THE DELTA-SIGMA TOOLBOX Version 7.3

Getting Started

The Delta-Sigma toolbox requires the `Signal Processing Toolbox` and the `Control Systems Toolbox`. Certain functions (`clans` and `designLCBP`) also require the `Optimization Toolbox`.

To obtain a copy of the Delta-Sigma toolbox, go to the MathWorks web site (`http://www.mathworks.`[illegible]`leexchange`) and search for `delsig`. To improve simulation speed, c[illegible]`M.c` file by typing `mex simulateDSM.c` at the Matlab prompt. Do the sam[illegible]`i2mif.c`. (for importing figures into FrameMaker) and `flattenStruct`[illegible]4).

For mo[illegible]f selected functions and more examples, see Chapters 8 and 9 of

[1] R. [illegible]es, *Understanding Delta-Sigma Data Converters*, John Wiley & So[illegible]

Toolbox Conventions

Frequencies are normalized; $f = 1$ corresponds to f_s.

Default values for function arguments are shown following an equals sign (=) in the parameter list. To use the default value for an argument omit the argument if it is at the end of the list, otherwise use `NaN` (not-a-number) or `[]` (the empty matrix) as a place-holder.

A matrix is used to describe the loop filter of a general single-quantizer delta-sigma modulator. See “MODULATOR MODEL DETAILS” on page 33 for a description of this “ABCD” matrix.

Demonstrations and Examples

`dsdemo1` Demonstration of the `synthesizeNTF` function. Noise transfer function synthesis for a 5th-order lowpass modulator, both with and without optimized zeros, plus an 8th-order bandpass modulator with optimized zeros.

`dsdemo2` Demonstration of the `simulateDSM`, `predictSNR` and `simulateSNR` functions: time-domain simulation, SNR prediction using the describing function method of Ardalan and Paulos, spectral analysis and signal-to-noise ratio for lowpass, bandpass, and multi-bit lowpass examples.

`dsdemo3` Demonstration of the `realizeNTF`, `stuffABCD`, `scaleABCD` and `mapABCD` functions: coefficient calculation and dynamic range scaling.

`dsdemo4` Audio demonstration of MOD1 and MOD2 with sinc^n decimation.

`dsdemo5` Demonstration of the `simulateESL` function: simulation of the element selection logic of a mismatch-shaping DAC.

`dsdemo6` Demonstration of the `designHBF` function. Hardware-efficient halfband filter design and simulation.

`dsdemo7` Demonstration of the `findPIS` function: positively-invariant set computation.

`dsdemo8` Demonstration of the `designLCBP` function: continuous-time bandpass modulator design. (This function requires the `Optimization Toolbox`.)

`dsexample1` Example discrete-time lowpass modulator.

`dsexample2` Example discrete-time bandpass modulator.

`dsexample3` Example continuous-time lowpass modulator.

`dsexample4` Example discrete-time quadrature bandpass modulator.

Key Functions

`ntf = synthesizeNTF(order=3,R=64,opt=0,H_inf=1.5,f0=0)` page 5
`ntf = clans(order=4,R=64,Q=5,rmax=0.95,opt=0)` page 6
`ntf = synthesizeChebyshevNTF(order=3,R=64,opt=0,H_inf=1.5,f0=0)` page 7

Synthesize a noise transfer function.

`[v,xn,xmax,y] = simulateDSM(u,ABCD` or `ntf,nlev=2,x0=0)` page 8

Sim [illegible] ulator with a given input.

`[snr,a` [illegible] `f,R,amp=...,f0=0,nlev=2,f=1/(4*R),k=13)` page 9

Det [illegible] power curve by simulation.

`[a,g,b` [illegible] `,form='CRFB',stf=1)` page 10

Con [illegible] tion into coefficients for a specific structure.

`ABCD =` [illegible] `orm='CRFB')` page 11

Cal [illegible] iven the parameters of a specified modulator topology.

`[a,g,b` [illegible] `orm='CRFB')` page 11

Cal [illegible] a specified modulator topology given the ABCD matrix.

`[ABCDs, umax] = scaleABCD(ABCD,nlev=2,f=0,xlim=1,ymax=nlev+2))` page 12

Perform dynamic range scaling on a delta-sigma modulator described by ABCD.

`[ntf,stf] = calculateTF(ABCD,k=1)` page 13

Calculate the NTF and STF of a delta-sigma modulator described by the ABCD matrix, assuming a quantizer gain of *k*.

`[ABCDc,tdac2] = realizeNTF_ct( ntf, form='FB', tdac, ordering=[1:n], bp=zeros(…), ABCDc)` page 14

Realize an NTF with a continuous-time loop filter.

`[sys, Gp] = mapCtoD(sys_c,t=[0 1],f0=0)` page 15

Map a continuous-time system to a discrete-time system whose impulse response matches the sampled pulse response of the original continuous-time system. See `dsexample3`.

`H = evalTFP(Hs,Hz,f)` page 16

`evalTFP` computes the value of the transfer function product `Hs*Hz` at a frequency `f`. Use this function to evaluate the signal transfer function of a CT system.

`[sv,sx,sigma_se,max_sx,max_sy]=simulateESL(v,mtf,M=16,dw=[1…],sx0=[0…])` page 17

Simulate the element-selection logic in a mismatch-shaping DAC.

