A Fully Automated Flowgraph Analysis Tool for Matlab

by Marko Neitola and Timo Rahkonen

Preface: Bug fixes and What's New

The SFG tool is now pushing a decade, and its time for an update in the manual as well.

The most difficult problem in the development of this tool is the symbolic math engine, which changed in MATLAB 2008b. I am not sure about recent evolution in the symbolic math instruction set, but the latest version works in R2014b.

For some older versions, the following fix seemed to be sufficient (add 4 find-commands):

lines 137-158:

```
innod(find(innod==sym(char(outnod(ind)))))=ym_ind;
outnod(find(outnod==sym(char(outnod(ind)))))=ym_ind;
innod(find(innod==sym(char(innod(ind)))))=ym_ind;
outnod(find(outnod==sym(char(innod(ind)))))=ym_ind;
```

The original submission included a parameter solver. This is now replaced by a function that finds a system of equations (soe) by comparing TF-object (H) and a symbolic transfer function (Hsym). Usage:

```
soe = find_soe(H, Hsym)
```

If soe is solvable, the built-in solve-command does the trick.

The rest of this manual is the original. It is still valid, except Chapter 5.

The latest version enables SFG system matrix generation without calculating the transfer functions. For most users, this is not important and the syntax is the same as before.

December 2014,
Dr. Marko Neitola
University of Oulu
Finland

A Fully Automated Flowgraph Analysis Tool for Matlab

M. Neitola T. Rahkonen

1. INTRODUCTION

This is a tool for finding a transfer function for a given system as fast and effortlessly as possible.
The user writes a nodelist presentation of the system in a text-file. The tool parses the system matrix and calculates the transfer function(s).

Signal flow graph [1,2] is simply a form of portraying a mathematically defined system as a group of equations. Flowgraph analysis can be helpful e.g. if a control system employ multiple feedback loops that make the analysis of the system tedious [3].

The goal was to construct an analysis tool for both continuous- and discrete-time systems. From a simple textual nodelist model one can easily generate one or several transfer functions between any nodes and apply the result to control system analysis. Such a versatile tool can be applied to filter design, delta-sigma modulator analysis or even reflection analysis s-parameter measurements in RF techniques and in optics. The tool is easy to customize to a given need and some design examples are presented in this paper.

This work is a “further development” of study presented in ICECS 2001 [3] and was published in [4].

```
Contents
    -m-files (*.m):
        flow_tf:            the flowgraph analysis function
        find_params:        parameter solver
        flowdemo:           a demo script
        test_all_nodelists: runs all *.flw files in the package

    -textual nodelist files (*.flw):
        zpar            terminated two-port model
        biquad:         biquad filter example
        iir:            digital IIR filter example
        integra:        a simple integrator
        efb:            error-feedback delta sigma
        MOD1:           1st order delta sigma
        MODL:           Lth order pure differentiation delta sigma
        CRFBSYM:        3rd order symbolic delta-sigma
        CRFBNUM:        3rd order numeric delta-sigma
        fiori           3rd order efb delta-sigma.
                      From paper by Fiori&Maloberti at ecctd2005
```

2. THE NODELIST FILE SYNTAX

The nodelist is written in **a plain text file**.
The main function (discussed in the next chapter) parses information from a textual nodelist file, where:

```
 * Comment begins with an asterisk.
* flow-line syntax is simple, three examples:
 innode outnode value
 1 2 3.147
 2 out h
 * Nodename can be text or number or both.

 * tf-command(s) compute the transfer function between any 2 nodes:
 .tf 1 out
 .tf noi out

 * pre-functions: initial values(.pre matlab_code):
 .pre syms z y
 .pre gainvalue=1; h=1/(z-1);
 * post-command (optional) defines the complex frequency-variable:
 .post z=tf('z',Ts)
 * tf(...) is matlab code, Ts is the sampling time
 * or for continuous-time systems
 .post s=tf('s')
 * or if you wish to re-evaluate symbolically
 .post syms w;z=exp(j*w);
```

Variable symbol(s) must be defined in pre-command (at least s or z if other parameters are numeric).

