It can display up to four traces read from standard input both in the time and frequency domain, i.e. it can operate as a spectrum analyzer. For the spectrum calculations the FFT routine from Ref. [4] is used and for the X-windows interface we use the libsx-library [6]. In order to control the multitude of options a number of command line switches, which can be specified in any order, are available to fine tune the behavior of the program.

- `-npts <int>`
  `<int>` defines the number of data points to be considered at a time, in, for example, Fourier transforms, defaults to 512.

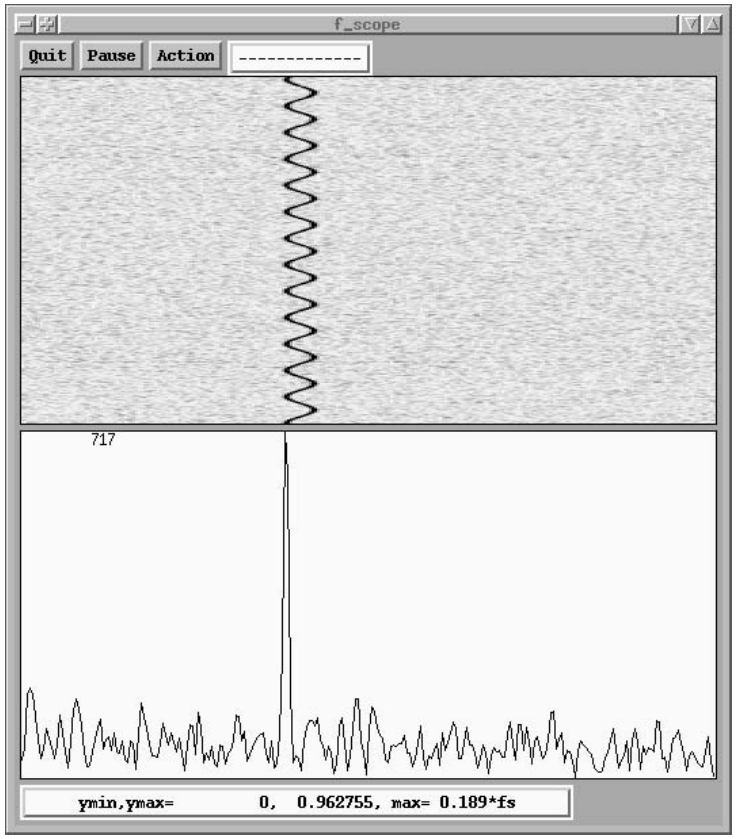

Figure 3: *The* `f_scope` *display window with the trace display on the bottom and the spectrogram on the top.*

- `-ndisp`
  `<int>` defines the width of the display in pixels, defaults to 512.

- `-file <file>`
  Makes `f_scope` read input data from file `<file>` rather than from standard input.

- `-sink`
  In the default configuration `f_scope` passes the input data on to standard output to allow inserting it in the middle of a data stream and monitoring intermediate signals. If, on the other hand, `-sink` is specified on the command line, the output data stream is discarded.

- `-aver_f <int>`
  In spectrum analyzer mode specifying this option will average `<int>` neighboring frequency bins before displaying the spectrum.

- `-aver_t <int>`
  In spectrum analyzer mode this option will perform a running (IIR-like) average over `<int>` consecutively acquired spectra.

- `-nch <int>`
  The integer `<int>` specifies the number of input channels , i.e. numbers of data columns in the input data stream.

- `-ich 0xedcba`
  This option lets the user specify a bit pattern that controls special options such a logarithmic display for each of the four data channels independently. Each of `a,b,...` represent a single hexadecimal number (0,1,2,...,E,F) which contains four individual bits. The least significant bit controls the first data channel, the next bit the second and so forth. The individual hexadecimal numbers are explained here:

  a: Display of specific trace on or off.

  b: Time domain or spectrum display.

  c: Logarithmic or linear scale.

  d: Auto-scale on or off.

  e: Colorfall display on or off.

- `-log`
  Sets all displays to logarithmic.

- `-spectrum`
  Sets all traces to display the spectrum.

- `-colorfall`
  Sets all traces to also display a colorfall.

- `-vsize1 <int>`
  Sets the vertical size (in pixel) of the upper colorfall-window. The default size is 256 pixel.

- `-nodc <int>`
  When displaying a spectrum this feature sets the first `<int>` pixels to zero and thus allows increasing the dynamic range of the auto-scaling when a large DC component is present.

- `-record <file>`
  When colorfall display is turned on this option opens a data file `<file>` in which the trace information is recorded in binary form. The format is one byte per horizontal pixel. In the default configuration 512 byte are written for every trace displayed.

- `-replay <file>`
  Replays a file `<file>` saved with the `-record` option.