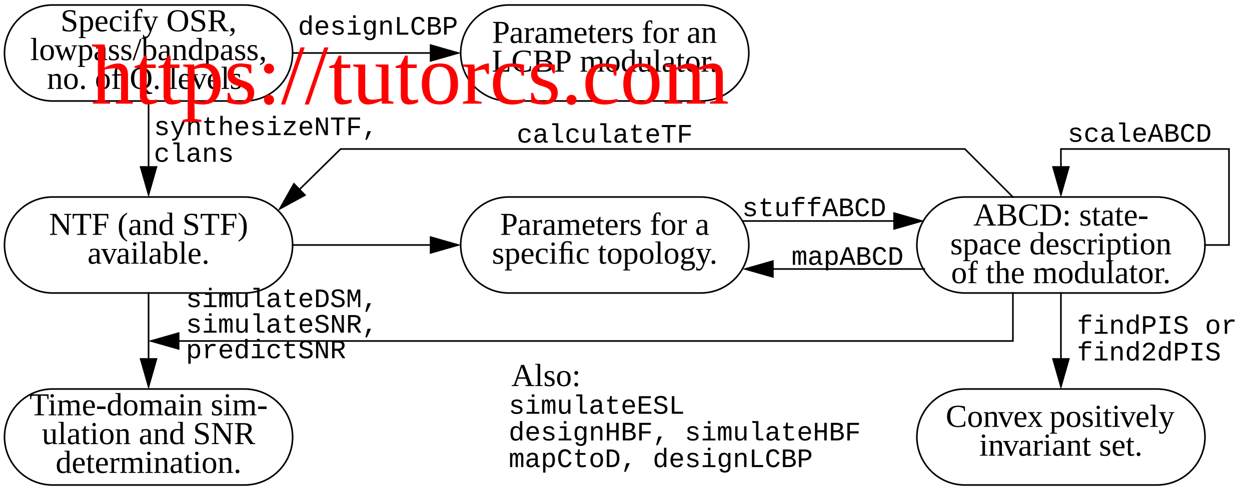

Specify OSR, lowpass/bandpass, no. of Q. levels
designLCBP
Parameters for an LCBP modulator.
synthesizeNTF, clans
calculateTF
scaleABCD
NTF (and STF) available.
Parameters for a specific topology.
stuffABCD
mapABCD
ABCD: state-space description of the modulator.
simulateDSM, simulateSNR, predictSNR
findPIS or find2dPIS
Time-domain simulation and SNR determination.
Also: simulateESL designHBF, simulateHBF mapCtoD, designLCBP
Convex positively invariant set.

Figure 1: Operations performed by the basic commands.

Functions for Quadrature Systems

`ntf = synthesizeQNTF(order=3,OSR=64,f0=0,NG=-60,ING=-20)` page 19

Synthesize a noise transfer function for a quadrature delta-sigma modulator.

`[v,xn,xmax,y] = simulateQDSM(u,ABCD` or `ntf,nlev=2,x0=0)` page 20

Simulate a quadrature delta-sigma modulator with a given input.

`ABCD = realizeQNTF(ntf,form='FB',rot=0,bn)` page 21

Co[illegible] transfer function into a complex ABCD matrix for the desi[illegible]

`ABCDr` [illegible] `[ABCDq ABCDp] = mapR2Q( ABCDr )` page 22

Con[illegible]to its real equivalent and vice versa.

`[ntf s`[illegible]`ulateQTF(ABCDr)` page 23

Cal[illegible]al transfer functions of a quadrature modulator.

`[sv,sx`[illegible]`_sy]= simulateQESL(v,mtf,M=16,sx0=[0…])` page 24

Si[illegible]tion Logic of a quadrature differential DAC.

Note: `simulateSNR` works for a quadrature modulator if given a complex NTF or ABCD matrix; `simulateDSM` can be used for a quadrature modulator if given an ABCDr matrix.

Specialty Functions

`[f1,f2,info] = designHBF(fp=0.2,delta=1e-5,debug=0)` page 25

Design a hardware-efficient half-band filter for use in a decimation or interpolation filter.

`y = simulateHBF(x,f1,f2,mode=0)` page 27

Simulate a Saramäki half-band filter.

`[snr,amp,k0,k1,sigma_e2] = predictSNR(ntf,R=64,amp=...,f0=0)` page 28

Predict the SNR vs. input amplitude curve of a binary modulator using the describing function method of Ardalan and Paulos.

`[s,e,n,o,Sc] = findPIS(u,ABCD,nlev=2,options)` page 29

Find a convex positively-invariant set for a delta-sigma modulator.

`[param,H,L0,ABCD,x] = designLCBP(n=3,OSR=64,opt=2,Hinf=1.6, f0=1/4,t=[0 1],form='FB',x0,dbg)` page 31

Design a continuous-time LC-bandpass modulator.

`[data, snr] = findPattern(N,OSR,NTF,ftest,Atest,f0,nlev,debug)`

Create a length-*N* bit-stream which has good spectral properties when repeated.

OTHER SELECTED FUNCTIONS

Delta-Sigma Utility

`mod1, mod2`

Set the `ABCD` matrix, `NTF` and `STF` of the standard 1st and 2nd-order modulators.

`snr = calculateSNR(hwfft,f)`

Esti[illegible]n-band bins of a Hann-windowed FFT and the location of the [illegible]

`[A B C` [illegible] `ABCD, m)`

Part[illegible] for an *m*-input state-space system.

