Filter Analysis and Design with FA

Quadrivium Incorporated
www.quadrivium.nl

May 30, 2009

Version 1.04

Contents

1 Introduction

FA is a program for design, analysis, and optimization of analog active integrated continuous-time filter networks. It is able to generate filter networks for standard transfer functions, such as Bessel, Butterworth, Chebyshev, and Cauer (elliptical). It can analyze filter networks to determine internal and external transfer functions in the frequency and time domain, the output noise spectrum, sensitivities for component-value perturbations, poles, zeros, and the dynamic range.

The main strength of FA is its ability to evaluate and optimize the dynamic range of filters, up to fundamental limits.

The companion program CI can generate SPICE-level netlists from filter specifications generated by FA.

2 Running FA

2.1 Batch Mode

If FA is started with a single command-line argument, it runs in batch mode. The argument specifies the input and output file names. For instance, if FA is started as

```
fa demo
```

FA looks for an input file named `demo.in`, writes to an output file `demo.out` or a plot file `demo.plot`.

If FA is invoked without command-line argument it reads standard input and writes to standard output. With input redirection it will still run in batch mode, as in

```
fa < demo.in > demo.out
```

2.2 Interactive Mode

If FA is started without command-line argument or input redirection, it runs in interactive mode. It maintains a command history which can be viewed with the command `history`. One can step through previously entered commands with the arrow keys. To exit FA, type the command `bye`.

3 Syntax

In the syntax descriptions in this manual, the following notation conventions apply.

Text in `teletype` is literal. Text in *italics*, enclosed between angle brackets, like <*word*> should be substituted by something else.

Anything enclosed in brackets (`[ ]`) is optional. If something is followed by an asterisk (`*`) it may be repeated any number of times, such that it occurs one or more times; a combination of brackets and an asterisk (`[ ]*`) indicates zero or more occurrences. A *bar* (|) means 'or', and a *colon* (:) indicates that the expression to the right of it defines the expression to its left.

All commands except the `bye` command should be terminated with a semicolon.

4 Variables and Expressions

A string of characters can can be used as a FA variable identifier if it has the following properties:

- it is not a FA keyword;
- it starts with a letter (A to Z or a to z);

+	binary addition
−	binary subtraction
*	binary multiplication
^	binary power
sin(x)	sine of x
cos(x)	cosine of x
tan(x)	tangent of x
asin(x)	arc sine of x
acos(x)	arc cosine of x
atan(x)	arc tangent of x
log(x)	natural logarithm of x
log10(x)	logarithm with base 10 of x
abs(x)	absolute value of x
sqrt(x)	square root of x
exp(x)	e^x
cn(x, k)	Jacobi cn of x with modulus k
sn(x, k)	Jacobi sn of x with modulus k
dn(x, k)	Jacobi dn of x with modulus k

Table 1: FA mathematical functions that can be used in expressions.

- it contains only letters and figures.

Specifically, note that `a`, `b`, `c`, `d`, `f`, `g`, `h` `k`, `m`, and `w`, as well as their uppercase equivalents, are all FA keywords, and hence cannot be used as variable identifiers.

The variables represent real numbers or expressions and can in turn be used in expressions, for instance:

```
y = 2 * x + sin(z);
output: y;
x = 1;
z = 2;
output: y;
```

produces the following output.

```
2.9093
```

Table 1 shows the available mathematical functions. The functions sn, cn, and dn are the Jacobian elliptic functions.

A number of FA variables have a predefined value, although it is possible to set them to any other value. Some of these have a special meaning for FA, which is explained in appropriate sections. The predefined FA variables are in Table 2.

5 Specify a transfer function

Filter networks generated by FA usually are cascade networks with approximately unity bandwidth. After generation, the networks can be transformed to ladder networks, and the bandwidth can be scaled to any positive value. However, most analyses and transformations are numerically better conditioned for a unity-bandwidth filter. It is therefore advisable to perform as many as possible operations before changing the bandwidth to the target value.

variable	value	section
bla	0	13.2
epsabs	$1 \cdot 10^{-6}$	14.1
epsrel	$1 \cdot 10^{-6}$	14.1
fmax	8	13.1
fr	0	7.2.4, 10.2.1, 10.3
itmax	0	14.3
noisefactor	1	10
pi	3.14159...	
si	1	10.3
so	1	10.3

Table 2: Predefined FA variables.

5.1 Butterworth

FA generates a Butterworth filter upon the following command.

`butterworth ( <`*order*`> ) ;`

where `<`*order*`>` is the order of the filter. This will result in a ladder filter with a −3dB bandwidth of 1rad/s. For instance:

```
butterworth(4);
```

results in a fourth-order Butterworth filter. To generate Butterworth cascade networks, specify a Chebyshev filter with zero ripple.

5.2 Chebyshev

The command

`chebyshev ( <`*order*`> , <`*ripple*`> ) ;`

with

`<`*order*`>` : integer; order,
`<`*ripple*`>` : real; ripple in dB

results in a Chebyshev cascade filter. Its bandwidth is 1rad/s. If the ripple is zero, the bandwidth is the frequency where the damping is −3dB. If the ripple is nonzero, the bandwidth is the highest frequency where the damping equals the ripple.

5.3 Inverse Chebyshev

To generate an inverse Chebyshev filter, use

`ic ( <`*order*`> , <`*damping*`> ) ;`

with

`<`*order*`>` : integer; order,
`<`*damping*`>` : real; stopband damping in dB

The stopband begins at 1rad/s.

5.4 Bessel

A Bessel cascade filter results from

`bessel ( <order> ) ;`

The denominator polynomial of this filter is an unscaled Bessel polynomial. This means that the −3dB bandwidth of the filter is somewhat larger than 1rad/s.

5.5 Elliptic

An elliptic or Cauer filter can be generated with

`cauer ( <order> , <ripple> , <stopband> ) ;`

<*order*>	:	integer; order,
<*ripple*>	:	real; passband ripple in dB,
<*stopband*>	:	real; normalized lowest stopband frequency.

The bandwidth of this filter, defined in the same way as for Chebyshev filters, is 1rad/s. The stopband begins at <*stopband*>, so this variable must be larger than 1.

5.6 Gaussian

The command

`gauss ( <order> ) ;`

results in a filter with a Gaussian response. This response approximates the ideal Gaussian transfer function, which is $e^{-(\omega/\omega_c)^2}$. The higher the order of the filter, the better the approximation. ω_c is the cut-off frequency, which is initially 1rad/s. This type of filter will approximately have a factor e, or 8.7dB, damping at 1rad/s. The −3dB point is at 0.59rad/s. A Gaussian response is similar to a Bessel response. It has a good phase linearity, but poor damping.

5.7 Gauss-Chebyshev

A Gauss-Chebyshev, or transitionally Gaussian filter combines the advantages of both the Gaussian and the Chebyshev response. For low frequencies, the response is approximately Gaussian, which results in a good phase linearity. Above a certain frequency, the transition frequency, the response becomes more like a Chebyshev response, which means good damping.

The command to generate such a filter is

`tgauss ( <order> , <ripple> , <damping> ) ;`

and results in a transitional Gaussian filter with specified order and ripple in dB, up to a specified damping in dB. For instance `tgauss(4, 0.01, 6);` results in a filter which is Gaussian to 6dB with 0.01dB ripple. To keep the group delay reasonably constant, the ripple should be small, most often between 0.001dB and 0.01dB.

5.8 Bessel-Chebyshev

Similar to a Gauss-Chebyshev filter is a Bessel-Chebyshev filter generated with

`bc ( <order> , <ripple> , <damping> ) ;`

or

`bessel_chebyshev ( <order> , <ripple> , <damping> ) ;`

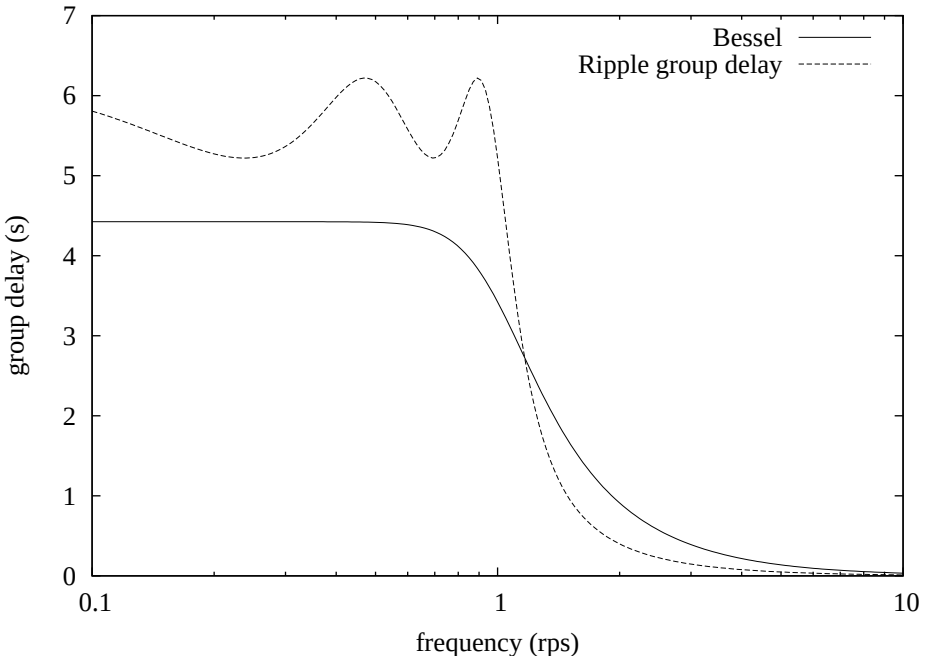

Figure 1: Group-delay response of a ripple-group-delay filter and a Bessel filter.

5.9 Ripple group delay

The command

`rgd(` <*order*> `,` <*ripple*> `) ;`

generates a ripple-group-delay filter of order <*order*>. The parameter <*ripple*> is the group-delay ripple in seconds. The resulting filter is a low-pass filter. Its bandwidth in which it approximates a constant group delay within the ripple specification is 1rad/s.

Like Bessel filters, ripple-group-delay filters approximate constant group delay in their pass bands. While the Bessel filter has zero group-delay ripple, the ripple-group-delay filter has a specified non-zero amount of ripple. Allowing this ripple has the advantage that the stop-band damping increases.

As an example, figure 1 shows the group-delay respones of a fifth-order ripple-group-delay filter with one second ripple generated with the command `rgd(5,1);` with the response of a fifth-order Bessel filter with the same amount of group-delay variation in the band between zero and 1rad/s. Figure 2 compares the damping characteristics of these filters. It is clearly visible that the ripple-group-delay filter has a larger amount of stop-band damping.

5.10 Custom pole pattern

The `cascade` command generates a cascade filter with a specified pole-zero pattern.