Besides command line options the program's behavior can also be controlled a run-time by a selection of buttons and menus. On the top row a `Quit` button allows the user to terminate the program. To the right is a button that allows the user to `Pause` and `Start` the display. The menu labeled `Action` open a menu with the following choices

- `Edit Parameters` opens another window in which the following parameters can be edited at run time

  - `Timeout` is the time in ms between displaying consecutive frames.
  - `Ymin, Ymax` are the lower and upper limits of the display. These are only used as limits if `Autoscale` is not selected.
  - `Aver_time` is the same as the `-aver_t` parameter that can be set on the command line and controls the averaging over consecutive spectra.
  - `Aver_freq` is the same as the `-aver_f` parameter that can be set on the command line and controls the averaging over frequency bins.
  - `ich` is the same as the `-ich` parameter that can be set on the command line.
  - `zoom pixel` allows the user to zoom in on a specific pixel range in the frequency domain.

Since some parameters can also be edited by hot-keys the `Reload` button allows that all displayed values are updated. The `Dump` button dumps the currently visible traces into an ASCII file.

- `Toggle Auto-scale` turns auto-scaling on and off.
- `Waveform` selects that all traces are displayed in waveform.
- `Spectrum` selects that all traces are displayed as spectra.
- `Toggle Log Display` selects that all traces are displayed in logarithmic form.

The `f_scope` program also supports a selection of hot-keys that trigger various actions if the cursor is in the lower display window.

1,2,3,4 toggles whether the first,second,third,forth trace is displayed or not.

q,w,e,r toggles whether the first,second,third,forth trace are displayed as waveform or spectrum.

a,s,d,f toggles whether the first,second,third,forth trace are displayed with logarithmic scale if in spectrum mode.

z,x,c,v toggles whether the first,second,third,forth trace are displayed with auto-scale or fixed scale.

F1,F2,F3,F4 toggles whether the first,second,third,forth trace are used in the colorfall display in the top window.

o dumps the contents of all `nch` traces into an ASCII file that can be imported into any standard viewing program.

p toggles zoom mode on and off.

Finally it should be noted that the system of programs is easily extendable using almost any computer language, such as C or Fortran, but also using shell scripting languages such as `bash` or `awk`.

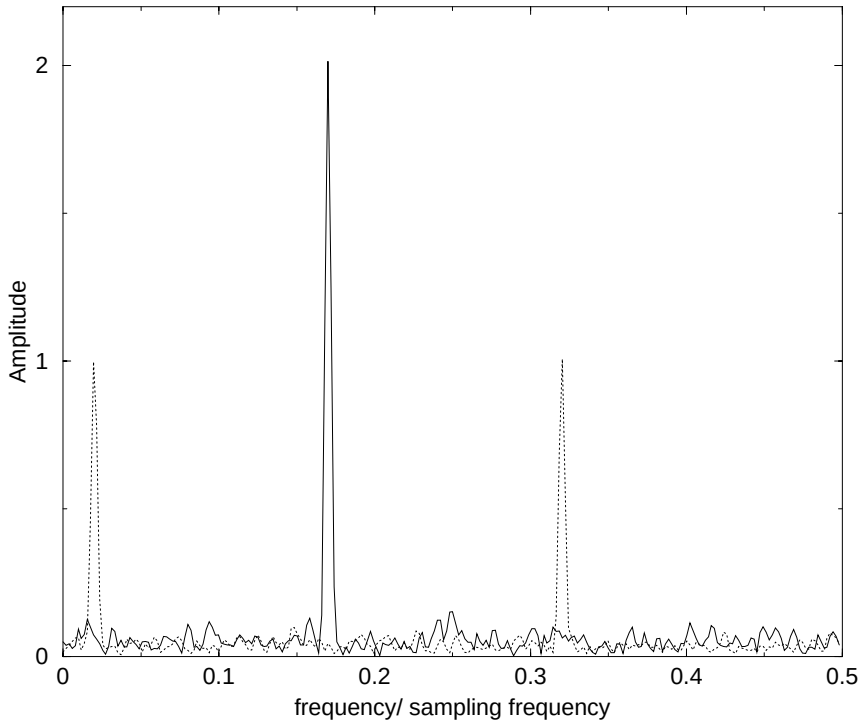

Figure 4: *The spectrum of a sine wave with frequency 0.17, added noise is shown as a solid line. The dashed lines shows the same signal after mixing with a signal with frequency 0.15.*

3 Tutorial

Here we will show how the programs can be combined to generate more complex applications and to test various filters.

A simple example is a sine wave that is displayed on a spectrum analyzer. We simply use `f_sine` to generate a data stream with a frequency of 0.17 times the sampling rate and an amplitude of 2, then `f_scope` is used to display the data stream

```
f_sine -fr 0.17 -am 2 | f_scope -spectrum -sink
```

where we specified the `-sink` command line argument to prevent `f_scope` to pass the data stream on to standard output and the `-spectrum` command line argument to display the spectrum rather than the waveform of the input data stream.