`H_inf` [illegible]

Con[illegible] (maximum absolute value) of a z-domain transfer function[illegible]

`y = im`[illegible]

Con[illegible]pulse response from the comparator output back to the comparator input for the given NTF.

`y = pulse(S,tp=[0 1],dt=1,tfinal=10,nosum=0)`

Compute the sampled pulse response of a continuous-time system.

`sigma_H = rmsGain(H,f1,f2)`

Compute the root mean-square gain of the discrete-time transfer function `H` in the frequency band (`f1,f2`).

General Utility

`dbv(), dbp(), undbv(), undbp(), dbm(), undbm()`

The dB equivalent of voltage/power quantities, and their inverse functions.

`window = ds_hann(N)`

A Hann window of length `N`. Unlike MATLAB's original `hanning` function, `ds_hann` does not smear tones which are located exactly in an FFT bin (i.e. tones having an integral number of cycles in the given block of data). MATLAB 6's `hanning(N,'periodic')` function and MATLAB 7's `hann(N,'periodic')` function do the same as `ds_hann(N)`.

Graphing

`plotPZ(H,color='b',markersize=5,list=0)`

Plot the poles and zeros of a transfer function.

`plotSpectrum(X,fin,fmt)`

Plot a smoothed spectrum

`figureMagic(xRange,dx,xLab, yRange,dy,yLab, size)`

Performs a number of formatting operations for the current figure, including axis limits, ticks and labelling.

`printmif(file,size,font,fig)`

Print a figure to an Adobe Illustrator file and then use `ai2mif` to convert it to FrameMaker MIF format. `ai2mif` is an improved version of the function of the same name originally written by Deron Jackson <djackson@mit.edu>.

`[f,p] = logsmooth(X,inBin,nbin)`

Smooth the FFT `X`, and convert it to dB. See also `bplogsmooth` and `bilogplot`.

synthesizeNTF

Synopsis: `ntf = synthesizeNTF(order=3,OSR=64,opt=0,H_inf=1.5,f0=0)`
Synthesize a noise transfer function (NTF) for a delta-sigma modulator.

Arguments

order The order of the NTF. *order* must be even for bandpass modulators.

OSR The oversampling ratio. *OSR* need only be specified when optimized NTF zeros are requested.

opt A flag used to request optimized NTF zeros. `opt=0` puts all NTF zeros at band-center (DC for lowpass modulators). `opt=1` optimizes the NTF zeros. For even-order modulators, `opt=2` puts two zeros at band-center, but optimizes the rest.

H_inf The maximum out-of-band gain of the NTF. Lee's rule states that *H_inf*<2 should yield a stable modulator with a binary quantizer. Reducing *H_inf* increases the likelihood of success, but reduces the magnitude of the attenuation provided by the NTF and thus the theoretical resolution of the modulator.

f0 The center frequency of the modulator. `f0`≠`0` yields a bandpass modulator; `f0=0.25` puts the center frequency at $f_s/4$.

Output

ntf The modulator NTF, given as an LTI object in zero-pole form.

Bugs

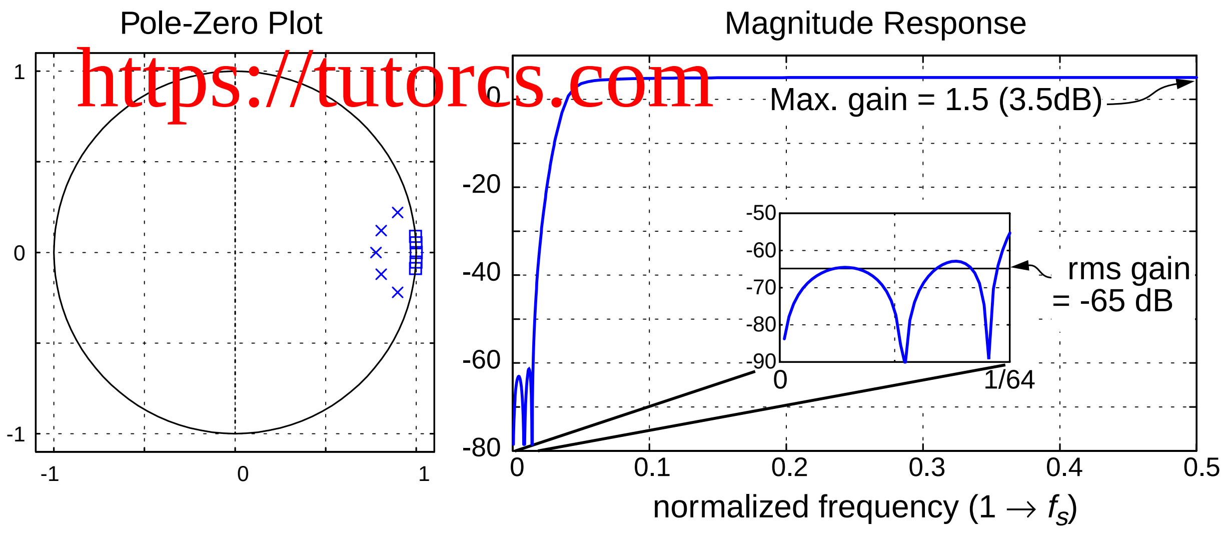
If *OSR* or *H_inf* are low, the NTF is not optimal. Use `synthesizeChebyshevNTF` instead.