`cascade (` <*PoleZeroSpec*> [, <*PoleZeroSpec*>]`*` `) ;`

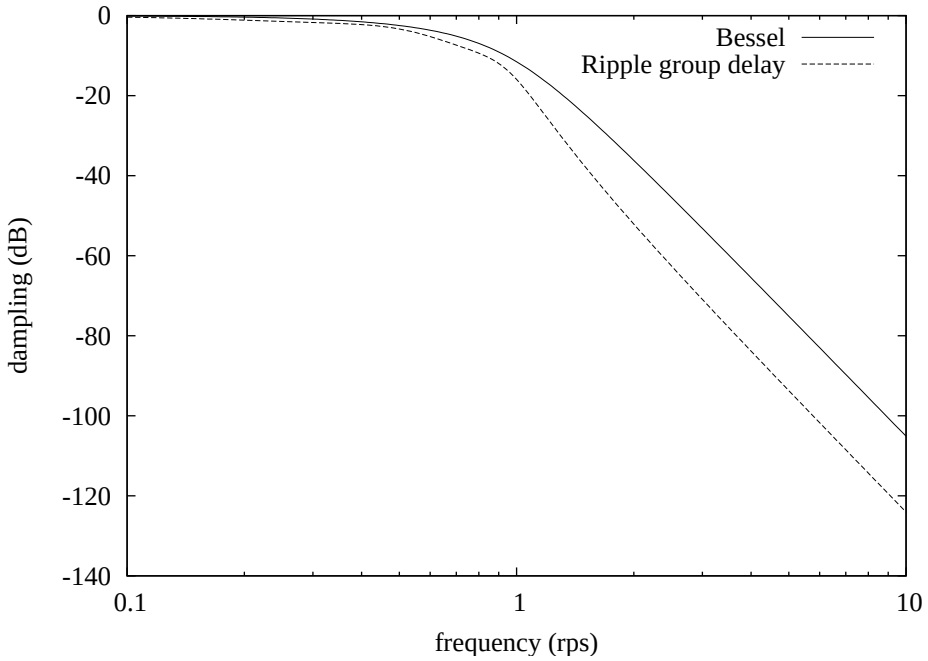

Figure 2: Damping of a ripple-group-delay filter and a Bessel filter.

```
<PoleZeroSpec>  :  <PoleSpec> [<ZeroSpec>]
<PoleSpec>      :  pole ( <real> )
                |  pole ( <real>, <imag> )
<ZeroSpec>      :  zero ( <real> )
                |  zero ( <real>, <imag> )
```

Each pole or pole/zero specification represents one first or second order substage in the resulting cascade filter. The stages appear in the same order as they are specified, starting at the input of the filter. A pole may appear without a zero, but not the other way round. The pole specification needs to precede the zero specification. A pole or zero with a zero imaginary part is interpreted as a single real root; otherwise a pair of complex roots is assumed. A non-real pole may be accompanied by a real zero, but if it is tried the other way round, only the real part of the zero is taken into consideration.

Each biquad has built-in damping to move the poles that this biquad realizes away from the imaginary axis. Each one of the two integrators that constitute a biquad may contribute to the total amount of damping in this biquad. By default, half of the total for each biquad goes to each integrator. This behavior can be modified with the variable `cascdist`. `cascdist` should be between 0 and 1. If it is 0, the first integrator will not contribute to the damping, and the second integrator will contribute everything. If it is 1, the first integrator will contribute all damping, and the second integrator nothing. If it is 0.5, half of the total goes to each integrator, which is the default.

As an example, the following input

```
cascade (
        pole (-0.425971),
        pole (-0.285118, 0.703363) zero (0, 1.55741),
        pole (-0.081731, 1.01253)  zero (0, 2.33188)
) ;
```

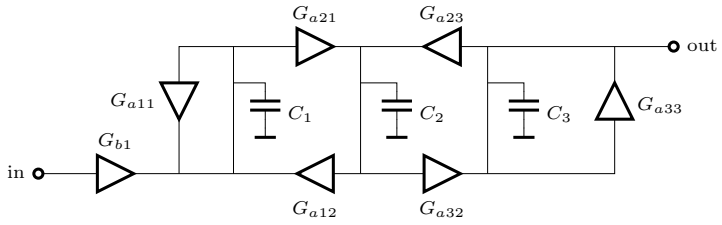

Figure 3: An active lowpass ladder filter.

results in the same filter as the following.

```
cauer(5, 0.5, 1.5);
```

6 Filter Networks

When the user specifies a transfer function to FA, FA will generate a filter network that realizes this transfer function. There is an infinite number of filter networks that realize the same transfer function. In stead of specifying the transfer function, the user may also specify the filter network. FA stores filter network as a state-space representation. The user may specify or request the state-space representation of the network.

FA can also transform a network into a different network which realizes the same transfer function. This is useful for network optimization.

6.1 Specification

6.1.1 State-Space Representation

Figure 3 shows an active realization of a third-order ladder filter. The filter consists of a network of transadmittance stages and capacitors. With $C_1 = C_2 = C_3 = 100\text{pF}$, $G_{b1} = 6.283 \cdot 10^{-7}\text{A/V}$, $G_{a11} = G_{a33} = -6.283 \cdot 10^{-7}\text{A/V}$, $G_{a21} = G_{a32} = -4.443 \cdot 10^{-7}\text{A/V}$ and $G_{a12} = G_{a23} = 4.443 \cdot 10^{-7}\text{A/V}$ the filter is a Butterworth type lowpass filter with a cutoff frequency of 1kHz. Figure 4 shows the results of a SPICE transfer-function analysis of the circuit.

The capacitors in the filter act as integrators, and the voltages on the capacitors are the output voltages of these integrators. These voltages form the *state variables* x_i of the filter. Let x_i designate the voltage over C_i. It is not difficult to derive the network equations in terms of the state variables:

$$\begin{aligned} sC_1x_1 &= G_{a11}x_1 + G_{a21}x_2 + G_{b1}V_{\text{in}} && (1)\\ sC_2x_2 &= G_{a21}x_1 + G_{a23}x_3 && (2)\\ sC_3x_3 &= G_{a32}x_2 + G_{a33}x_3 && (3)\\ V_{\text{out}} &= x_3, && (4) \end{aligned}$$

where V_{in} and V_{out} are the input and output voltages of the filter. Define the *state vector* $\boldsymbol{X}$ as

$$\boldsymbol{X} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

then the network equations become

$$\begin{aligned} s\boldsymbol{X} &= \boldsymbol{A}\boldsymbol{X} + \boldsymbol{B}V_{\text{in}} && (5)\\ V_{\text{out}} &= \boldsymbol{C}\boldsymbol{X} + \boldsymbol{D}V_{\text{in}} && (6) \end{aligned}$$

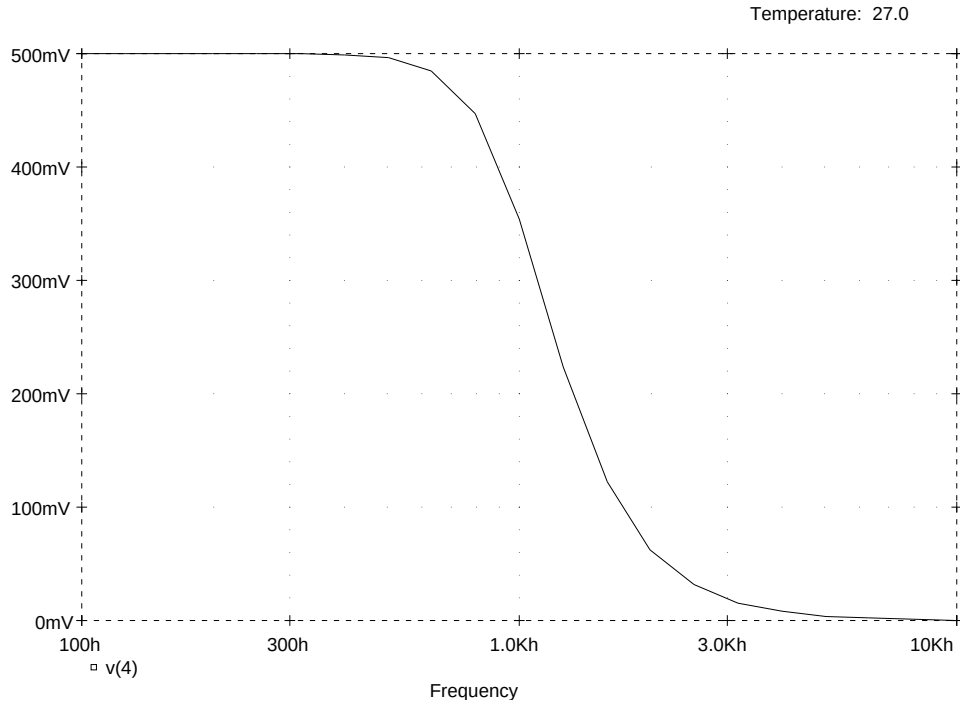

Figure 4: The transfer function of the filter of Figure 3.

with

$$\begin{aligned} \boldsymbol{A} &= \begin{pmatrix} \frac{G_{a11}}{C_1} & \frac{G_{a12}}{C_1} & 0 \\ \frac{G_{a21}}{C_2} & 0 & \frac{G_{a23}}{C_2} \\ 0 & \frac{G_{a32}}{C_3} & \frac{G_{a33}}{C_3} \end{pmatrix} \qquad \boldsymbol{B} = \begin{pmatrix} \frac{G_{b1}}{C_1} \\ 0 \\ 0 \end{pmatrix} \\ \boldsymbol{C} &= \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \qquad \boldsymbol{D} = 0. \end{aligned} \tag{7}$$

or numerically:

$$\begin{aligned} \boldsymbol{A} &= \begin{pmatrix} -6283 & 4443 & 0 \\ -4443 & 0 & 4443 \\ 0 & -4443 & -6283 \end{pmatrix}, \qquad \boldsymbol{B} = \begin{pmatrix} 6283 \\ 0 \\ 0 \end{pmatrix}, \\ \boldsymbol{C} &= \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \qquad \boldsymbol{D} = 0. \end{aligned} \tag{8}$$

Because these three matrices completely specify the structure of the filter, FA uses them for this purpose. A FA specification for this filter is

```
input :

order = 3;

A =
-6283   4443   0
-4443   0      4443
 0     -4443  -6283  ;
```

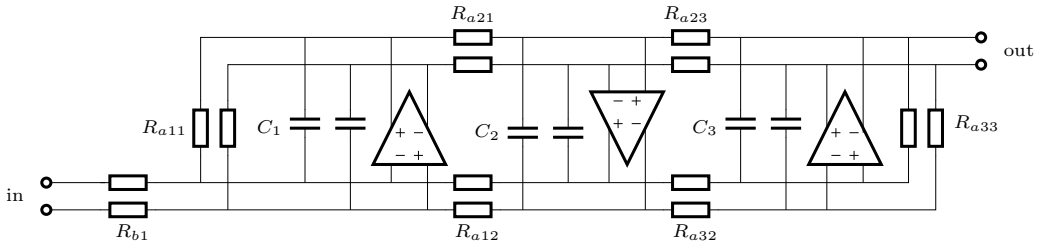

Figure 5: An opamp-R-C realization of the filter described by state matrices (8).

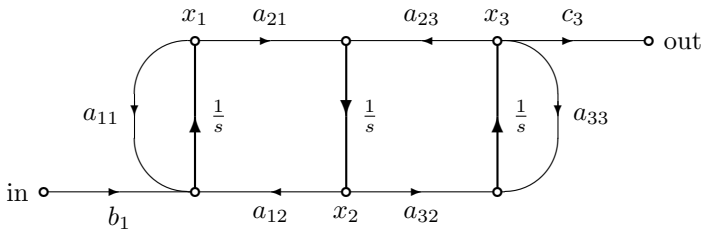

Figure 6: The signal flow graph for the filters of Figures 3 and 5.

```
B =
6283
0
0 ;

C =
0 0 1 ;
;
```

The order of the filter must be specified first. Then the state matrices appear, in the order $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$. The `input` command as well as the matrix specifications each require a semicolon as termination character. The specification of $\boldsymbol{D}$ is optional. If there is no specification for $\boldsymbol{D}$ (as in this example) it receives a value of zero.

There are several ways to realize the filter network structure as described by matrices (8). One of these realizations is the filter of Figure 3. An alternative realization as an opamp-R-C filter is in Figure 5. If the capacitor values are the same as in Figure 3, and the resistor values are the absolute values of the reciprocal values of the corresponding transadmittances, both examples will have the same state-space representation and transfer function.

Figure 6 shows a signal-flow diagram depicting the state-space model of the example filters of this section.