Post-command with tf(...) produces control system toolbox's LTI-structures(s) in H.tf.
Use the post command if there is just one complex frequency-variable (s or z).

3. THE MAIN FUNCTION

```
function H = flow_tf(flwfilename)

 Input - flwfilename
  The flowgraph file string e.g. H = flow_tf('my_fgfile.flw')

The suffix is not mandatory - I use just it to separate nodelist files from
other text files.

 OUTPUT - H
  A struct array including at least symbolic tf's (H.sym).
  If there are more than 1 ".tf " -lines, H.sym{1} is the
  first-defined transfer function in the nodelist file.
```

4. EXAMPLES

4.1. A two-input feedback system (A first order delta sigma loop)

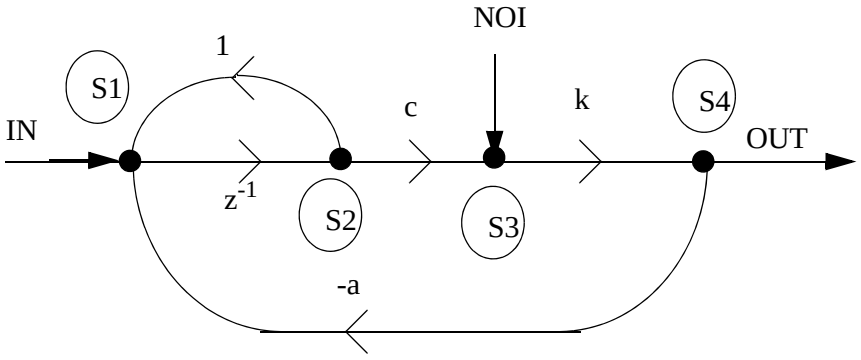

Figure 1: An example flowgraph of a 2-input, 1-output discrete-time system.

In the first example, the actual procedure for a system matrix generation is shown. This is automated in the flowgraph analysis tool. The system of equations is constructed by summing the <u>incoming signals</u> for each summation node S1, S2, S3 and S4:

$$\begin{cases} S1=IN+1*S2-a*S4 \\ S2=z^{-1}*S1 \\ S3=NOI+c*S2 \\ S4=k*S3 \end{cases} \quad \Leftrightarrow \quad \begin{bmatrix} 1 & -1 & 0 & a \\ -z^{-1} & 1 & 0 & 0 \\ 0 & -c & 1 & 0 \\ 0 & 0 & -k & 1 \end{bmatrix} \begin{bmatrix} S1 \\ S2 \\ S3 \\ S4 \end{bmatrix} = \begin{bmatrix} IN \\ 0 \\ NOI \\ 0 \end{bmatrix} \tag{1}$$

Now, for solving a transfer function between any two nodes, we need to calculate the inverse of the system matrix of Eq. (1). **<u>This is what the program does automaically.</u>**

Using the tool, the analysis for the example in Fig. 1 goes as follows: a Matlab program parses a user-defined textual nodelist (Fig. 2), identifies the nodes (S1 to S4) and constructs a system matrix. The ".pre" commands give the interpreter initial information about variables. Naturally all variables are needed to be defined as symbolic or numeric before the matrix inverse is calculated by MATLAB.

```
S1 S2 z^-1
S2 S1 -1
S2 S3 c
S3 S4 k
S4 S1 -a
.pre syms z
.pre c=1,a=1,k=1
.tf S1 S4
.tf S3 S4
.post z=tf('z',1)
```

Figure 2: A textual nodelist for the system shown in Fig. 1.

Since the input signal is not used in the transfer function analysis, one could name the nodes more self-explanatory e.g. node S1 could be named IN, S3 named NOI and S4 named OUT.

The tool attempts to find a simplified, minimal realization for the transfer function(s).