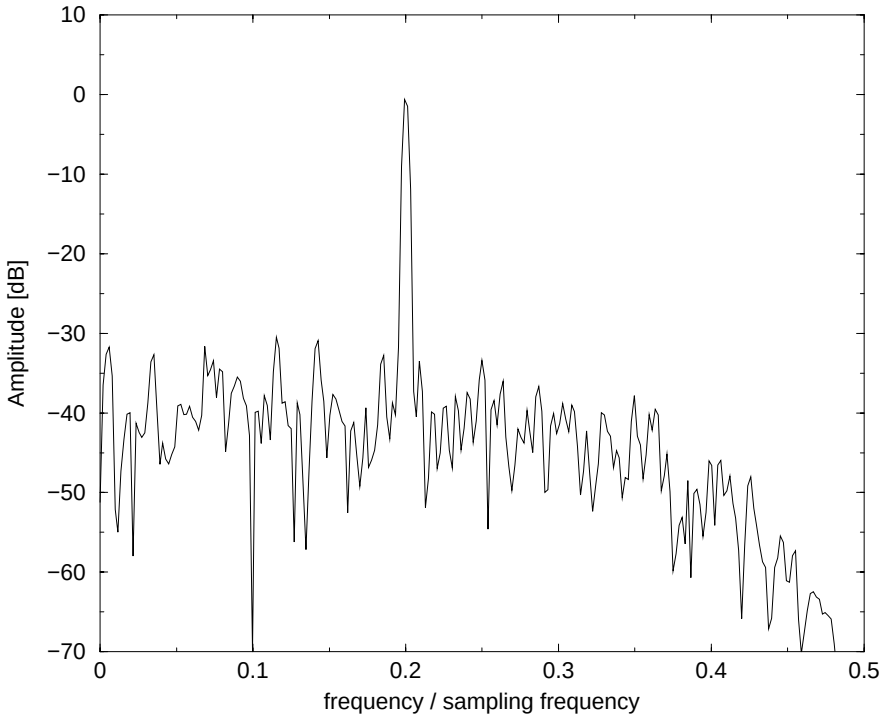

Figure 5: *...and after low-pass filtering and under-sampling by a factor 10.*

Adding a little noise is achieved by simply sandwiching the `f_noise` between the `f_sine` and `f_scope`

```
f_sine -fr 0.17 -am 2 | f_noise -am 0.5\
                      | f_scope -spectrum -sink
```

The resulting spectrum is shown as the solid line in Fig. 4. We can now proceed an mix the resulting signal with another sine signal that has the frequency 0.15 times the sampling rate and obtain two signals at 0.02 and 0.32 times the sampling rate which have only half the amplitude as can be expected from the multiplication of cosines

```
f_sine -fr 0.17 -am 2 | f_noise -am 0.5\
                      | f_mix -fr 0.15\
                      | f_scope -spectrum -sink
```

The result is shown as the dashed line in Fig. 4.

We may now be interested in a closer look at the signal at 0.02. This can be done by low-pass filtering. To this end we need to generate a low-pass filter that has a cutoff frequency around 0.05 which we can easily construct by the following command.

```
f_remez_lowpass -N 91 -bw 0.02 -df 0.03 > lp-0.05.fir
```

This command generates a FIR filter with 91 coefficients that has a cutoff frequency of 0.02 and a transition region between pass-band and stop-band of 0.03 and saves the filter coefficients in a file named `lp-0.05.fir`. This filter file we can now use to remove all high frequency contributions from the dashed spectrum in Fig. 4 with the command

```
f_sine -fr 0.17 -am 2 | f_noise -am 0.5\
                      | f_mix -fr 0.15\
                      | f_fir -ff lp-0.05.fir\
                      | f_scope -spectrum -sink
```

Having removed – or at least significantly attenuated – the frequencies above 0.05 we can decimate the signal by discarding 9 out of 10 data samples and thereby reducing the sample rate by a factor 10. This is achieved by adding an under-sampling program `f_undersample` to the processing pipeline

```
f_sine -fr 0.17 -am 2 | f_noise -am 0.5\
                      | f_mix -fr 0.15\
                      | f_fir -ff lp-0.05.fir\
                      | f_undersample -ra 10\
                      | f_scope -spectrum -sink
```

The resulting spectrum is shown in Fig. 5, where we see that the peak is now at frequency 0.2 which is ten times the original frequency of 0.02 due to the decimation process. We can also see the influence of the filter at frequencies larger than 0.4. Note also that we now display the signal strength in decibel. The three-stage process of mixing, low-pass filtering and under-sampling is colloquially called zoomed-FFT.