If a user specifies a filter transfer function, FA will determine a network that realizes this transfer function and store the network in the state-space format. FA will show the state matrices if the user enters the command `output: filter;`. For instance, the input specification

```
chebyshev(4,0.1);
```

```
output: filter;
```

results in the following.

```
(4, 4)
  -6.377299e-01   4.650002e-01   0.000000e+00   0.000000e+00
  -4.650002e-01  -6.377299e-01   0.000000e+00   0.000000e+00
  -0.000000e+00  -1.184767e+00  -2.641564e-01   1.122610e+00
   0.000000e+00   0.000000e+00  -1.122610e+00  -2.641564e-01

B =
(4, 1)
  -1.339622e+00
   0.000000e+00
  -0.000000e+00
   0.000000e+00

C =
(1, 4)
   0.000000e+00   0.000000e+00   0.000000e+00   1.000000e+00

D = 0.000000e+00
```

6.1.2 Capacitors

In most cases, the best way to specify the capacitances to FA is via the variable `ctot` that stands for the total capacitance value to be used for the whole filter. If FA runs into an analysis where it needs to know the capacitance values and `ctot` is not defined it issues an error message. By default the total capacitance `ctot` is divided equally over the capacitors of the filter. It is also possible to divide the capacitance optimally, see Section 10.4.2.

It is possible to specify the individual capacitances of the filter, for instance as follows.

```
caps = 1e-11 2e-11 3e-11 ;
```

This example applies to a third-order filter, and the three capacitor values are 10pF, 20pF and 30pF, respectively. Upon specification of the capacitor values in this way, FA will automatically adjust the value of `ctot`.

Setting the `evalc` will automatically redistribute the total capacitance, dependent on the status of the `eqc` and `optc` flags. If `eqc` is set, the capacitance will be redistributed equally. If `optc`, it will be distributed optimally. Specifying the capacitors individually will automatically clear the `evalc` flag.

6.1.3 Intrinsic and Extrinsic Integrators

For a correct understanding of internal transfer functions, the difference between *intrinsic and extrinsic integrators* must be clear.

The signal flow graph of the filter shown in Figure 6 contains three branches designated by $1/s$. These branches represent the *intrinsic* integrators. The *extrinsic* integrators each consist of one intrinsic integrator together with all the branches in the signal flow graph that are connected to its input. Consider for instance the leftmost integrator in the signal flow graph of Figure 6. Figure 7 shows both the intrinsic and the extrinsic integrator that correspond to this integrator. The intrinsic integrators have no direct representation in the filters of Figures 3 and 5. The extrinsic integrators, however, are represented

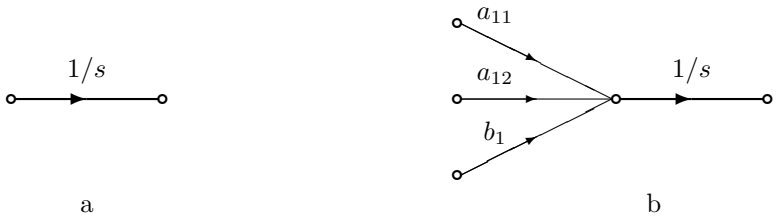

Figure 7: Examples of an intrinsic and an extrinsic integrator.

in these circuits. The output node of an intrinsic integrator is the output node of the corresponding extrinsic integrator and therefore corresponds to the output node of the appropriate physical integrator. The input nodes of the intrinsic integrators, on the other hand, do not exist in real filters. But these nodes are important in the filter design program.

6.2 Basic network transformations

Network transformations are useful to adjust the transfer function or to optimize the network properties. The command

```
transform:  lowpass( <bandwidth> );
```

will scale the bandwidth of the filter with a factor <*bandwidth*>. This is useful to modify the bandwidths of the filters that FA generates to realize one of the standard transfer functions of Section 5.

The command

```
transform:  bandpass( <ω0>, <ωc>);
```

transforms a lowpass filter into a bandpass filter. This will double the order of the filter. The bandwidth of the resulting filter is ω_c times the bandwidth of the original lowpass filter. The central frequency of the bandpass filter is ω_0.

With the following input

```
butterworth(3);
transform: bandpass( 2 * pi * 1e4, 2 * pi * 1e3);
output: filter;
freq=lin( 2 * pi * 0.8e4, 2 * pi * 1.2e4, 100);
plot: magn(H);
```

the initial lowpass equivalent filter is transformed to a bandpass filter with a central frequency of 10kHz and a bandwidth of 1kHz, the state-space representation of which is in the output file:

```
(6, 6)
  -6.283185e+03   6.283185e+04   6.283185e+03   0.000000e+00   0.000000e+00   0.000000e+00
  -6.283185e+04   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
  -3.141593e+03   0.000000e+00   0.000000e+00   6.283185e+04   3.141593e+03   0.000000e+00
   0.000000e+00   0.000000e+00  -6.283185e+04   0.000000e+00   0.000000e+00   0.000000e+00
   0.000000e+00   0.000000e+00  -6.283185e+03   0.000000e+00  -6.283185e+03   6.283185e+04
   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00  -6.283185e+04   0.000000e+00

B =
(6, 1)
   1.256637e+04
```

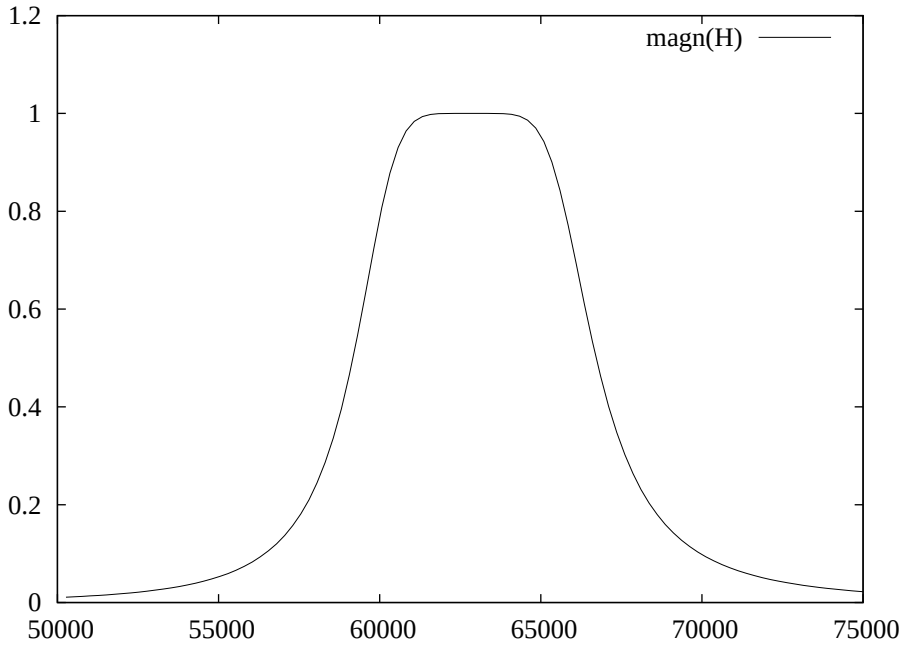

Figure 8: The magnitude response of a filter after transformation to a bandpass filter.

```
   0.000000e+00
   0.000000e+00
   0.000000e+00
   0.000000e+00
   0.000000e+00

C =
(1, 6)
   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00   1.000000e+00   0.000000e+00

D = 0.000000e+00
```

The magnitude plot is shown in Figure 8.

It is usually a good idea to perform network analysis and manipulation before lowpass and bandpass transformation. Operations are usually numerically better conditioned on a unity-bandwidth prototype than on a filter with a very large or small bandwidth. This is especially the case for bandpass filters because they have twice the order. Many operations remain valid after transformation. For instance, dynamic range and scaling conditions remain unchanged after lowpass transformation.

6.3 Transposition

`transform: transpose;` turns the filter into its transpose: $\boldsymbol{A}$ becomes $\boldsymbol{A}^T$, $\boldsymbol{B}$ becomes $\boldsymbol{C}^T$, $\boldsymbol{B}$ becomes $\boldsymbol{C}^T$, and $\boldsymbol{D}$ is unchanged. This in a way interchanges input and output of the filter. The transfer function is not changed by this action. For example: a ladder filter with damping at the input will become a ladder filter with damping at the output.

6.4 Ladder Filters

Originally, a ladder filter is a passive filter, containing inductors, capacitors, and damping resistors. The damping resistors can be at the input or the output or both. When transformed into active filters, these ladder filters have the attractive property that the state matrices are sparse, which helps to reduce the number of resistors (or transconductors) in the final realization. Another attractive property of these active ladder filters is that they can have very good dynamic-range properties.

FA can transform any filter into a ladder filter. The command

```
transform:  orthogonal;
```

will transform the current filter into a single-ended ladder filter with all damping at the input. The filter is orthogonal because its state-correlation matrix $\boldsymbol{K}$ (see Section 14.1) is equal to the Identity matrix, which makes the state variables form an orthogonal set.

A symmetrical ladder will result from the command:

```
transform:  ladder;
```

The `orthogonal` transformation is more robust than the `ladder` transformation. While the `orthogonal` transformation should always produce a result, the `ladder` transformation may fail because solutions do not always exist. For instance, generating a symmetric ladder of a Bessel filter of orders 7 and 8 does not work, but it is successful with order 9. In case of no convergence, FA will generate an error and leave the filter unchanged.

Even though `ladder` is less robust, it can do much more than `orthogonal`. It can generate ladders with many different distributions of the ladder damping. Whereas in passive ladder filters the damping is only at the input or output, in active ladder filters the damping can be in any integrator. The command

```
transform:  ladder;
```

is equivalent to

```
transform:  ladder(0.5);
```

The factor 0.5 says that half of the damping should be at the input and therefore the other half at the output. Likewise, `ladder(1)` will result in all damping at the input and no damping at the output. `ladder(0)` will place all damping at the output. `ladder(0.3, 0.2)` will result in a ladder with 30% of the damping in the first stage, 20% in the second stage, and the rest (50%) in the last stage.

7 Analysis in the Frequency Domain

The transfer characteristics of a filter is often specified as a function of frequency. Not only analysis of these transfer functions, but also other properties of the filter, such as the output noise spectrum, require analysis in the frequency domain.

7.1 Frequency Specification

For some analyses in the frequency domain it is necessary to specify a set of analysis frequencies. The command

```
freq = lin ( <start> , <stop> , <nrpoints> ) ;
```

specifies a linear distribution of frequencies, and

```
freq = log ( <start> , <stop> , <nrpoints> ) ;
```

a logarithmic distribution. The meaning of the symbols is as follows.

<*start*>	:	expression; start frequency
<*stop*>	:	expression; stop frequency
<*nrpoints*>	:	integer; number of frequency points evaluated for a linear sweep, or the number of frequency points evaluated per decade for a logarithmic sweep

By default, all frequencies are expressed in radians per second. In frequency plots, however, frequency may be expressed in Hertz, but only if the `herz` flag is set (see Section 9) as follows.

```
set:  hertz;
```

As an example, the frequency specification for the filter of the previous paragraph might be,

```
freq = log(628.3, 6.283e4, 10);
```

or almost equivalently:

```
freq = log(2 * pi * 100, 2 * pi * 1e4, 10);
```

or

```
set: hertz;
freq = log(100, 1e4, 10);
```

7.2 Plot Specification

Once the frequencies have been specified, a plot command starts analysis at these frequencies. This command is

```
plot:   [<FreqPlotSpec>]* ;
```

where <*FreqPlotSpec*> can be one of the following.

```
<transfer function>
magn(  <transfer function>  )
phase(  <transfer function>  )
delay(  <transfer function>  )
db(  <transfer function>  )
relsens
abssens
sens
onoise
dr
drfact
drscfact
dropfact
outputcurrents
<variable>
```

This chapter explains some of these specifications. Others follow later.

The `plot` command writes its results to a plot file. Another program, such as gnuplot, may read this file and produce an actual plot. Alternatively, the `xplot` command will open an X window and present the plot it it. This window allows zooming by dragging the mouse over the plot with the right mouse button down. Pressing the key "f" will fit the plot data in the window, the key "g" toggles the grid, and the keys "q" or "c" will close the window.