Example

Fifth-order lowpass modulator; zeros optimized for an oversampling ratio of 32.

```
>> H = synthesizeNTF(5,32,1)
Zero/pole/gain:
       (z-1) (z^2 - 1.997z + 1) (z^2 - 1.992z + 1)
--------------------------------------------------------------
(z-0.7778) (z^2 - 1.613z + 0.6649) (z^2 - 1.796z + 0.8549)

Sampling time: 1
```



clans

Synopsis: `ntf = clans(order=4,OSR=64,Q=5,rmax=0.95,opt=0)`

Synthesize a noise transfer function (NTF) for a lowpass delta-sigma modulator using the CLANS (Closed-loop analysis of noise-shaper) methodology [1]. This function requires the optimization toolbox.

[1] J. arley, “Design of multibit noise-shaping data converters,” *Analog In Processing Journal*, vol. 3, pp. 259-272, 1993.

Argum

order e NTF.

OSR ing ratio.

Q number of quantization levels used by the fed-back quantization atically, $Q = \|h\|_1 - 1$, i.e. the sum of the absolute values of the impulse response samples minus 1, is the maximum instantaneous noise gain.)

rmax The maximum radius for the NTF poles.

opt A flag used to request optimized NTF zeros. `opt=0` puts all NTF zeros at band-center (DC for lowpass modulators). `opt=1` optimizes the NTF zeros. For even-order modulators, `opt=2` puts two zeros at band-center, but optimizes the rest.

Output

ntf The modulator NTF, given as an LTI object in zero-pole form.

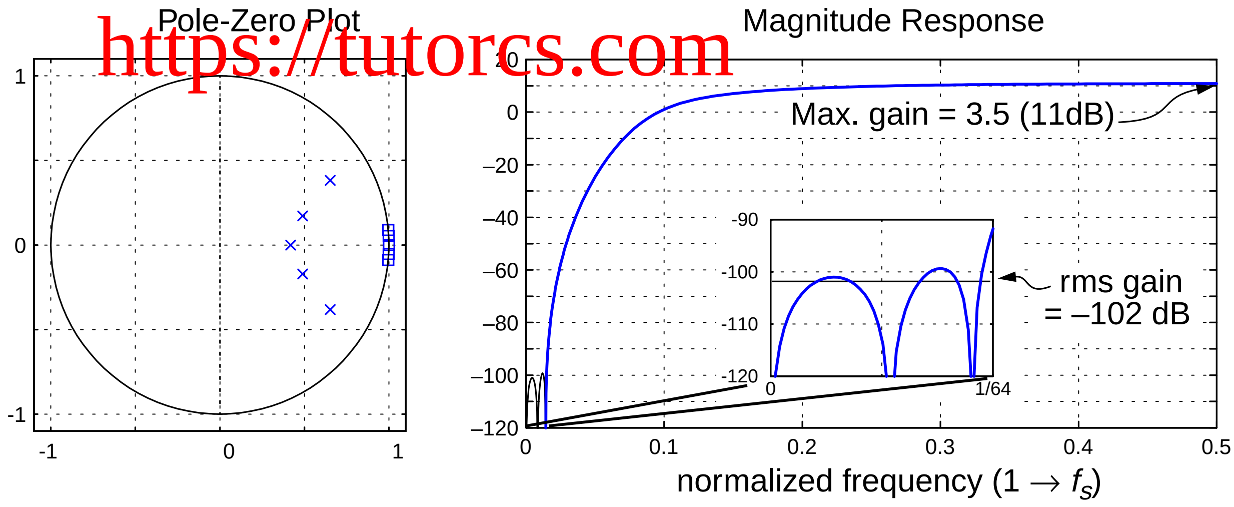
Example

Fifth-order lowpass modulator; (time-domain) noise gain of 5, zeros optimized for *OSR* = 32.

```
>> H= clans(5,32,5,.95,1)

Zero/pole/gain:
              (z-1) (z^2 - 1.997z + 1) (z^2 - 1.992z + 1)
----------------------------------------------------------
(z-0.4183) (z^2 - 0.9784z + 0.2685) (z^2 - 1.305z + 0.5714)

Sampling time: 1
```