Since the filtering and under-sampling stage, which is also called decimation, is such a convenient item there also exists a program, called `f_dec`, that performs it in a single stage, such that the previous command could be written as

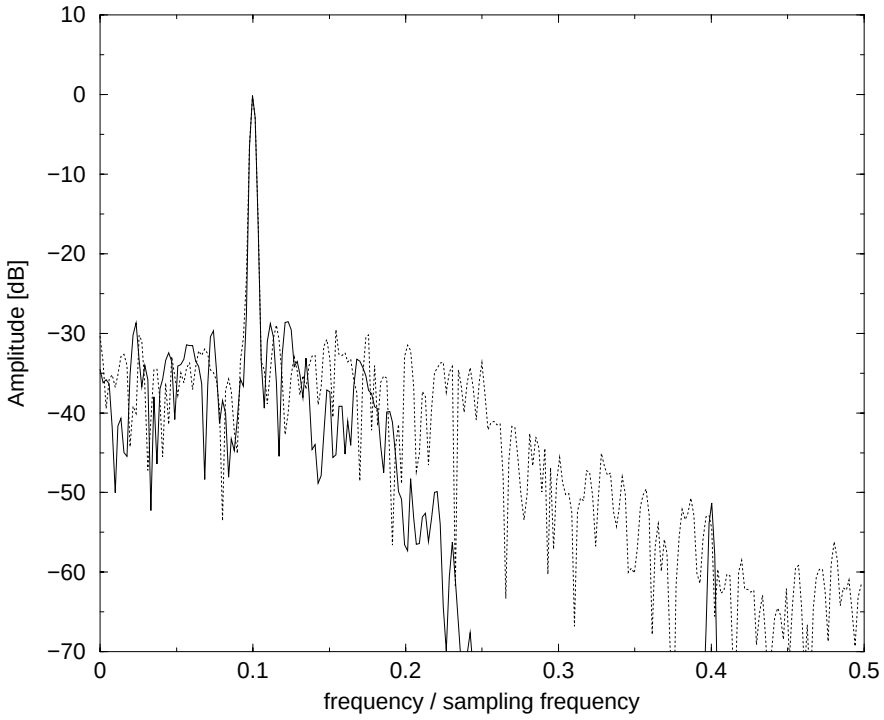

Figure 6: *The spectrum from the same input data as in Fig. 5, but using an IIR low-pass filter and a decimation factor 5. Note the small peak at 0.4 which is an alias of the line at 0.32 that passed through the filter.*

```
f_sine -fr 0.17 -am 2 | f_noise -am 0.5\
                      | f_mix -fr 0.15\
                      | f_dec -ff lp-0.05.fir -dec 10\
                      | f_scope -spectrum -sink
```

In the previous examples we have used a FIR filter, but we can of course also use an IIR filter. We know that building a filter of comparable quality as the FIR filter above with its very sharp decay is difficult with IIR filters, so we settle for a more moderate low-pass filter. We create a four-pole Chebychev filter with 1 % passband ripple and a 3 dB cutoff of 0.05 times the sampling frequency by running the command

```
f_make_chebychev -pr 1 -np 4 -bw 0.05 > lp-0.05.iir
```

and use it together with a times-5 under-sampling to investigate the low frequency mixing product.

```
f_sine -fr 0.17 -am 2 | f_noise -am 0.5\
                      | f_mix -fr 0.15\
                      | f_iir -ff lp-0.05.fir\
                      | f_undersample -ra 5\
                      | f_scope -spectrum -sink
```

The resulting spectrum is shown in Fig. 6 as the dashed line. For comparison the solid line shows data from Fig. 5 with a decimation factor 5 instead of 10. We see the main peak that was at 0.02 shifted by the decimation factor 5 to 0.1. At higher frequencies the FIR filter (solid) has a sharp decay and the IIR filter rolls off much more slowly. At 0.4 the FIR shows the peak that was originally at 0.32 and was imperfectly attenuated by the filter. The slow roll-off of the IIR makes this peak vanish in the noise floor. Note that the alias is about 50 dB lower than the main peak, so that the filters are quite good in any case.

This brings us to the question of the quality of the filters that we generated. We can analyze it by passing a unit pulse through the filter, recording it for some time to allow it to decay to zero and Fourier-transforming the output. This is done by the command line sequence

```
f_ping -le 512 | f_iir -ff lp-0.05.iir > tmp.tmp
f_fft -x -f tmp.tmp | xmgr -source stdin
```

where `f_ping` generates the unit pulse of length 1024 samples which is passed through the IIR filter and recorded in the temporary file `tmp.tmp` that is subsequently Fourier analyzed and piped into the display program `xmgr`. Repeating the same exercise with the FIR filter created above we can compare the frequency response of the filters in Fig. 7. We can see that the FIR filter – shown in dashed lines – has a much steeper fall-off than the IIR filter – shown as a solid line. The IIR filter also has a much more pronounced tail. Instead of `xmgr`[7] any other program that can display simple traces can be used.