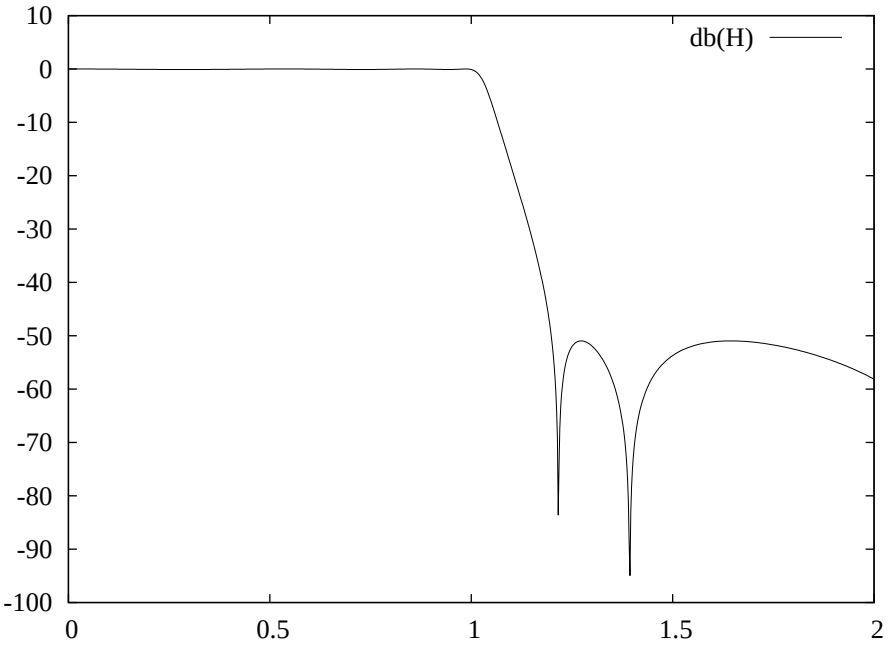

Figure 9: Transfer function of an elliptic filter.

7.2.1 Transfer Functions

FA uses three transfer functions, designated by the letters F, G and H. H is the transfer function of the filter from input to output. F is a vector of transfer functions from the input of the filter to the outputs of the integrators of the filter, and G is the vector of transfer functions from the inputs of the *intrinsic* integrators to the output of the filter.

Plotting H will result in one curve in the output, but plotting F or G will result in a number of curves that is equal to the order of the filter.

`magn`, `phase`, `delay`, and `db` signify the magnitude, the phase, the group delay, and the magnitude in decibel of the specified transfer function, respectively. If this keyword is omitted, the real and imaginary part are plotted.

As an example, let an input file `example.in` contain the following FA commands.

```
cauer(7, 0.1, 1.2);
freq = lin (0, 2, 1000);
plot: db(H);
```

Executing FA with this file as input results in a file called `example.plot`. Plotting the contents of this file with gnuplot results in the plot of Figure 9. Using the `xplot` command in stead of the `plot` command results in an interactive X window, shown in Figure 10.

7.2.2 Output Noise Spectrum

To determine the output noise spectrum of a filter, FA needs to know the following.

1. The state-space model of the filter network;
2. the capacitance values;

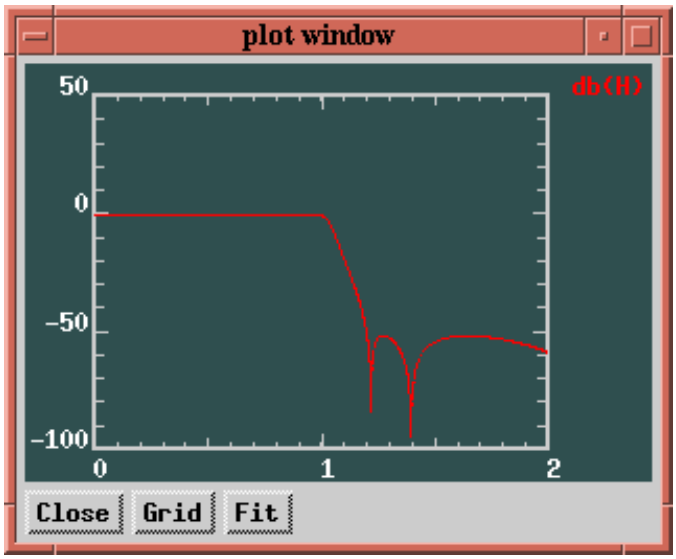

Figure 10: The transfer function of Figure 9 through `xplot`.

3. the noise factor.

An easy way to specify the capacitance values is to give a value to the variable `ctot`. The noise factor, stored in the variable `noisefactor` has a default value of 1.

An `onoise` plot specification results in a plot of the output noise spectrum of the filter in V^2/Hz. Note that this spectrum is double-sided. This means that also negative frequencies must be taken into account when integrating the noise spectrum to obtain the total output noise voltage. Most programs, such as SPICE, determine noise spectra single sided. The double-sided noise spectrum of a resistor R is $2kTR$, its single-sided spectrum is $4kTR$.

This analysis assumes a single-ended filter. If the filter is differential, the noise voltage is twice the value from the analysis, and the noise power is four times the result from this analysis. Because the differential signal voltages are also twice the single-ended signal voltages, the dynamic range remains unchanged.

7.2.3 Sensitivities

For the determination of the sensitivity of the transfer function of the filter for component variations without Monte Carlo analysis, three functions are provided. These can be specified by the keywords `relsens`, `abssens` and `sens`. They stand for the following quantities.

$$\text{relsens}^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}\left|\frac{a_{ij}}{H(\omega)}\cdot\frac{dH(\omega)}{da_{ij}}\right|^2 + \sum_{i=1}^{n}\left(\left|\frac{b_i}{H(\omega)}\cdot\frac{dH(\omega)}{db_i}\right|^2 + \left|\frac{c_i}{H(\omega)}\cdot\frac{dH(\omega)}{dc_i}\right|^2\right) \tag{9}$$

$$\text{sens}^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}\left|a_{ij}\cdot\frac{dH(\omega)}{da_{ij}}\right|^2 + \sum_{i=1}^{n}\left(\left|b_i\cdot\frac{dH(\omega)}{db_i}\right|^2 + \left|c_i\cdot\frac{dH(\omega)}{dc_i}\right|^2\right) \tag{10}$$

$$\text{abssens}^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}\left|\frac{dH(\omega)}{da_{ij}}\right|^2 + \sum_{i=1}^{n}\left(\left|\frac{dH(\omega)}{db_i}\right|^2 + \left|\frac{dH(\omega)}{dc_i}\right|^2\right) \tag{11}$$

The state-space parameters of the filter (the a_{ij}, b_i and c_i) are in practice realized by resistances or transconductances in combination with capacitors. Therefore the deviations of the components of the filter are reflected in its state space parameters. The particular function that is used to determine the deviations in the transfer function of the filter depends on the assumptions on the nature of the component deviations.

`relsens` is the ratio of the total relative standard deviation of the transfer function of the filter to the relative standard deviation of the a_{ij}, the b_i and the c_i. So `relsens` applies to a situation where the components suffer from the same *relative* deviation.

This is also the assumption underlying the use of `sens`. The difference is that the *absolute* deviation of the transfer function is computed (which should still be multiplied by the relative deviation of the component values).

If the components suffer from the same absolute deviation `abssens` should be used.

The functions `relsens2`, `sens2`, and `abssens2` are similar to `relsens`, `sens`, and `abssens`. The difference is that the former do not take the contributions due to the b_i and the c_i into account. This might be useful if (like in the example of section 6.1) there is only one b_i and only one c_i unequal to zero. In that case, variations in these branches will not alter the 'shape' of the filter, but only its gain.

7.2.4 Variables

Any variable can be plot if it resolves as a function of the frequency variable `fr`. That is,

```
pf = 1 / (y + x);
x = 10 * fr;
plot: pf;
```

is wrong, but

```
pf = 1 / (y + x);
x = 10 * fr;
y = 1;
plot: pf;
```

is okay.

7.3 Poles and Zeros

The poles and the zeros of the transfer function of the filter can be determined by adding the specifications `poles` and/or `zeros` to an `output` specification.

Example:

```
output:  zeros poles;
```

7.4 Polynomials

The keywords `denominator` or `numerator` in an `output` specification results in the denominator or numerator polynomial of the transfer function in the output.

8 Analysis in the Time Domain

FA can determine the response of a filter in the time domain on a unit step or a unit impulse at its input or at the inputs of the intrinsic integrators. The associated commands are much like the commands that pertain to frequency analyses.

default or reset to	set to	meaning if set	section
rps	hertz	Hertz in frequency plots	7.1
eqc	optc	optimal capacitor values	10.4.2
	evalc	redivide capacitance	6.1.2
nosom	som	enable printing second order modes	14.2
sol	num	determine $\boldsymbol{W}$ and $\boldsymbol{K}$ by numerical integration	14.1
mfr	sfr	single frequency	10.4.1
scale_inf	scale2	use ℓ_2 scaling	10.4.1

Table 3: Specifications for the setting or resetting flags in pairs that counteract each other. The default settings are in the leftmost column.

8.1 Time Specification

The time interval under consideration is specified as follows.

`time = lin ( <`*start*`>, <`*stop*`>, <`*nrpoints*`> ) ;`

The variables have the same meaning as before:

<*start*>	:	expression; start time
<*stop*>	:	expression; stop time
<*nrpoints*>	:	integer; number of time points evaluated.

8.2 Plot Specification

A plot in the time domain is specified as:

`plot:  [<`*TimePlotSpec*`>]* ;`

where <*TimePlotSpec*> can be:

`impulse( <`*transfer function*`> )`
`step( <`*transfer function*`> )`

<*transfer function*> can be either `F`, `G`, or `H`, as defined in Section 7.2.1.

In the example below, the step response of a bandpass filter is determined:

```
chebyshev(3, 3.0);
transform: bandpass(10, 1);
time= lin(0, 50, 1000);
plot: step(H);
```

The filter is generated from a third order Chebyshev lowpass equivalent filter, as explained in Section 6.2. The result is shown in Figure 11.

9 Flags

The `set` command is used to set or reset flags. The syntax is as follows.

`set: [<`*flag*`>]* ;`

Table 3 lists the possibilities for <*flag*>. The default settings are in the leftmost column. The possible settings are grouped in pairs that counteract each other.

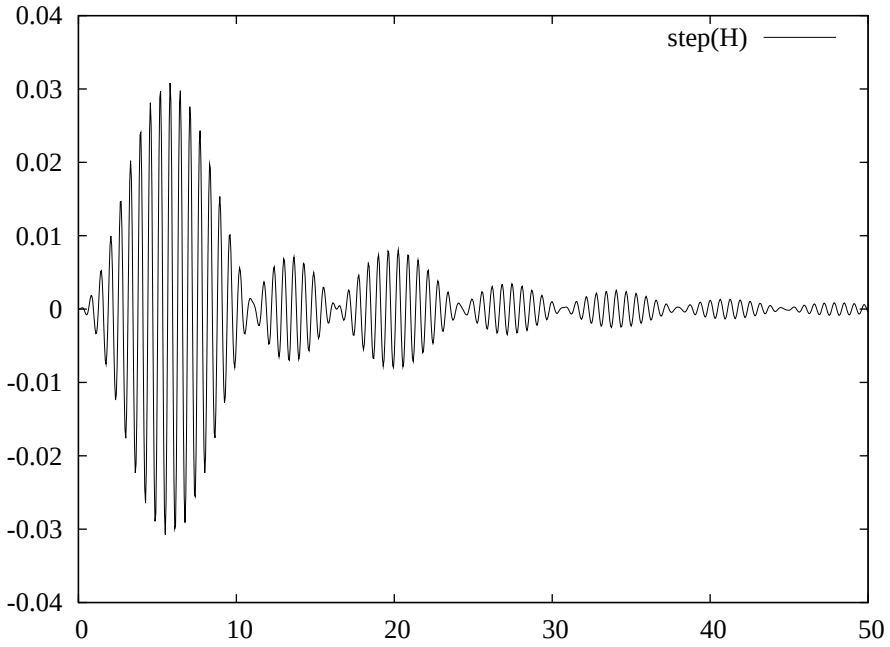

Figure 11: The step response of a bandpass filter.