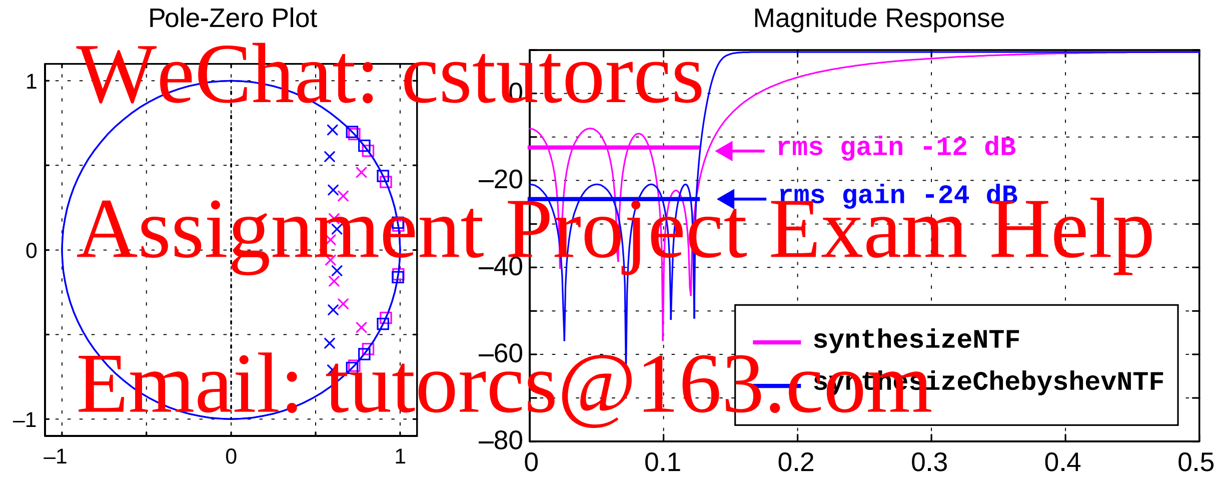
synthesizeChebyshevNTF

Synopsis: `ntf = synthesizeChebyshevNTF(order=3,OSR=64,opt=0,H_inf=1.5,f0=0)`
Synthesize a noise transfer function (NTF) for a delta-sigma modulator. `synthesizeNTF` assumes that magnitude of the denominator of the NTF is approximately constant in the passband. When the *OSR* or H_inf are low, this assumption breaks down and `synthesizeNTF` yields a non-optimal NTF. `sy`[illegible]`F` creates non-optimal NTFs, but fares better than `synthesizeNTF` in the a[illegible]nces.

Argum[illegible]
Same as [illegible]t that the *f0* argument is not supported yet.

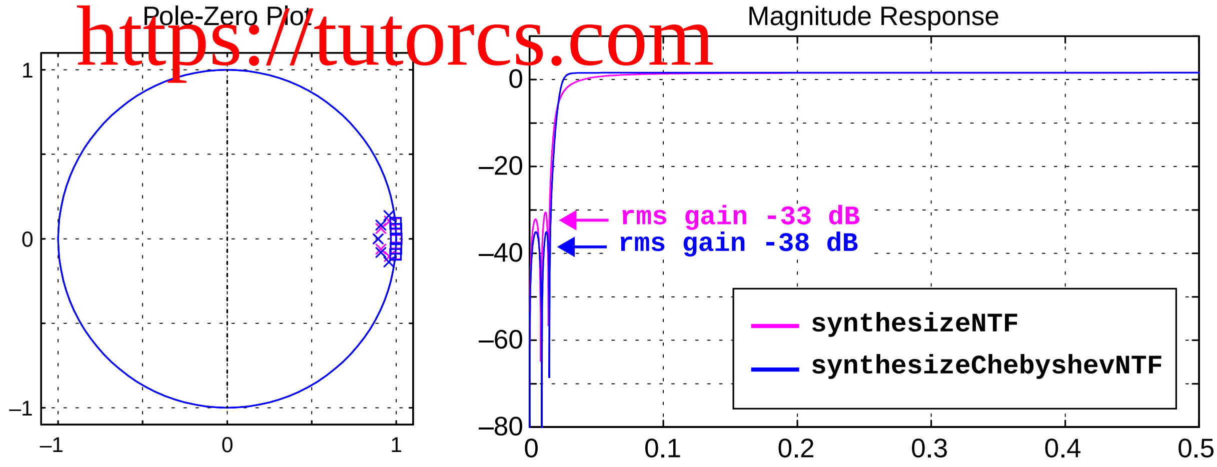
Examp[illegible]
Compa[illegible]`nthesizeNTF` and `synthesizeChebyshevNTF` when *OSR* is low:

```
>> OSR [illegible]nf = 3;
>> H1 = synthesizeNTF(order,OSR,1,H_inf);
>> H3 = synthesizeChebyshevNTF(order,OSR,1,H_inf);
```



Repeat for *H_inf* low:

```
>> OSR = 32; order = 5; H_inf = 1.2;
>> H1 = synthesizeNTF(order,OSR,1,H_inf);
>> H3 = synthesizeChebyshevNTF(order,OSR,1,H_inf);
```



simulateDSM

Synopsis: `[v,xn,xmax,y] = simulateDSM(u,ABCD|ntf,nlev=2,x0=0)`

Simulate a delta-sigma modulator with a given input. For maximum speed, make sure that the compiled `mex` file is on your search path (by typing `which simulateDSM` at the MATLAB prompt).

Argum

u ence to the modulator, given as a $m \times N$ row vector. *m* is the num- sually 1). Full-scale corresponds to an input of magnitude *nlev*–1.

ABCD escription of the modulator loop filter.

ntf NTF, given in zero-pole form.
STF is assumed to be unity.

nlev levels in the quantizer. Multiple quantizers are indicated by mak- y.

x0 The initial state of the modulator.

Output

v The samples of the output of the modulator, one for each input sample.

xn The internal states of the modulator, one for each input sample, given as an $n \times N$ matrix.

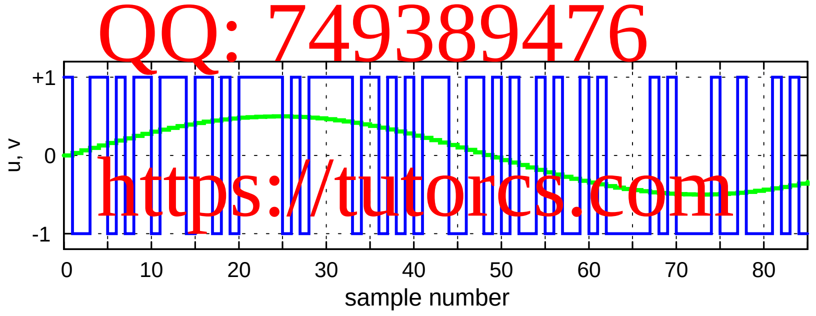
xmax The maximum absolute values of each state variable.

y The samples of the quantizer input, one per input sample.