The filter's characteristics can also be tested in a more direct way by filtering white noise which is achieved by the command line

```
f_noise -source -am 10 | f_iir -ff lp-0.05.iir\
                       | f_scope -- -aver_t 30
```

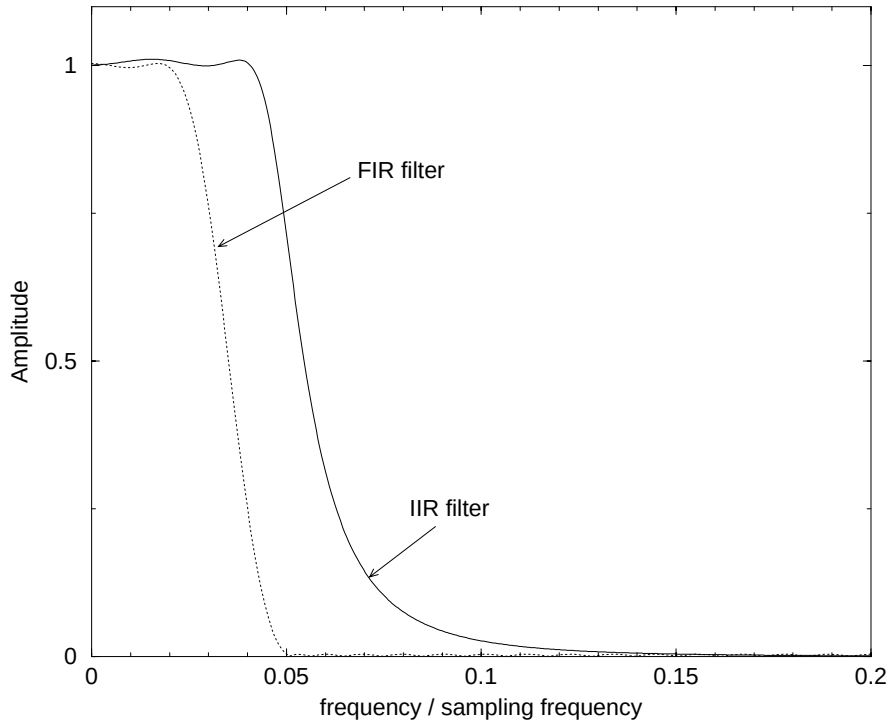

Figure 7: *The filters.*

Here we use the noise generator `f_noise` as a source and generate gaussian white noise with rms amplitude 10, pass it through the IIR filter discussed above and display it using `f_scope`. Note that we use the abbreviation `--` for `-sink -spectrum` and that we do an exponential average over 30 consecutive spectra. The comparison of the filter response is shown in Fig. 8 where we display the impulse response already shown in Fig. 7 as a dashed line and the result from the "filtered noise" approach as a solid line. We see that the curves agree quite well.

Here we briefly report the command that leads to Fig. 3. We use the sine wave modulator `f_wobble` and add a noise and then display

```
f_wobble -fr 0.2 -df 0.0001 -da 0.01\
                | f_noise -am 1\
                | f_scope --
```

Initially the `f_tools` toolbox was written to test different methods in

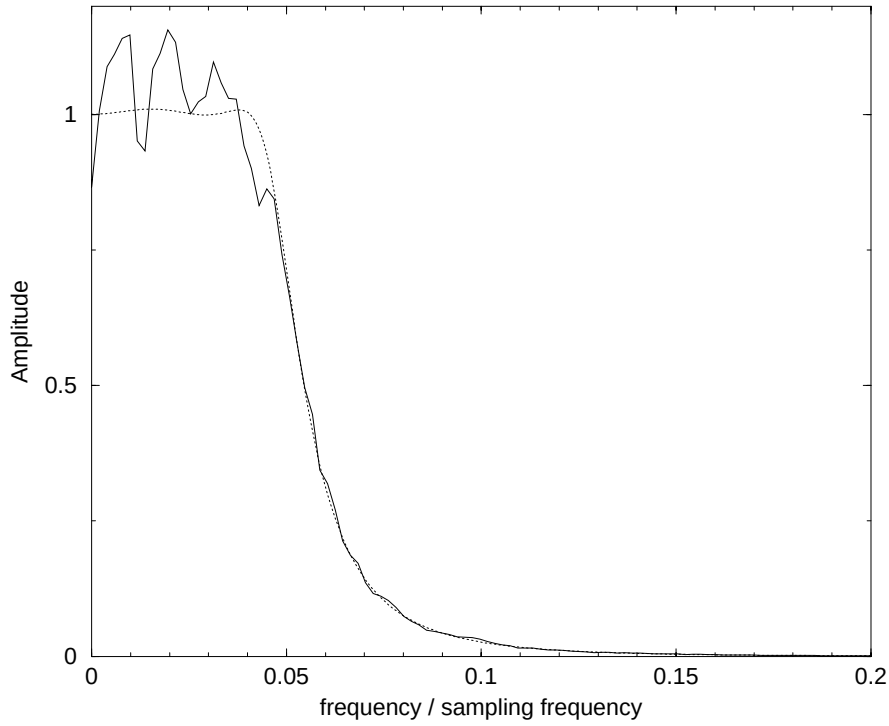

Figure 8: *Comparing the impulse response of the IIR filter (dashed) with the "filtered noise" version (solid).*