10 Dynamic Range

The dynamic range of a filter is defined as the ratio between the maximal and minimal signal levels it can process. The maximal signal level is determined by the distortion in the filter; the minimum signal level is determined by the amount of noise produced by the filter. Generally, dynamic range is expressed in dB. The dynamic range is the most important design parameter of active filters, for two reasons.

Firstly, for many applications the dynamic range of active filters is not or barely enough. This makes careful optimization of the dynamic range very important.

Secondly, the chip area and the power consumption of integrated active filters are proportional to the dynamic range. Each 3dB obtained from optimization reduces chip area and power consumption by a factor two.

FA can evaluate and optimize the dynamic range of active filters.

10.1 Introduction

Generally an active filter consists of a network of active integrators. As active components usually suffer far more from non-linearities than passive components, usually the maximal signal level at the output of the filter is determined by the maximal signal level at the inputs or outputs of the integrators. As the outputs of the (extrinsic) integrators are directly connected to the inputs of other integrators, it does not matter whether the signal level is actually limited at the inputs or the outputs of the integrators; the results are the same. It is also assumed that the maximal signal level is the same for all the integrators.

The minimal signal levels in the filter are determined by the noise generation of the filter. In Section 10.2 it is shown that the noise of the filter can be linked to the capacitor values. So at least three things must be known to determine the dynamic range:

- the maximum signal levels at the outputs of the integrators,

- the total capacitance available to realize the filter,
- the noise factor of the active parts of the filter.

Consider the following example.

```
butterworth(3);
ctot = 30e-12;
umax = 1;
noisefactor = 1;
output: dr;
```

This specifies a third-order Butterworth filter with 30pF capacitance, 1V maximum root-mean-square signal level, and unity noise factor (which means that the active parts are assumed noiseless, the noise only stems from the passive components, that is, the resistors or the transadmittances). The FA output is then

```
Output noise level: 3.715373e-05 V.
Scaling is L-infinity.
Maximum output level: 6.314818e-01 V.
Dynamic range: 1.699646e+04 (84.61 dB).
```

The following section will discuss the definition of the dynamic range. After this, we will see how we can optimize the dynamic range.

10.2 Definition of Dynamic Range

As already mentioned, the dynamic range is the ratio of the maximal signal level and the noise level at the output of the filter.

10.2.1 Maximum Signal Level

The maximum signal level is dependent on the nature of the input signal of the filter. Several assumptions on the nature of the input signal lead to different values for the maximum signal level.

Sinusoid with unknown frequency A very popular assumption is that the input signal is a sinusoid with unknown frequency. In the time domain:

$$V_{\text{in}}(t) = V_{\text{A}} \sin(\omega t). \tag{12}$$

Let $v_{\text{in}}(s)$ be the Laplace transform of $V_{\text{in}}(t)$. If the transfer function in the Laplace domain from the input of the filter to the output of integrator i is f_i:

$$f_i(s) = \frac{x_i(s)}{v_{\text{in}}(s)}, \tag{13}$$

the amplitude of the signal at the output of this integrator will be

$$|x_i(j\omega)| = |f_i(j\omega)| V_{\text{A}} \tag{14}$$

Let the maximum output signal amplitude at the output of any integrator be V_{max}. The maximum filter input amplitude $V_{\text{max,in}}$ is then:

$$V_{\text{max,in}} = \frac{V_{\text{max}}}{\max_{i,\omega} |f_i(j\omega)|} \tag{15}$$

and the maximum filter output signal level is

$$V_{\text{max,out}} = V_{\text{max,in}} \max_{\omega}(|H(j\omega)|). \tag{16}$$

The mean-squared maximal output level is

$$\overline{V^2_{\text{max,out}}} = \frac{V^2_{\text{max,out}}}{2} \tag{17}$$

FA will use this assumption to calculate the maximum output signal level and the dynamic range if the `scale_inf` flag is set by

```
set:  scale_inf
```

This is the default.

Wide-band signal Another assumption on the input signal is that it is a wide-band signal with the spectral properties of white noise. Let the spectrum of the input signal be S_{in}. The mean squared output signal $\overline{V^2_{\text{s},i}}$ of integrator i is then

$$\overline{V^2_{\text{s},i}} = S_{\text{in}}|f_i|_2^2 \tag{18}$$

where

$$|f_i|_2^2 \stackrel{\text{def}}{=} \frac{1}{2\pi}\int_{-\infty}^{\infty} S_i(\omega)|f_i(j\omega)|^2 d\omega. \tag{19}$$

$S_i(\omega)$ is an input weighting function that is by default identical to 1. If the maximally allowed signal level at the outputs of the integrators is V_{max}, the maximum allowed value for S_{in} is

$$S_{\text{max,in}} = \frac{V^2_{\text{max}}}{\max_i |f_i|_2^2}. \tag{20}$$

If H is the transfer function of the filter, the maximum output signal level of the filter is

$$\overline{V^2_{\text{max,out}}} = S_{\text{max,in}}\frac{1}{2\pi}\int_{-\infty}^{\infty}|H(j\omega)|^2 d\omega = S_{\text{max,in}}|H|_2^2. \tag{21}$$

FA will use this assumption in determining maximal signal levels if the `scale2` flag is set by

```
set:  scale2
```

Sinusoid with fixed frequency In RF systems, the input signal of a filter may contain a blocker, which is a sinusoidal component at a fixed frequency which is by far the strongest component of the input signal. Let the frequency of this component be ω_{a}. The maximum input signal level of the filter is then

$$V_{\text{max,in}} = \frac{V_{\text{max}}}{\max_i |f_i(j\omega_{\text{a}})|}. \tag{22}$$

The maximum output signal level for sinusoidal signals is

$$\overline{V^2_{\text{max,out}}} = |H(j\omega_{\text{a}})|^2 \frac{V^2_{\text{max,in}}}{2} \tag{23}$$

If the blocker has been sufficiently suppressed, it does not play a role in determining the maximum output signal level. In that case one of the other methods described above should be used to determine the maximum output signal level.

To use this assumption on the input signal, specify

```
set: scale2 sfr;
```

and store the blocker frequency in the variable `fr`.

10.2.2 Noise

The filter output noise level is calculated from the input-referred noise spectra $S_{\mathrm{n},i}$ of the *intrinsic* integrators and the transfer functions g_i from the inputs of the intrinsic integrators to the output of the filter. If the noise spectra $S_{\mathrm{n},i}$ are white, the output noise level of the filter is

$$\overline{V_{\mathrm{n,o}}^2} = \sum_{i=1}^{n} S_{\mathrm{n},i} |g_i|_2^2. \tag{24}$$

The dynamic range is

$$DR = \frac{\overline{V_{\mathrm{max,out}}^2}}{\overline{V_{\mathrm{n,o}}^2}} \tag{25}$$

where $\overline{V_{\mathrm{max,out}}^2}$ follows either from (17) or from (21) or (23).

In the same way as $|f_i|_2^2$ is dependent on an input weighting function, $|g_i|_2^2$ is dependent on an output weighting function $S_o(\omega)$, which by default is identical to 1. Furthermore, one can prove [1] that

$$S_{\mathrm{n},i} = \frac{2kT\xi}{C_i} \left(|b_i| + \sum_{j=1}^{n} |a_{ij}| \right) \tag{26}$$

where C_i is the capacitance used for integrator i, and ξ is the noise factor.

FA will determine the dynamic range from the scaling flags, the total capacitance in the filter (the variable `ctot`), the maximal integrator output level (the variable `umax`) and the noise factor (the variable `noisefactor`) upon the command:

```
output:  dr;
```

If the variable `ctot` is not set, it will generate an error message.

10.3 Weighting Functions

By default the input signal source is assumed to have a 'white' spectrum, that is, its power spectral density is assumed to be independent of frequency. It is also assumed that the circuit which is connected to the output of the filter is equally sensitive to noise at all frequencies. If this does not represent the actual situation, assumptions can be modified using an *input and output weighting function*.

The input weighting function S_i is used to determine $|f_i|_2^2$ in (19). Similarly, the output weighting functions S_o is used to determine $|g_i|_2^2$, which is used in (24). In FA specifications, the input weighting function is called `si`, and the output weighting function is `so`. They must directly or indirectly be specified as a function of frequency, represented by the variable `fr`. If the input spectrum is expected to be a sinc function it could be specified like this,

```
si = (sin(fr * 1000) / (fr * 1000))^2;
```

or equivalently like this.

```
si = (sin(x) / x)^2;
x = fr * 1000;
```

For the change to take effect the flag `num` must be set. If this is done the optimizations and the dynamic range evaluations will use the weighting functions.

10.4 Optimization of Dynamic Range

The dynamic range of a filter is dependent on the filter transfer function and the realization of that transfer function. One method to improve the dynamic range of a filter is increasing the total capacitance that is used to realize the filter. If for instance the capacitance is enlarged by a factor two, the conductances of the filter are also to be enlarged by a factor two, if the transfer function of the filter is to remain unchanged by this action. This implies a reduction of the noise output of the filter by a factor two, and thus an increase of the dynamic range by 3dB.

The total capacitance that can be used to realize the filter is limited, usually by the chip area, sometimes also by the supply power. In the following we assume that the capacitance is limited to a total value of C_{tot}.

In that case the dynamic range is still dependent on the maximal signal levels in the filter. These levels are usually limited at the inputs or outputs of the integrators, and are dependent on the realization of the integrators. We assume that these maximal signal levels are known, and are (for all the integrators) equal to V_{max}.

There are three ways to further increase the dynamic range: optimization of the capacitance distribution, scaling and network transformation.

10.4.1 Scaling

Consider the following input file.

```
butterworth(3);
ctot = 3e-12;
umax = 1;
freq=lin(0,5,100);
output: dr;
plot: magn(f);
```

which specifies a third-order Butterworth filter. The output file is

```
Output noise level: 1.174904e-04 V.
Scaling is L-infinity.
Maximum output level: 6.314818e-01 V.
Dynamic range: 5.374751e+03 (74.61 dB).
```

and the result of the plot command is in Figure 12. This figure shows that the peak values of the f_1, f_3 and f_3 are around 1.6, 1.2, and 1.0 respectively. If the input signal of the filter is a sinusoid with unknown frequency, the output amplitude of integrator 1 is potentially 1.6 times the output amplitude of integrator 3. If the maximum integrator output signal RMS level is 1V, the maximum filter input signal level will be 1V/1.6 = 0.6V. Integrator 1 limits the maximum input signal level, while the other integrators would allow the input signal to be larger. This is not good for the dynamic range.

To equalize the integrator output signal levels, use

```
transform:  scale;
```

Adding this somewhere between the `butterworth(3);` command and the `output:  dr;` command results in the following output file

```
Output noise level: 1.370834e-04 V.
Scaling is L-infinity.
Maximum output level: 1.000000e+00 V.
Dynamic range: 7.294827e+03 (77.26 dB).
```

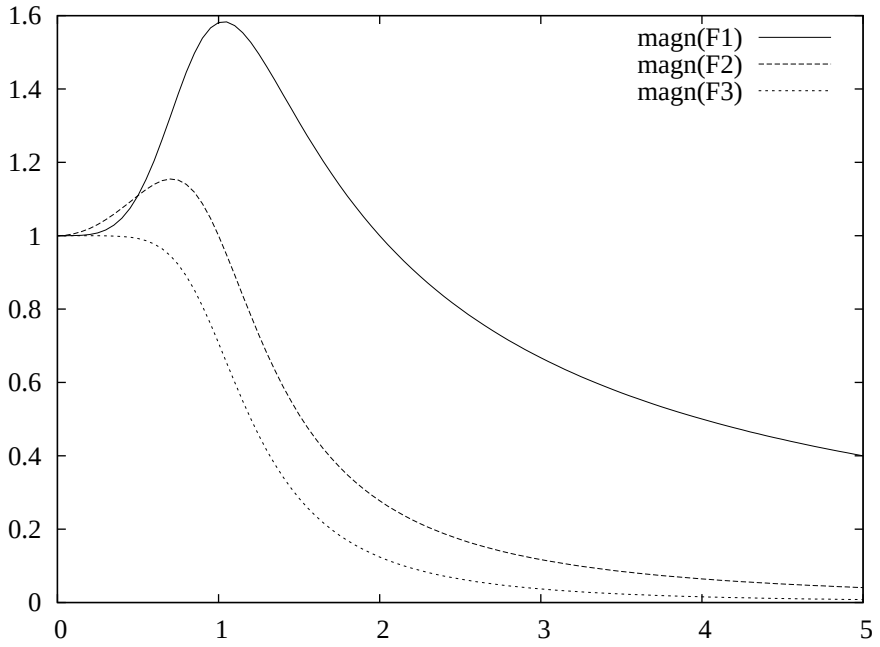

Figure 12: The internal transfer functions F from the input of the filter to the outputs of the integrators for the example filter.

and the internal transfer function plot of Figure 13. All internal transfer functions peak at a value of 1, and the dynamic range has increased by 2.65dB.