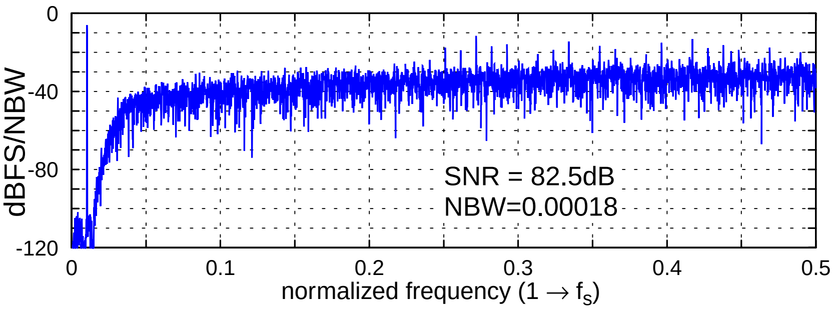
Example

Simulate a 5th-order binary modulator with a half-scale sine-wave input and plot its output in the time and frequency domains.

```
>> OSR = 32; H = synthesizeNTF(5,OSR,1);
>> N = 8192; fB = ceil(N/(2*OSR));
>> f=85; u = 0.5*sin(2*pi*f/N*[0:N-1]);
>> v = simulateDSM(u,H);
```



```
t = 0:85;
stairs(t, u(t+1),'g');
hold on;
stairs(t,v(t+1),'b');
axis([0 85 -1.2 1.2]);
ylabel('u, v');
```



```
spec=fft(v.*ds_hann(N))/(N/4);
plot(linspace(0,0.5,N/2+1), ...
  dbv(spec(1:N/2+1)));
axis([0 0.5 -120 0]);
grid on;
ylabel('dBFS/NBW')
snr=calculateSNR(spec(1:fB),f);
s=sprintf('SNR = %4.1fdB\n',snr)
text(0.25,-90,s);
s=sprintf('NBW=%7.5f',1.5/N);
text(0.25, -110, s);
```

simulateSNR

Synopsis: `[snr,amp] = simulateSNR(ntf|ABCD|function,OSR,amp,f0=0,nlev=2, f=1/(4*OSR),k=13,quadrature=0)`

Simulate a delta-sigma modulator with sine wave inputs of various amplitudes and calculate the signal-to-noise ratio (SNR) in dB for each input.

Argum

ntf NTF, given in zero-pole form.

ABCD escription of the modulator loop filter.

functior function taking the input signal as its sole argument.

OSR ing ratio. *OSR* is used to define the "band of interest."

amp sting the amplitudes to use. Defaults to [-120 -110...-20 -15 -10 -9 ere 0 dB means a full-scale (peak value = *nlev*-1) sine wave.

f0 The center frequency of the modulator. *f0*≠0 corresponds to a bandpass modulator.

nlev The number of levels in the quantizer.

f The normalized frequency of the test sinusoid; a check is made that the test frequency is in the band of interest. The frequency is adjusted so that it lies precisely in an FFT bin.

k The number of time points used for the FFT is 2^k.

quadrature A flag indicating that the system being simulated is quadrature. This flag is set automatically if either *ntf* or *ABCD* are complex.

Output

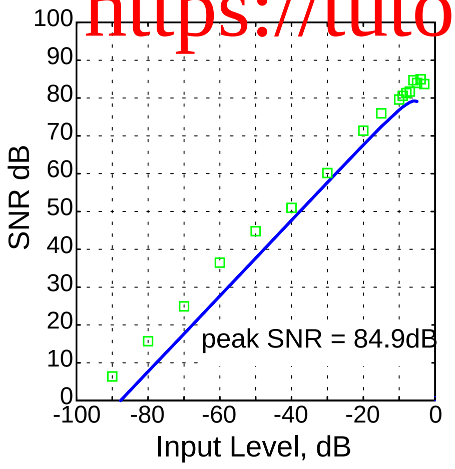
snr A row vector containing the SNR values calculated from the simulations.

amp A row vector listing the amplitudes used.

Example

Compare the SNR vs. input amplitude curve for a fifth-order modulator determined by the describing function method with that determined by simulation.

```
>> OSR = 32; H = synthesizeNTF(5,OSR,1);
>> [snr_pred,amp] = predictSNR(H,OSR);
>> [snr,amp] = simulateSNR(H,OSR);
```



```
plot(amp,snr_pred,'b',amp,snr,'gs');
grid on;
figureMagic([-100 0], 10, 2, ...
   [0 100], 10, 1);
xlabel('Input Level, dB');
ylabel('SNR dB');
s=sprintf('peak SNR = %4.1fdB\n',...
   max(snr));
text(-65,15,s);
```

realizeNTF

Synopsis: `[a,g,b,c] = realizeNTF(ntf,form='CRFB',stf=1)`

Convert a noise transfer function (NTF) into a set of coefficients for a particular modulator topology.