order to extract weak coherent signals in a noisy background as is further discussed in Ref. [8]. For this purpose the *image rejection mixer* and the *adaptive line enhancer* (ALE) were included and will briefly be discussed in the following paragraphs.

The image rejection mixer has the advantage of mixing only one sideband, either the upper *or* the lower, of the local oscillator frequency is mixed. This is advantageous if one wants to analyze a weak signal in the presence of a strong one. Putting the local oscillator frequency between that of the weak and the strong signal and using an image rejection mixer it is possible to suppress the strong signal. We test this setup with

```
f_sine -fr 0.11 | f_sine -pipe -fr 0.16\
                | f_scope --
```

This generates a data stream with two super-imposed sines with frequencies

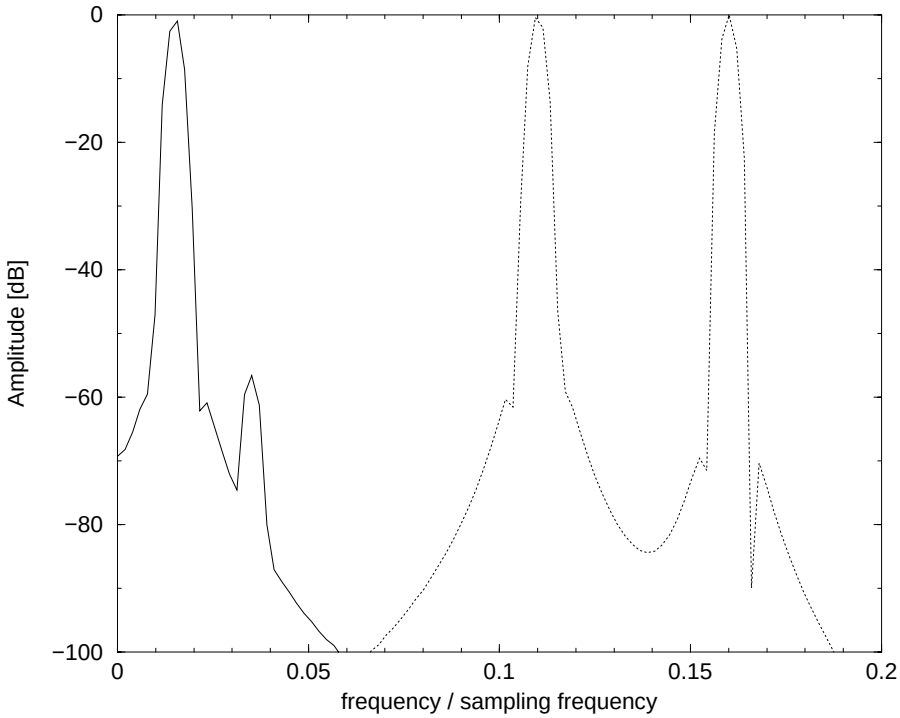

Figure 9: *The dashed line shows two sines with frequencies 0.110 and 0.160 and the solid line the spectrum after passing the signal through an image reject mixer with local oscillator frequency 0.145. Note that the lower sideband is aliased into the positive frequency range and is visible at 0.035, but attenuated by almost 60 dB.*

0.11 and 0.16 times the sample rate. The resulting spectrum is shown as the dashed line in Fig. 9. We then include the image reject mixer with a mixing frequency of 0.145 and a low-pass filter with cutoff frequency 0.1. The following command line does the job

```
f_sine -fr 0.11 | f_sine -pipe -fr 0.16\
                | f_irm -fr 0.145 -ff lp-0.1.dat\
                | f_scope --
```

We then display the spectrum as the solid line in Fig. 9. The upper sideband at $0.015 = 0.160 - 0.145$ is very pronounced but the lower sideband which appears at $0.035 = -(0.110 - 0.145)$ is attenuated by almost 60 dB or by a

factor 1000. If we had used a normal mixer the lower sideband had had the same magnitude as the upper sideband.