This form of scaling is appropriate if the input signal is a sinusoid with unknown frequency. It is known by the name ℓ_∞ scaling. This is the default type of scaling in FA.

In case of ℓ_∞ scaling, the program has to find the tops of the internal transfer functions. It may find a local peak. FA does an initial grid search to prevent this. It compresses the frequency range from zero to infinity into the interval $[0, 1]$ and divides this into a number of steps. It evaluates the transfer functions at each step to find an initial estimate for the global maximum. The default number of steps is 50. The number of steps can be changed with the command

```
scale_grid = <int>
```

If the input signal is not a sinusoid with unknown frequency, but a wide-band signal as explained in Section 10.2, equalizing the peaks of the internal transfer functions will not optimize the dynamic range. In this case the integrals of the squared magnitudes of the transfer functions must be equalized. This is a form of scaling known as ℓ_2 scaling. FA will use this type of scaling if the `scale2` flag is set, as in `set: scale2;`.

As an example, this input file

```
butterworth(3);
ctot = 3e-12;
umax = 1;
freq=lin(0,5,100);
set: scale2;
output: dr;
```

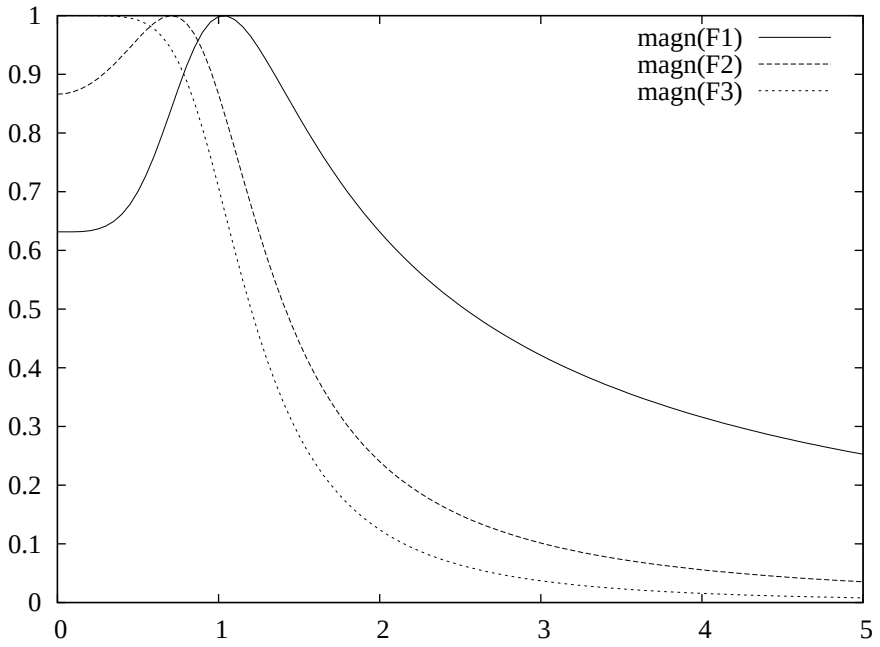

Figure 13: The internal transfer functions F after scaling.

```
transform: scale;
output: dr;
plot: magn(f);
```

produces the following output

```
Output noise level: 1.174904e-04 V.
Scaling is L2.
Maximum output level: 4.472136e-01 V.
Dynamic range: 3.806383e+03 (71.61 dB).


Output noise level: 1.559230e-04 V.
Scaling is L2.
Maximum output level: 1.000000e-00 V.
Dynamic range: 6.413422e+03 (76.14 dB).
```

and the plot of Figure 14. This plot shows that the scaling operation did not equalize the peaks of the transfer functions, but the integrals of the squared magnitudes of the transfer functions. The scaling operation improved the dynamic range by 4.53dB.

The spectrum of the input signal may not be flat. For example, let the input spectral density be given by the formula

$$S_{\mathrm{in}}(\omega) = \frac{1}{1 + (\omega/\omega_{\mathrm{i}})^2} \tag{27}$$

so that most of the spectral energy is present in the band from 0 to ω_{i}. In FA, the variable `si` represents this input spectrum, and `fr` represents frequency. To make the spectrum frequency dependent, specify

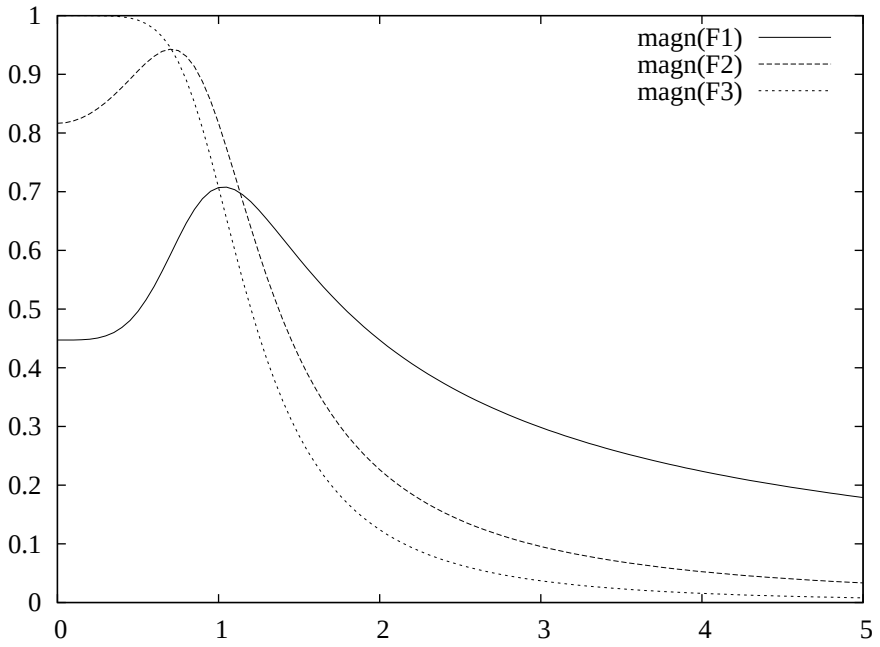

Figure 14: The internal transfer functions F after ℓ_2 scaling.

`si` as a function of `fr`. A specification of the spectrum (27) with $\omega_\mathrm{i} = 0.5$rad/s could be

```
si = 1/((fr/0.5)^2+1);
```

Additionally, FA needs to be instructed to determine internal signal levels by numeric integration and not by solving a set of equations, as follows.

```
set: num;
```

An example FA input file is

```
butterworth(3);
ctot = 3e-12;
umax = 1;
freq=lin(0,5,100);
set: num scale2;
si = 1/((fr/0.5)^2+1);
output: dr;
transform: scale;
output: dr;
plot: magn(f);
```

which gives the result

```
Output noise level: 1.174904e-04 V.
Scaling is L2.
Maximum output level: 7.608857e-01 V.
Dynamic range: 6.476151e+03 (76.23 dB).
```

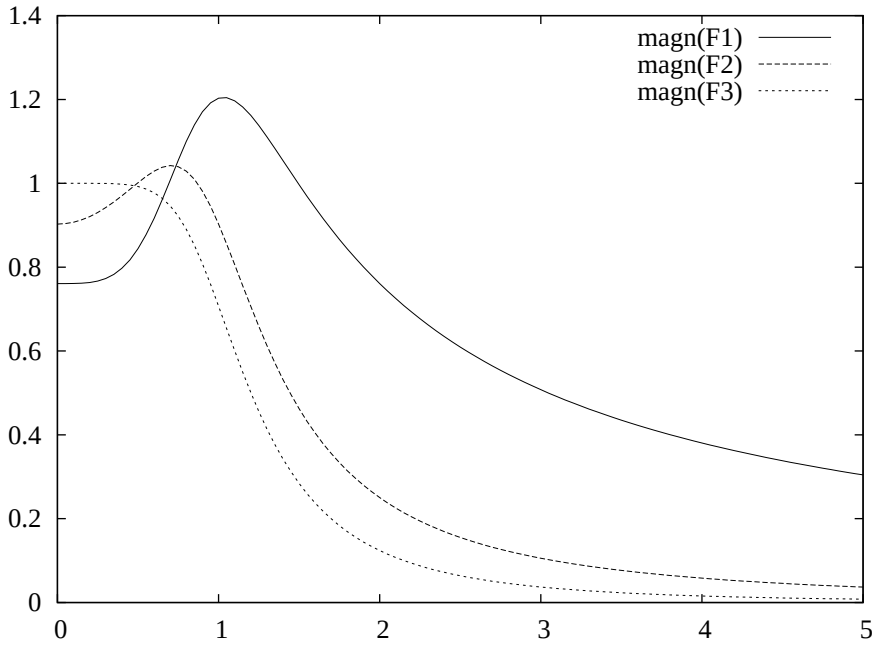

Figure 15: The internal transfer functions F after ℓ_2 scaling with a non-uniform input spectrum.

```
Output noise level: 1.288175e-04 V.
Scaling is L2.
Maximum output level: 1.000000e+00 V.
Dynamic range: 7.762919e+03 (77.80 dB).
```

and the internal transfer function plot of Figure 15.

Finally, consider the example where there is a strong blocker at 2rad/s. To specify this, set the `sfr` and the `scale2` flags, and assign the frequency of the blocker to the variable `fr`, as follows.

```
butterworth(3);
ctot = 3e-12;
umax = 1;
freq=lin(0,5,100);
set: sfr scale2;
fr = 2;
output: dr;
transform: scale;
output: dr;
plot: magn(f);
```

The FA output is

```
Output noise level: 1.174904e-04 V.
Scaling is L2.
```

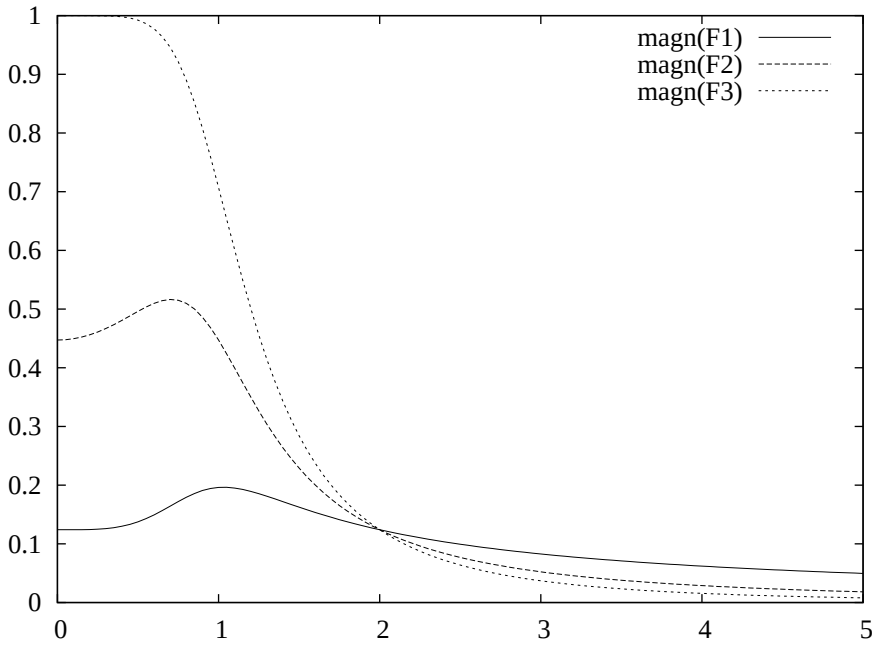

Figure 16: The internal transfer functions F after single-frequency scaling.

```
Maximum output level: 1.240347e-01 V.
Dynamic range: 1.055701e+03 (60.47 dB).


Output noise level: 3.486558e-04 V.
Scaling is L2.
Maximum output level: 1.000000e+00 V.
Dynamic range: 2.868159e+03 (69.15 dB).
```

and the internal transfer function plot of Figure 16.