Arguments

ntf NTF, given in zero-pole form (i.e. a zpk object).

form ying the modulator topology.

ascade-of-resonators, feedback form.

ascade-of-resonators, feedforward form.

ascade-of-integrators, feedback form.

ascade-of-integrators, feedforward form.

ny of the above, but the quantizer is delaying.

Structures are described in detail in “MODULATOR MODEL DETAILS” on page 33.

stf The modulator STF, specified as a zpk object. Note that the poles of the STF must match those of the NTF in order to guarantee that the STF can be realized without the addition of extra state variables.

Output

a Feedback/feedforward coefficients from/to the quantizer $(1 \times n)$.

g Resonator coefficients $(1 \times \lfloor n/2 \rfloor)$.

b Feed-in coefficients from the modulator input to each integrator $(1 \times n + 1)$.

c Integrator inter-stage coefficients. $(1 \times n$. In unscaled modulators, *c* is all ones.)

Example

Determine the coefficients for a 5th-order modulator with the cascade-of-resonators structure, feedback (CRFB) form.

```
>> H = synthesizeNTF(5,32,1);
>> [a,g,b,c] = realizeNTF(H,'CRFB')
a = 0.0007    0.0084    0.0550    0.2443    0.5579
g = 0.0028    0.0079
b = 0.0007    0.0084    0.0550    0.2443    0.5579    1.0000
c = 1      1      1      1      1
```

See Also

`realizeNTF_ct` on page 14 to realize an NTF with a continuous-time loop filter.

stuffABCD

Synopsis: `ABCD = stuffABCD(a,g,b,c,form='CRFB')`
Calculate the ABCD matrix given the parameters of a specified modulator topology.

Arguments

a Feedback/feedforward coefficients from/to the quantizer. 1 × n

g Resonator coefficients. 1 × $\lfloor n/2 \rfloor$

b Feed-in coefficients from the modulator input to each integrator. 1 × $n+1$

c Integrator inter-stage coefficients. 1 × n

form see `realizeNTF` on page 10 for a list of supported structures.

Output

ABCD A state-space description of the modulator loop filter.

mapABCD

`[a,g,b,c] = mapABCD(ABCD,form='CRFB')`
Calculate the parameters for a specified modulator topology, assuming ABCD fits that topology.

Arguments

ABCD A state-space description of the modulator loop filter.

form see `realizeNTF` on page 10 for a list of supported structures.

Output

a Feedback/feedforward coefficients from/to the quantizer. 1 × n

g Resonator coefficients. 1 × $\lfloor n/2 \rfloor$

b Feed-in coefficients from the modulator input to each integrator. 1 × $n+1$

c Integrator inter-stage coefficients. 1 × n

scaleABCD

Synopsis: `[ABCDs,umax]=scaleABCD(ABCD,nlev=2,f=0,xlim=1,ymax=nlev+5,umax,N=1e5)`

Scale the ABCD matrix so that the state maxima are less than a specified limit. The maximum stable input is determined as a side-effect of this process.

Arguments

ABCD [illegible] escription of the modulator loop filter.

nlev [illegible] levels in the quantizer.

f [illegible] frequency of the test sinusoid.

xlim [illegible] e states. May be given as a vector.

ymax [illegible] or judging modulator stability. If the quantizer input exceeds [illegible] ulator is considered to be unstable.

Output

*ABCD*s The scaled state-space description of the modulator loop filter.

umax The maximum stable input. Input sinusoids with amplitudes below this value should not cause the modulator states to exceed their specified limits.

calculateTF

Synopsis: `[ntf,stf] = calculateTF(ABCD,k=1)`
Calculate the NTF and STF of a delta-sigma modulator.

Arguments

ABCD [illegible] description of the modulator loop filter.

k [illegible] e for the quantizer gain.

Output

ntf [illegible] NTF, given as an LTI system in zero-pole form.

stf [illegible] STF, given as an LTI system in zero-pole form.

Examp[illegible]

Realize [illegible] with the cascade-of-resonators structure, feedback form. Calculate the ABCD matrix of the loop filter and verify that the NTF and STF are correct.

```
>> H = synthesizeNTF(5,32,1)

Zero/pole/gain:
       (z-1) (z^2 - 1.997z + 1) (z^2 - 1.992z + 1)
----------------------------------------------------------
(z-0.7778) (z^2 - 1.613z + 0.6649) (z^2 - 1.796z + 0.8549)

Sampling time: 1
>> [a,g,b,c] = realizeNTF(H)
a =
    0.0007    0.0084    0.0550    0.2443    0.5579
g =
    0.0028    0.0079
b =
    0.0007    0.0084    0.0550    0.2443    0.5579    1.0000
c =
     1     1     1     1     1
>> ABCD = stuffABCD(a,g,b,c)
ABCD =
    1.0000         0         0         0         0    0.0007   -0.0007
    1.0000    1.0000   -0.0028         0         0    0.0084   -0.0084
    1.0000    1.0000    0.9972         0         0    0.0633   -0.0633
         0         0    1.0000    1.0000   -0.0079    0.2443   -0.2443
         0         0    1.0000    1.0000    0.9921    0.8023   -0.8023
         0         0         0         0    1.0000    1.0000         0
>> [ntf,stf] = calculateTF(ABCD)
Zero/pole/gain:
       (z-1) (z^2 - 1.997z + 1) (z^2 - 1.992z + 1)
----------------------------------------------------------
(z-0.7778) (z^2 - 1.613z + 0.6649) (z^2 - 1.796z + 0.8549)

Sampling time: 1

Zero/pole/gain:
1

Static gain.
```

realizeNTF_ct

Synopsis: `[ABCDc,tdac2] = realizeNTF_ct( ntf, form='FB', tdac, ordering=[1:n], bp=zeros(…), ABCDc)`

Realize a noise transfer function (NTF) with a continuous-time loop filter.