We illustrate the usefulness of the image rejection mixer in another example where we modulate the carrier frequency 0.222 by the small amount 0.001 such that the carrier oscillates between 0.221 and 0.223. One period of such an oscillation takes $1/10^{-5} = 10^5$ turns Running the command

```
f_wobble -fr 0.222 -da 0.001 -df 1e-5 | f_scope --
```

will show that the slow variation of the frequency is invisible. If we, however, use the image rejection mixer with a mixing frequency of 0.22 and a by-5 decimation stage will map the frequency to 0.002 and expand the frequency scale by a factor 5 such that the center frequency is now at 0.01. Adding another decimation stage with a by-10 decimator will expand the scale by another factor 10 such that the total expansion factor is 50. The resulting oscillation will be with an amplitude 0f 0.050 around a center frequency of 0.1 as can be verified by the following command line.

```
f_wobble -fr 0.222 -da 0.001 -df 1e-5\
     | f_irm -le 51 -dec 5 -fr 0.22 -ff lp-0.1.dat\
     | f_dec -dec 10 -ff lp-0.05.dat\
     | f_scope --
```

This example illustrates how one can extract information around a very small bandwidth. One effectively zooms in on a small bandwidth region around a given carrier. Note also that decimation in stages (here we only used two stages) is very effective to isolate the interesting frequency range. Building a FIR filter with frequency resolution and pass- to stop-band transition region would require a very big number of filter coefficients.

The power of the ALE can be seen in the following example. First we run

```
f_sine -fr 0.123 | f_noise -am 10 | f_scope --
```

to generate a sine wave, add noise with 10 times the rms amplitude of the sine, and observe the resulting spectrum. Obviously, it would be impossible to see any peak at 0.123. In order to test the ALE we then execute

```
f_sine -fr 0.123 | f_noise -am 10\
                 | f_ale -nf 100 -mu 1e-9\
                 | f_scope --
```

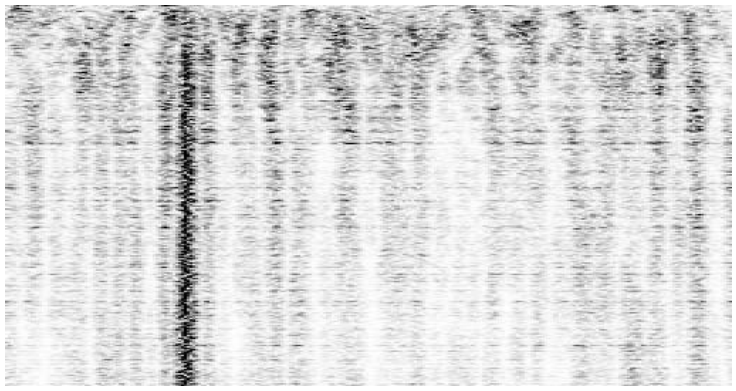

Figure 10: *The spectrogram while the adaptive line enhancer learns to find the sine in the noise. The horizontal axis corresponds to the frequency axis between 0 and 0.5 times the sampling frequency and the vertical axis corresponds to time progressing from top to bottom. The amplitude of the spectra is encoded in the gray value of the pixels and black denotes higher values. Note that the amplitude was set to auto-scale in this spectrogram.*

and wait for a few tens of spectra because the ALE algorithm is adaptive and needs some time to learn and adjust the filter weights. Then the peak will emerge from the noise as can be seen in Fig. 10.

The use of the histogram program `f_histogram` can be illustrated by feeding the output of the gaussian random number generator `f_noise` to `f_histogram` and on to `f_scope` in order to see how "gaussian" the random number distribution actually is. This feat is accomplished by the following command

```
f_noise -source | f_histogram -nbin 128 | f_scope -sink
```

Note that we do not use `f_scope` in spectrum analyzer mode but display the waveform instead. The number of bins was changed to 128 from the default value of 64.

Finally we comment on the use of `tf_mult` and `tf_add` which can be used to concatenate transfer functions both cascaded and in parallel. Thus the IIR filter made by cascading the filter defined in file `f1.iir` and file `f2.iir` can be calculated by the simple command

```
tf_mult f1.iir f2.iir > result.iir
```

where the output is redirected into the file `result.iir` which can be directly used by all other programs of the `f_tools` suite. The parallel concatenator `tf_add` works in much the same way.

Beyond the tools provided by the `f_tools` package one can employ standard Unix programs such as `awk` to manipulate the data stream. As an example we show how to use `awk` to extract a single column, here the third one, as expressed by `$3`, from a complex data file and then pipes a single-column data stream into `f_scope` which displays it.

```
cat complex.file | awk '{printf("%s\n",$3)}' | f_scope --
```

One can thus use one's favorite Unix tools or self-written programs to extend the data processing toolbox.

All the examples discussed so far only used internal data sources, namely the sine generator the wobble generator and the noise source. Since all programs communicate directly via the command line it is of course very easy to interface them to other real world sources. All that is needed to display some data stream, that is generated for example by the sound card of a PC, is a simple program that reads the sound-card's ADC and writes the samples to standard output formatted as a single float value per line. Then all software discussed so far can be interfaced and used to manipulate the sound stream. To generalize, one can write a program that generates data from simulations or measurements and pipe the result to standard output, as long as there is 512 data points per chunk the display will even be synchronized. The simulation in Ref. [8] are for example analyzed in this way.