4.2. A terminated two-port

In this example, we have a circuit defined as a z-parameter two-port. The circuit is terminated by ZS (input) and ZL (output), Vin is the voltage of the source and v2 is the two-port's output voltage. Circuit diagram is shown in Fig. 3.

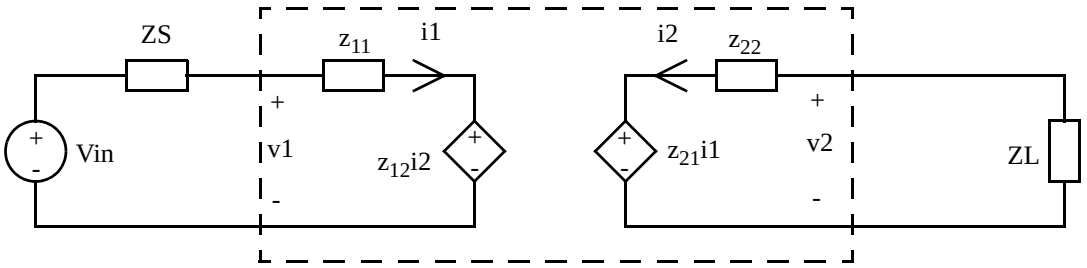

Figure 3: Terminated two-port z-parameter model.

A signal flow graph presentation is easy to construct by first writing the proper equations. When choosing which equations to write, remember that Vin is the source and v2 is the sink. In Eq. (2) we have the system of equations so that parameters in the right hand side are sources and in the left hand side, sinks. The corresponding nodelist definitions are also added to Eq. (2).

sinks

EQUATIONS

$$i1 = \frac{Vin - z_{12}i2}{ZS + z_{11}} = \frac{1}{ZS + z_{11}} \cdot Vin + \frac{-z_{12}}{ZS + z_{11}} \cdot i2$$

$$i2 = \frac{-z_{21}}{z_{22} + ZL} \cdot i1$$

$$v2 = -ZL \cdot i2$$

sources

sources sinks

NODELIST

```
vin  i1  1/(ZS+z11)
i2   i1  -z12/(ZS+z11)

i1   i2  -z21/(z22+ZL)

i2   v2  -ZL
```

(2)

For more information, see chapter 6 in [1]

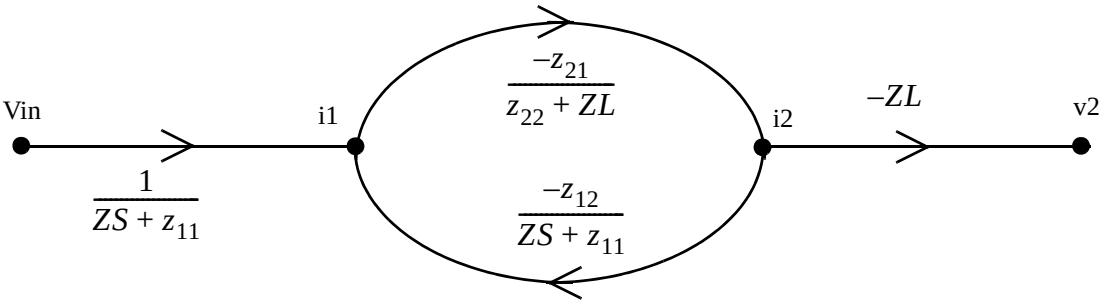

Figure 4: Flow graph of the terminated two-port z-parameter model.

```
* The beginning of the nodelist file is in Eq.(2).
* To complete the file, we also need to define the z-parameters and
* the impedances ZS and ZL
.pre syms z21 z11 z12 z22 ZS ZL
* Transfer function from vin to v2, i.e v2/Vin:
.tf vin v2
* as a bonus, lets find transfer function from
* 2-port's input voltage v1 to v2:
.post ZS=0 * this will create symbolic H.tf with ZS=0
```