10.4.2 Optimization of Capacitance Distribution

If the total capacitance value C_{tot} is known, the individual capacitance values can be determined such that the dynamic range is maximized. By default FA distributes the capacitance not optimally but equally. If the total capacitance is specified, the individual values are shown if a `caps` item is added to an `output` command. The capacitance is divided optimally if `optc` is set (see Section 9). For example, the input

```
butterworth(3);
ctot = 30e-12;
umax = 1;
noisefactor = 1;
transform: scale;
output: "equally distributed capacitances:" caps dr;
```

```
set: optc;
output: "optimally distributed capacitances:" caps dr;
```

results in

```
c1 = 1.000000e-11
c2 = 1.000000e-11
c3 = 1.000000e-11


Output noise level: 4.334959e-05 V.
Scaling is L-infinity.
Maximum output level: 1.000000e+00 V.
Dynamic range: 2.306827e+04 (87.26 dB).

optimally distributed capacitances:
c1 = 9.117466e-12
c2 = 9.957500e-12
c3 = 1.092503e-11


Capacitance is divided optimally.
Output noise level: 4.323185e-05 V.
Scaling is L-infinity.
Maximum output level: 1.000000e+00 V.
Dynamic range: 2.313110e+04 (87.28 dB).
```

We see that at least in this case the improvement in dynamic range by optimal redistribution of capacitance is very small.

10.4.3 Network Transformation

The operation of scaling a filter changes the values of the 'non-integrating' branches of the state-space graph (the filter network) but does not change the graph structure. If also the graph structure may be changed, there are more possibilities to optimize the dynamic range.

FA has several transformations to optimize the network of the filter. They are designated by the keywords `do`, `do2` and `do3` (where 'do' stands for 'dynamic-range optimization') and work quite differently.

`do` is an implementation of the dynamic-range optimization algorithm that was developed by Mullis and Roberts [2] and Hwang [3] for digital filters. This method assumes that the input-referred noise spectra for the integrators is *independent of the network realization and equal for all the integrators*. That this is not true in general is clear from equation (26). However, for high-Q bandpass filters, this is a reasonable assumption.

When optimizing a bandpass filter through the transformation `do`, this optimization should be done on the low-pass equivalent of the filter *before* lowpass to bandpass transformation, see Section 6.2.

Also, this optimization assumes the input signal to be a wide-band signal with the spectral properties of white noise, see Section 10.2.1.

`do2` takes everything into account and therefore returns with a filter that has a dynamic range that is higher than the dynamic range of the filter `do` yields. This is a more complex task than `do` performs. Therefore, `do2` is implemented as a numerical routine, whereas `do` is analytical and much faster.

In general, the filter structures returned by `do` and `do2` are not very practically realizable as a real circuit. Usually no or only few of the entries of $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ will be zero, so that the realization has

a very complex structure. The use of these routines is mainly that they show how much the dynamic range of a given realization differs from optimum.

`do3` is very similar to `do2`. However, it restricts the transformation for quicker computation, and to obtain a less dense filter realization. For this to work properly, the state matrix $\boldsymbol{A}$ before transformation should only have zero-valued elements below the first subdiagonal. This is the case for cascade and ladder filters. If before transformation $\boldsymbol{B}$ has only one non-zero element b_1 and $\boldsymbol{C}$ only has one nonzero element c_n (where n is the order of the filter), `do3` will keep it this way, which is beneficial for filters without finite transmission zeros.

These dynamic-range optimizers do not need the values of C_{tot}, V_{max} and the noise factor to work. These are only needed if the dynamic range is to be evaluated.

Here is an example input file.

```
butterworth(3);
ctot = 30e-12;
umax = 1;
noisefactor = 1;
transform: scale;
output: dr;
set: optc;
transform: do2;
output: "optimal filter:" filter caps dr;
```

The output is:

```
Output noise level: 4.334959e-05 V.
Scaling is L-infinity.
Maximum output level: 1.000000e+00 V.
Dynamic range: 2.306827e+04 (87.26 dB).

optimum at
  0.09  0.13 -0.17 -0.05 -0.05  0.17 DR change is  +1.180dB
optimal filter:
A =
(3, 3)
  -1.091324e+00   7.175562e-01   3.558719e-01
  -1.489248e-01  -3.718107e-06   5.197124e-01
  -5.287917e-01  -1.372476e+00  -9.086724e-01

B =
(3, 1)
   1.605863e+00
  -1.293019e-01
  -3.333185e-01

C =
(1, 3)
   4.972029e-02  -6.906501e-01   5.074617e-01

D = 0.000000e+00

c1 = 6.054107e-12
```

```
c2 = 1.435668e-11
c3 = 9.589212e-12


Capacitance is divided optimally.
Output noise level: 3.774110e-05 V.
Scaling is L-infinity.
Maximum output level: 1.000000e-00 V.
Dynamic range: 2.649631e+04 (88.46 dB).
```

So this optimization results in an improvement of the dynamic range of about 1.2dB above the dynamic range of the scaled filter. This also indicates that the maximum dynamic range attainable for the given transfer function and values of C_{tot} and $V_{\max}$ is 88.46dB.

10.4.4 Optimal Ladder Filters

The network transformations `do` and `do2` in Section 10.4.3 usually result in filter networks that have little practical value, because they are dense and rely on component matching for large levels of signal suppression in the stop band. The transformation `do3` is designed to result in filters that do not have these disadvantages. Still the filter networks generated by `do3` are denser than ladder filters.

FA can optimize the dynamic range of filter networks, while constraining the networks to be ladder networks. This will generally result in a network which is less dense than the network that `do3` would generate. However, the dynamic range from `do3` should be higher.

The command to generate optimal ladder filters is

```
transform:  optladder;
```

This optimization consists of two phases: a coarse search and a fine search. In the coarse search, FA steps through all possible combinations of ladder parameters, taking a limited number of steps for each parameter, to find an initial guess of the optimum. To set the number of steps, specify

```
transform:  optladder( grid = <steps> );
```

where <*steps*> is an integer that specifies the number of steps to take per parameter. The default value is 5.

It is possible to provide FA with an initial guess for the ladder parameters. FA will then skip the initial search. An example for a third-order filter may be

```
transform:  optladder(0.3, 0.2);
```

In this case, FA will start from a ladder filter with 30% of the total damping in integrator 1, 20% in integrator 2, and the rest (50%) in integrator 3. The damping parameters should each be between 0 and 1, and add up to a total of 1 or less. The number of parameters should be less than the order of the filter. Unspecified parameters will receive a zero value.

To only perform the coarse search and skip the fine search, specify

```
transform:  optladder2( <steps> );
```

10.5 Dynamic Range to Frequency Plots

If the filter input signal consists of a sine wave, the dynamic range of the filter depends on the frequency of this signal as follows.

$$DR\,(\omega) = \frac{V_{\max}^2 |H(j\omega)|^2}{\sum_{i=1}^{n} S_{\mathrm{n},i} |g_i|_2^2 \max_k |f_k(j\omega)|^2}. \tag{28}$$

To plot this function, use the command

```
plot:  dr;
```

Mainly to investigate the results of dynamic-range optimization with a weighting function or using a single frequency, the plot specifications `drfact`, `drscfact` and `dropfact` were incorporated. `drfact` specifies a plot of the normalized dynamic range as a function of frequency. `drscfact` specifies for each frequency the normalized dynamic range that can be reached if the filter is scaled *for that specific frequency*. `dropfact` specifies the normalized dynamic range that can be reached if the filter is optimized by `do` *for that specific frequency*. This means that the dynamic range factors as plotted by `drscfact` and `dropfact` can never be reached by one and the same filter at all frequencies, but usually at only one frequency.

The definitions of these factors are:

$$\text{drfact} = \frac{|H(j\omega)|^2}{\max_i |f_i(j\omega)|^2 \sum_j w_{jj}} \tag{29}$$

$$\text{drscfact} = \frac{|H(j\omega)|^2}{\sum_i |f_i(j\omega)|^2 w_{ii}} \tag{30}$$

$$\text{dropfact} = \frac{|H(j\omega)|^2}{\left(\sum_i \sqrt{\lambda_i}\right)^2} \tag{31}$$

where λ_i are the eigenvalues of the matrix $\boldsymbol{F}\boldsymbol{F}^*\boldsymbol{W}$ the factors of which are explained in Section 14.

These plotting specifications yield illustrative pictures for a filter that is optimized at one frequency.

11 Bias

The integrators in the filter have to deliver currents to charge capacitors and drive the resistors in the filter network. Based on the state-space description of the network, the values of the capacitors, and the maximum signal voltage levels, FA can determine the maximum current that each integrator has to deliver. These currents are a function of frequency, and they can be plot through the command

```
plot:  outputcurrents;
```

The maximum value of these currents over frequency determine the minimum output-stage bias currents of the integrators if they are biased in class A. FA determines these currents upon the command

```
output:  bias;
```

12 File manipulation

With `read` a specified file can be read as in

```
read "demo.in" ;
```

With `write` the current filter is written to a specified file in such a format that FA can read it again. So

```
write "temp.fil" ;
```

writes the current filter to the file `temp.fil`.

13 Output Format

13.1 Maximum Number of Numbers on a Line

For the printing of large matrices the rows of which do not fit on one line the maximum number of numbers to be printed on one line can be specified by `fmax`. So

`fmax = <`*value*`> ;`

sets `fmax` to `<`*value*`>`. Its default is eight.

13.2 Amount of Output

The `transform` commands are silent, but may be allowed to talk. The amount of talkativeness can be adjusted by the variable `bla`. The default value of `bla` is zero, which means no output from a `transform` command. As `bla` increases, the amount of output increases too. Values larger than one are only useful for debugging purposes.

14 Advanced Features

14.1 State Correlation Matrices

In the determination or optimization of the dynamic range of a filter the state-correlation matrices $\boldsymbol{K}$ and $\boldsymbol{W}$ are very important. $\boldsymbol{K}$ and $\boldsymbol{W}$ are defined as follows:

$$\boldsymbol{K} = \frac{1}{2\pi}\int_{-\infty}^{\infty} S_i \boldsymbol{F}\boldsymbol{F}^* d\omega \tag{32}$$

$$\boldsymbol{W} = \frac{1}{2\pi}\int_{-\infty}^{\infty} S_o \boldsymbol{G}^*\boldsymbol{G} d\omega \tag{33}$$

in which $\boldsymbol{F}$ and $\boldsymbol{G}$ are a column and row matrix, respectively, defined as:

$$\boldsymbol{F} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix} \tag{34}$$

$$\boldsymbol{G} = (g_1, g_2, \ldots g_n) \tag{35}$$

and S_i and S_o are the input and output weighting functions. The functions f_i and g_i are introduced in Section 10.2. From this section and the definitions given above it follows that

$$|f_i|_2^2 = k_{ii}, \tag{36}$$

$$|g_i|_2^2 = w_{ii}. \tag{37}$$

When this is compared to the dynamic range formulas in section 10.2 it is clear why $\boldsymbol{K}$ and $\boldsymbol{W}$ are so important for the evaluation of dynamic range. The optimization algorithm of Mullis and Roberts [2] and Hwang [3] use these matrices.