4 Conclusions

We described a toolbox of small programs that perform digital signal processing tasks. The programs follow the standard Unix paradigm and read their data from standard input and write to standard output such that they are easily interfaced among themselves and to other programs that generate measured or simulated data. We describe the programs themselves and their use in a tutorial section. The tools presented can be used to design digital filters, use these filters in data streams, use mixers, both standard and image reject, and use adaptive line enhancers and display the results. Other features are the ease to design multi-rate filters to zoom in on a small frequency bandwidth.

Discussions with T. Lofnes, TSL about a wide range of signal processing issues are gratefully acknowledged.

References

[1] `http://www.mathworks.com`

[2] E. Ifeachor, B. Jervis, "Digital Signal Processing," Addison Wesley, 1993.

[3] Steven W. Smith, "The Scientist's and Engineer's Guide to Digital Signal Processing," Analog Devices Press.

[4] S. Haehnichen, "The Simple FFT package, Version 1.5," 1992, under GNU public license.

[5] T. Fisher, "The `mkfilter` Filter Generation Program," 1998, source available at `http://www-users.cs.york.ac.uk/~fisher/mkfilter/`

[6] D. Giamapolo, "Libsx v1.2, The Simple X library," available from Linux archives in the `X11/libs/clibs` subdirectory as `libsx-1.2.tar.gz`.

[7] `http://plasma-gate.weizmann.ac.il/Xmgr/`

[8] V. Ziemann, "Analysis of a method to measure the duo-decapole component of the LHC triplet magnets with a wobbling closed bump," TSL Note-2002-54, April 2002.

A Installation

The `f_tools` programs can be picked up from the author's web site at

`http://www3.tsl.uu.se/~ziemann/f_tools/`

as a tarred and gzipped file that unpacks into its own subdirectory named `f_tools`. You should inspect the Makefile in there whether its suits your needs. Building the software and installing it is done by the normal `make; make install` sequence. Initially there are binaries included in the program an you could just do `make install`. to get started. You only need to run `make` if you want to re-build from source.

Before building you should get the `mkfilter` package from the mkfilter web page at `http://www-users.cs.york.ac.uk/~fisher/mkfilter/` and unpack it into a subdirectory called `mkfilter` inside the `f_tools` subdirectory. This package is needed for the Remez-Parks-McClellan filter generation program.

Another package needed for the `f_scope` program is the `libsx` package that is easily obtained from a linux archive in the `X11/libs/clibs` subdirectory as `libsx-1.2.tar.gz`.

After these two packages are installed you can do `make; make install` and begin using the programs. My suggestion, however, is to try some of the tutorial examples first to get the hang of it...

B File formats

The programs of the `f_tools` suite operate on and generate a group of different files, of which are in plain ASCII format. Those are the files containing filter coefficients.

B.1 FIR filter files

These files contain the filter coefficients of the FIR filters in the format of one coefficient read in as a float value per line. The first coefficient is $a(0)$, followed by $a(1)$ and so forth. Comment lines start with a hash mark ("#"). The programs automatically determine the length of the filter and adjust their internal arrays accordingly. The length of the filter (the number of taps) is not explicitely written in the file.

B.2 IIR filter files

The IIR filter files are only marginally more complex than the FIR filter files, because now two polynomials must be specified, one of for the numerator (the a coefficients) and one for the denominator (the b coefficients). We require to specify the leading $b(0)$ term even though it is almost always unity, because we find it more convenient to specify the polynomials that constitute the transfer function explicitely and completely. We use an ampersand ("&") to separate the $a-$ and $b-$ coefficients. As in the FIR case the lengths of the polynomials are determined automatically and need not be specified

explicitely. An example of a file for a second order band-pass IIR filter is shown here

```
# f_make_iir: second order bandpass
# center=0.2 bw=0.01
       0.0293487
     0.000402489
      -0.0297512
&
               1
        -0.599493
           0.9409
```

Keep in mind that we follow the convention of Ref. [2] with respect to the sign of the b coefficients use the definition in eq. 1.

B.3 .hst files

The `f_scope` program has the capability to record the spectrogram display visible at the upper half of the display in Fig. 3. In order to save space the data are saved in binary form, as one byte per pixel, as this was the easiest to implement, because these are the grey-scale values which are calculated anyway for the spectrogram display. Since the display is `npts` (default is `npts=512`) pixels wide there are 512 byte per line stored consecutively in the file that is specified by the `-record file.hst` command line argument to the `f_scope` program. A simple program that extracts traces from the .hst files and converts them to Postscript or other output formats is available and was used to generate for example Fig. 10.