4.3. An Analog biquad filter

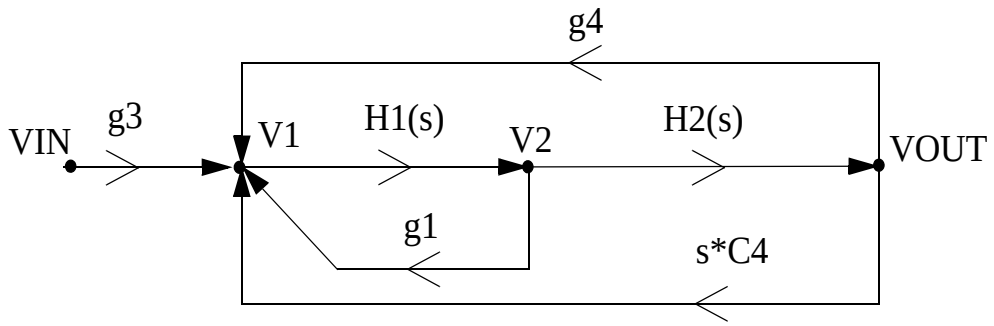

Figure 5: Flowgraph of an analog biquad filter.

Fig. 5 shows a signal flowgraph of a commonly used biquad filter that consists of two cascaded integrators and a flow g4 enclosing the feedback. The Q value for the 2nd order resonator can be tuned either by a resistive feedback around the first integrator H1 (flow g1) or capacitive feedback around both integrators H1 and H2 (flow s*C4). The transfer function solved is shown in Eq. (3). and one can clearly see that both feedback configurations can be used to control the Q value of the resonator.

```
>> H=flow_tf('biquad.flw')
* begin nodelist file:
* biquad
vin v1 g3
v1 v2 h1
v2 out h2
v2 v1 g1
vout v1 g4
vout v1 s*c4
.pre syms s g3 c1 a g4 g1 c4
.pre h1=-1/(s*c1);
.pre h2=a/s;
.tf vin vout
* end nodelist file
node     number
1        1
2        2
out      3
in       4
Elapsed time is 0.110000 seconds.
```

Figure 6: Analysis run for the biquad filter: The program prints the contents of the nodelist file, parser gives an index number to an arbitrary node name and finally, program outputs a transfer function H.

This 4-node description gives the transfer function from VIN to VOUT in 0.11 seconds and the result is

$$\frac{V_{OUT}(s)}{V_{IN}(s)} = \frac{-(g_3/C_1)\cdot a}{s^2 + \frac{g_1 + a\cdot C_4}{C_1}\cdot s + \frac{g_4\cdot a}{C_1}} \tag{3}$$

4.4. An IIR Filter

Fig. 7 shows the example network for IIR filter. Pay close attention for the naming of nodes and signals. The signal for input and output are named w and y, while the nodes attached to input and output are IN and OUT.

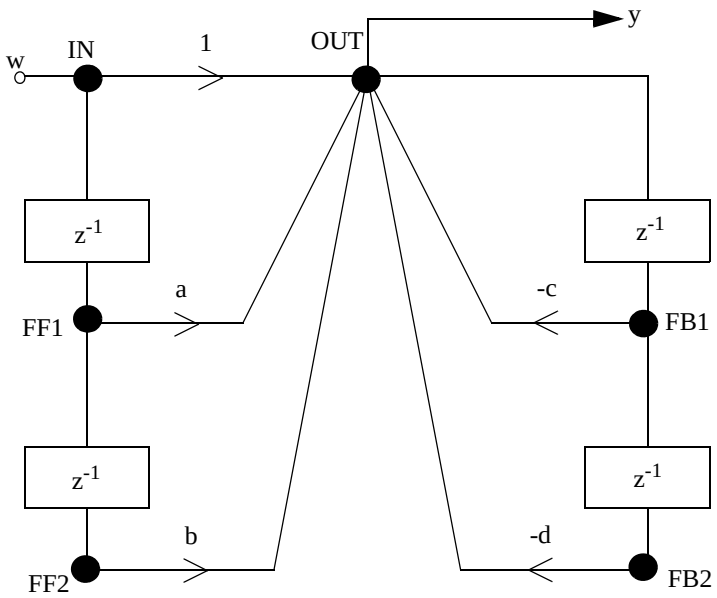

Figure 7: A digital realization of an IIR filter.