These matrices are automatically determined in FA whenever they are necessary. They can be written to the output by an `output` command, as in

```
output: K W;
```

If this command is encountered and the matrices are still not determined in another analysis they are implicitly determined.

If $S_i = 1$ and $S_o = 1$, and the filter has not more than one pole on the imaginary axis, $\boldsymbol{K}$ and $\boldsymbol{W}$ are the unique solutions of the equations

$$\boldsymbol{A}^T\boldsymbol{W} + \boldsymbol{W}\boldsymbol{A} = -\boldsymbol{C}^T\boldsymbol{C} \quad (38)$$

$$\boldsymbol{A}\boldsymbol{K} + \boldsymbol{K}\boldsymbol{A}^T = -\boldsymbol{B}\boldsymbol{B}^T \quad (39)$$

and it is by solving these equations that FA determines the matrices if `sol` is set, which is default. This is certainly the fastest way known, and in most cases the most reliable one. In this way the noise (and other) integrals are determined without numerical integration.

If S_i and/or S_o are not unity-valued constant functions $\boldsymbol{K}$ and $\boldsymbol{W}$ must be determined as a numerical integral, that is by directly evaluating (32) and (33). This can be done by setting the flag `num` (see section 9) before any (implicit) determination of the matrices. The effect of setting `num` is annihilated by setting `sol`, after which $\boldsymbol{W}$ and $\boldsymbol{K}$ are determined (again) by solving (38) and (39).

If the matrices are determined by numerical integration the relative or absolute accuracies by which the integrals are to be determined must be given. This is done via the keywords `epsrel` and `epsabs`, respectively, as follows.

`epsrel = <`*value*`> ; epsabs = <`*value*`> ;`

The default values for `epsrel` and `epsabs` are $1 \cdot 10^{-6}$. The numerical integration routine returns answers with an accuracy that corresponds to the largest of the two specified by `epsrel` and `epsabs`. `epsrel` is also used as a key to approximations that sometimes are necessary if singular matrices are encountered. If this is necessary FA always issues a warning.

Once the matrices are determined, they need not to be determined anymore. If by some `transform` command the filter is transformed, the $\boldsymbol{K}$ and $\boldsymbol{W}$ matrices are transformed accordingly. For that reason it is handy to determine these matrices *before* a lowpass-to-bandpass transformation, because after the transformation the order of the filter is twice the order it had, and the calculation will cost much more time.

14.2 Second-Order Modes

The second-order modes are the square roots of the (always positive) eigenvalues of the matrix product $\boldsymbol{K}\boldsymbol{W}$. These play an important role in optimizing dynamic range using `do` [1, 2, 3]. If the flag `som` is set, the second-order modes are printed if afterwards an optimization transformation by `do` is specified.

14.3 Modification of `do`

One of the last steps `do` does is finding an unitary transformation matrix that transforms the (by that time diagonalized) $\boldsymbol{K}$ and $\boldsymbol{W}$ matrices into a form that is not diagonal but has equal diagonal entries. This unitary transformation matrix is not unique and therefore the optimal filter that is found is not unique. Two different methods to find this matrix are implemented. Default is the method of Hwang [3] that is guaranteed to terminate in a finite number of steps. The method of Mullis and Roberts [2] is iterative and in some cases can need much computing time before convergence, but sometimes more elegant filter structures are found. This method is chosen if the variable `itmax` is chosen greater than zero. This variable then specifies the maximum number of iterations allowed for this method. If no convergence takes place within the specified number of iterations, an error message is issued.

15 Circuit Generation

In addition to FA, there is a program called CI that generates SPICE listings of active filters from a filter generated by FA. If in FA the command `writesp` is given, a file with the extension `.ci` is generated. So if the FA input file was called `demo.in`, the ci-file is called `demo.ci`. This file contains in order:

- a state-space description of the filter,
- a specification of the capacitors, and
- the frequency specification, if given by the user.

If, after this file has been generated and FA has ended, the command `ci demo` is issued, a file `demo.cir` will be generated. This file contains a SPICE listing of a filter circuit with idealized transconductances. The nature of the generated filter circuit can be modified if the `realization` command is added to the CI input file. This must be done after the capacitor specification and before the frequency specification, if present. The syntax of this command is

`realization: <`*realization*`> ;`

<*realization*> can be:

`transconductor`
`opamp`
`opamp bandpass ( <`ω_0`>, <`ω_c`> )`
`transconductor bandpass ( <`ω_0`>, <`ω_c`> )`
`simple transconductor bandpass ( <`ω_0`>, <`ω_c`> )`

These specifications speak for themselves. `transconductor` is default. With the `bandpass` command, a capacitor-coupled-biquad bandpass filter with the specified filter as lowpass equivalent is generated. This might also be done by transforming the filter to a bandpass filter in FA, and generating the CI file afterwards. But then a transconductor or resistor-coupled filter is generated by CI, which in the case of bandpass filters usually is badly realizable, due to parasitic phase shift in the resistors. The CI `bandpass` specification works similar to the FA `transform: bandpass()` command.

16 A design example

As an example, consider the design of a third-order Chebyshev lowpass filter with 0.18dB passband ripple, a 1MHz bandwidth and 80dB dynamic range. The following FA input file is a start.

```
chebyshev(3, 0.18);
transform: scale;
ctot = 3e-12;
noisefactor = 2;
umax = 1;
output: dr;
```

It is always a good idea to scale the filter for optimal dynamic range, so the input file contains a scale command. The total capacitance value is an initial guess. The noise factor is 2, or 3dB. Practice shows that this is a reasonable estimate. The maximum RMS single-ended integrator output signal level is 1V. FA generates the following output.

```
Output noise level: 2.389684e-04 V.
Scaling is L-infinity.
Maximum output level: 1.000000e-00 V.
Dynamic range: 4.184654e+03 (72.43 dB).
```

The dynamic range is 72.43dB, which is 7.57dB too little. To increase the dynamic range to 80dB, the total capacitance needs to be multiplied by $10^{7.57/10} = 5.71$. This makes the required total capacitance 17.14pF. Running FA again with this capacitance value

```
chebyshev(3, 0.18);
transform: scale;
ctot = 17.14e-12;
noisefactor = 2;
umax = 1;
output: dr;
```

gives the following result.

```
Output noise level: 9.997597e-05 V.
Scaling is L-infinity.
Maximum output level: 1.000000e-00 V.
Dynamic range: 1.000240e+04 (80.00 dB).
```

The dynamic range is exactly 80dB. The next step is specifying the bandwidth. This could have been done earlier, but the dynamic range is not dependent on the bandwidth, and the unity-bandwidth case is better numerically conditioned.

```
chebyshev(3, 0.18);
transform: scale;
ctot = 17.14e-12;
transform: lowpass( 2 * pi * 1e6 );
writesp;
```

The `writesp` command instructs FA to write the state-space representation of the filter to a `.ci` file. If the input file was called `design.in`. the `.ci` file is `design.ci`. The content of this file is as follows.

```
input:

order = 3;

A =
-5.261592e+06  0.000000e+00  0.000000e+00
-8.390388e+06 -2.630796e+06  7.376818e+06
 0.000000e+00 -6.828428e+06 -2.630796e+06 ;

B =
 5.261592e+06
 -0.000000e+00
 0.000000e+00 ;

C =
```

```
 0.000000e+00  0.000000e+00  1.000000e+00 ;

D = 0.000000e+00 ;
;
C1 = 5.713333e-12 ;
C2 = 5.713333e-12 ;
C3 = 5.713333e-12 ;
```

By default, the program CI will generate a transconductor-capacitor network. To generate a balanced opamp network, add the line

```
realization: opamp;
```

to this file. The file will then contain the following.

```
input:

order = 3;

A =
-5.261592e+06  0.000000e+00  0.000000e+00
-8.390388e+06 -2.630796e+06  7.376818e+06
 0.000000e+00 -6.828428e+06 -2.630796e+06 ;

B =
 5.261592e+06
 -0.000000e+00
 0.000000e+00 ;

C =
 0.000000e+00  0.000000e+00  1.000000e+00 ;

D = 0.000000e+00 ;
;
C1 = 5.713333e-12 ;
C2 = 5.713333e-12 ;
C3 = 5.713333e-12 ;

realization: opamp;
```

Assuming that the name of this file is `design.ci`, start CI with the following command line.

```
ci design
```

This will result in a file `design.cir` which is a SPICE netlist:

```
Circuit Description from FA
* Opamp Filter

.SUBCKT OPAMP 1 2 3 4
E1 3 0 1 2 1000
E2 0 4 1 2 1000
```

```
.ENDS OPAMP

* Integrator 1
C1 5 8 2.856667E-12
C2 6 7 2.856667E-12
X1 5 6 7 8 OPAMP
* Integrator 2
C3 9 12 2.856667E-12
C4 10 11 2.856667E-12
X2 9 10 11 12 OPAMP
* Integrator 3
C5 13 16 2.856667E-12
C6 14 15 2.856667E-12
X3 13 14 15 16 OPAMP

* Resistors
* From A
* A(0, 0)
R1 8 5 66530.9
R2 7 6 66530.9
* A(1, 0)
R3 8 9 41721.4
R4 7 10 41721.4
* A(1, 1)
R5 12 9 133062
R6 11 10 133062
* A(1, 2)
R7 15 9 47453.8
R8 16 10 47453.8
* A(2, 1)
R9 12 13 51264.9
R10 11 14 51264.9
* A(2, 2)
R11 16 13 133062
R12 15 14 133062

* From B
* B(0)
R13 5 1 66530.9
R14 6 2 66530.9

* From C
* Output nodes: 15 and 16

VIN 1 0 AC 1
EIN 2 0 1 0 -1


.END
```

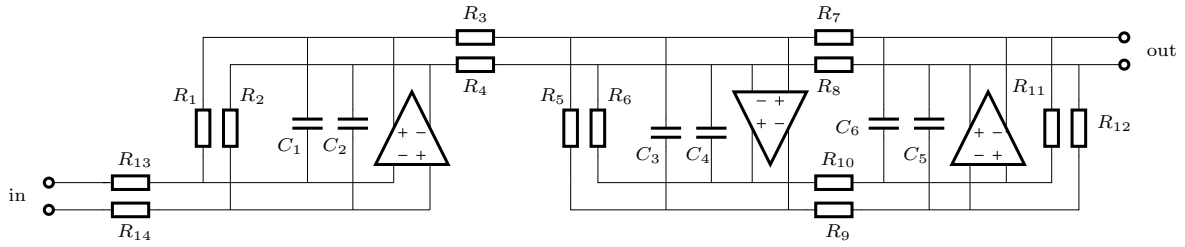

Figure 17: A schematic diagram of the example Chebyshev filter.

Figure 17 shows a schematic diagram of this filter. To determine the minimum output-stage class-A bias currents of the opamps, add a statement

```
output: bias;
```

to the FA input file. This results in

```
Maximal input signal level: 1
Minimum output bias currents:
Integrator 1:     7.79983e-05 A
Integrator 2:     6.73408e-05 A
Integrator 3:     6.61878e-05 A
Total bias current: 0.000211527 A.
```

References

[1] Gert Groenewold. The design of high dynamic range continuous-time integratable bandpass filters. *IEEE Transactions on Circuits and Systems*, 38(8):838–852, August 1991.

[2] C.T. Mullis and R.A. Roberts. Synthesis of minimum roundoff noise fixed point digital filters. *IEEE Transactions on Circuits and Systems*, CAS-23(9):551–562, September 1976.

[3] S.Y. Hwang. Minimum uncorrelated unit noise in state-space digital filtering. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-25(4):273–281, August 1977.