In transfer function analysis the signal type is irrelevant so for the sake of clarity, the nodes can be named according to their position in the topology. Fig. 8 shows the nodelist and will enlighten this naming philosophy.

```
>> H=flow_tf('iir.flw')
* begin nodelist file:
* iir
in out 1
in ff1 z^(-1)
ff1 ff2 z^(-1)
ff1 out a
ff2 out b
out fb1 z^(-1)
fb1 fb2 z^(-1)
fb1 out -c
fb2 out -d
.pre syms z a b c d
.tf in out
* end nodelist file

>> H.sym{1}
ans =
(a*z+b+z^2)/(z^2+c*z+d)
```

Figure 8: Flowgraph analysis run for the IIR filter example: nodelist file and the resulting transfer function.

4.5. A 3rd Order Multiple-Feedback Delta-Sigma Modulator

A delta sigma (DS) modulator is typically used as an analog-to-digital converter (ADC). It is an oversampling system consisting of a 2-input loop filter (for signal and feedback) and a quantizer. An Nth order DS consists of N integrators in the loop filter and some gain parameters. The output is quantized, D/A-converted and fed back for the input. The purpose of this type of ADC is to shape the quantizer error noise into higher frequencies, where the noise is cancelled in the preceding decimating filters.

In Fig. 9 we have a third order DS loop, where the quantizer is modelled as a gain and a noise input. This linear model is commonly used to analyze the stability of the DS modulator. The quantizer gain is not a pre-defined parameter: it depends on the topology and signal, and it can be estimated comparing the correlation between comparator input and output [5].

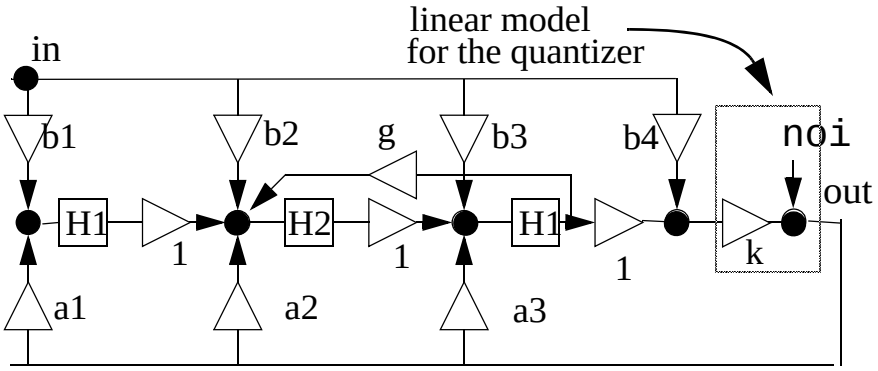

Figure 9: Linear mode of a 3rd order DS modulator. H1 and H2 are delayed and non-delayed integrators.

After optimizing the DS modulator parameters numerical a_i, b_i and g for oversampling ratio of 16, we can easily write the 6-node nodelist file. The analysis took about 0.5 seconds and outputs were in both symbolic and in a TF-object format. From latter, we can directly plot the responses for both transfer functions: Signal transfer function (STF) is from input to output and noise transfer function (NTF) is from noise input to output. As seen in Fig. 10, the STF is nicely flat and the NTF shapes most of the quantizer noise out of the signal band.

It is worth mentioning, that with a simple change in the nodelist file, the user can convert integrators into resonators and hence transform the low-pass-type modulator into a bandpass-type.

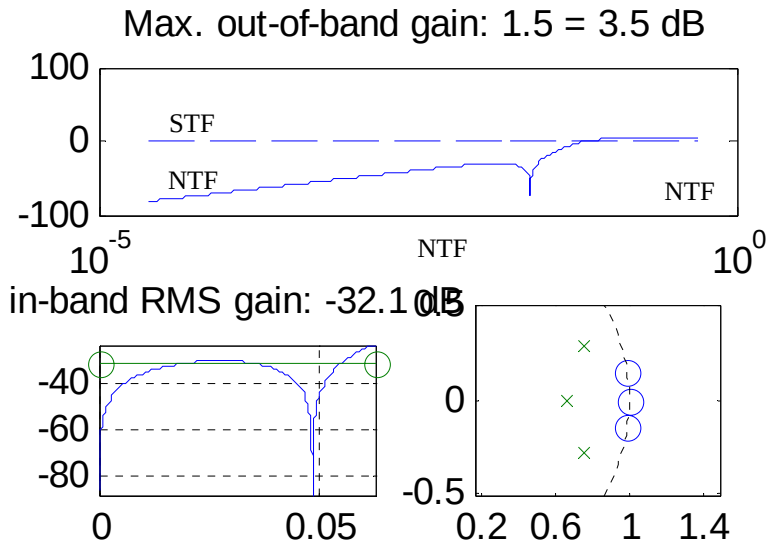

Figure 10: NTF and STF of the example modulator.

5. SOLVING NUMERIC PARAMETERS

If the user has designed one or more transfer functions, for the given system e.g. the noise and signal transfer functions (NTF and STF) for a delta-sigma modulator, he/she can produce a symbolic NTF and STF for the topology (parameters are symbolic). Finding the parameters is only a matter of making a match between the numeric and symbolic transfer functions. Writing a function that solves these parameters in matlab is not difficult, for it is only a matter of solving the equivalence in a system of equations.

In Fig. 11 this comparison is run in a custom program, whose inputs are NTF and STF and the symbolic equivalences H.sym{1,2}. We desire a flat STF so here it is 1. Only limitation for this procedure is that the system of equations must be numerically solvable.

R. Schreier's DS toolbox [6] is a very popular Delta-Sigma modulator design and simulation toolbox. One of the most important functions of it are synthesizing the modulator parameters for a given specifications. The toolbox has a couple of topology-options for the parameter realization, which pretty much cover the typical design needs i.e multiple-feedback topology (our example), feed-forward summation with one feedback signal. Both can have complex zeros and can also have delayless integrators.

If we have a custom topology that differs from these default topologies even slightly, the parameter realization can't be done using the DS toolbox. Here, the flowgraph analysis tool with the parameter solver can be a big help.

```
>>out=find_params([NTF 1],[H.sym{2} H.sym{1}])
Success!!!
out =
    a1: 0.04561498000000
    a2: 0.22703752000000
    a3: 0.56070378000000
    b1: 0.04561498000000
    b2: 0.25012482000000
    b3: 0.56070378000000
    b4: 1
     g: 0.02308730000000
```

Figure 11: A custom parameter solver usage in Matlab.

The parameter solver is also included in the package.

6. REFERENCES

[1] H. Ruston, J. Bordogna: Electric Networks: Functions, Filters and Analysis, McGraw-Hill 1966

[2] J. Vlach, K. Singal. Computer Methods for Circuit analysis and Design. Van Nostrand Reinhold Company Inc. 1983.

[3] T. Rahkonen, M. Neitola: Automated Flowgraph Analysis Using Matlab and Maple. 2001 IEEE international Conference on Electronics, Circuits, and Systems, Malta, Sept. 2.-5. 2001, pp. 605-608.

[4] Neitola, Marko & Rahkonen, Timo: A fully automated flowgraph analysis tool for matlab, Proceedings of the 17th European Conference on Circuits Theory and Design, ECCTD'05, Cork, Ireland,

[5] S. R. Norsworthy, R. Schreier and G. C. Temes: Delta-Sigma Data Converters: Theory, Design and Simulation, IEEE Press Marketing 1997.

[6] The MathWorks, www.mathworks.com User Community