Arguments

ntf [illegible] NTF, specified as an LTI object in zero-pole form.

form [illegible] ying the modulator topology.

[illegible] eedback form.

[illegible] eedforward form.

tdac [illegible] the feedback DAC(s). If `tdac(1)`≥1, direct feedback terms are [illegible] antizer. Multiple timings (one or more per integrator) for the FB [illegible] e specified by making tdac a cell array, e.g.

`tdac = {[1,2]; [1 2]; {[0.5 1],[1 1.5]}; []};`

In this example, the first two integrators have DACs with `[1,2]` timing, the third has a pair of dacs, one with `[0.5 1]` timing and the other with `[1 1.5]` timing, and there is no direct feedback DAC to the quantizer.

ordering A vector specifying which NTF zero-pair to use in each resonator. Default is for the zero-pairs to be used in the order specified in the NTF.

bp A vector specifying which resonator sections are bandpass. The default (zeros(…)) is for all sections to be lowpass.

ABCDc The loop filter structure, in state-space form. If this argument is omitted, ABCDc is constructed according to *form*.

Output

ABCDc A state-space description of the CT loop filter.

tdac2 A matrix with the DAC timings, including ones that were automatically added.

Example

Realize the NTF $(1-z^{-1})^2$ with a CT system (cf. the example on page 15).

```
>> ntf = zpk([1 1],[0 0],1,1);
>> [ABCDc,tdac2] = realizeNTF_ct(ntf,'FB')
ABCDc =
         0         0    1.0000   -1.0000
    1.0000         0         0   -1.5000
         0    1.0000         0    0.0000
tdac2 =
    -1    -1
     0     1
```

mapCtoD

Synopsis: `[sys, Gp] = mapCtoD(sys_c,t=[0 1],f0=0)`

Map a MIMO continuous-time system to a SIMO discrete-time equivalent. The criterion for equivalence is that the sampled pulse response of the CT system must be identical to the impulse response of the DT system. I.e. if y_c is the output of the CT system with an input v_c taken from a set of D[illegible] DT input v, then y, the output of the equivalent DT system with input v [illegible] for integer n. The DACs are characterized by rectangular impulse respons[illegible]fied in the t matrix.

Argum[illegible]

sys_c [illegible]ption of the CT system.

t [illegible] of the DAC pulse used to make CT waveforms from DT inputs. [illegible]sponds to one of the system inputs; [-1 -1] denotes a CT input. The default is [0 1] for all inputs except the first, which is assumed to be a CT input.

f0 The (normalized) frequency at which the Gp filters' gains are to be set to unity. Default 0 (DC).

Output

sys The LTI description for the DT equivalent.

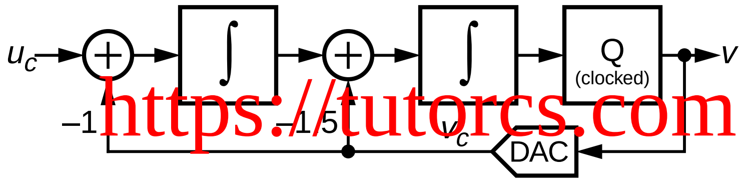
Gp The mixed CT/DT prefilters which form the samples fed to each state for the CT inputs.

Reference

[1] R. Schreier and B. Zhang, “Delta-sigma modulators employing continuous-time circuitry,” IEEE Transactions on Circuits and Systems I, vol. 43, no. 4, pp. 324-332, April 1996.

Example

Map the standard second-order CT modulator shown below to its DT equivalent and verify that the NTF is $(1-z^{-1})^2$.



$$\begin{bmatrix} \dot{x}_{1c} \\ \dot{x}_{2c} \\ y_c \end{bmatrix} = \left[\begin{array}{cc|cc} 0 & 0 & 1 & -1 \\ 1 & 0 & 0 & -1.5 \\ \hline 0 & 1 & 0 & 0 \end{array}\right] \begin{bmatrix} x_{1c} \\ x_{2c} \\ u_c \\ v_c \end{bmatrix}$$

```
>> LFc = ss([0 0;1 0], [1 -1;0 -1.5], [0 1], [0 0]);
>> tdac = [0 1];
>> [LF,Gp] = mapCtoD(LFc,tdac);
>> ABCD = [LF.a LF.b; LF.c LF.d];
>> H = calculateTF(ABCD)

Zero/pole/gain:
(z-1)^2
-------
  z^2

Sampling time: 1
```

evalTFP

Synopsis: `H = evalTFP(Hs,Hz,f)`

Use this function to evaluate the signal transfer function of a continuous-time (CT) system. In this context `Hs` is the open-loop response of the loop filter from the *u* input and `Hz` is the closed-loop noise transfer function.

Argum [illegible]

Hs [illegible] ime transfer function in zpk form.

Hz [illegible] transfer function in zpk form.

f [illegible] quencies.

Output [illegible]

H [illegible] $_s(j2\pi f)H_z(e^{j2\pi f})$.

evalTFP attempts to cancel poles in H_s with zeros in H_z.

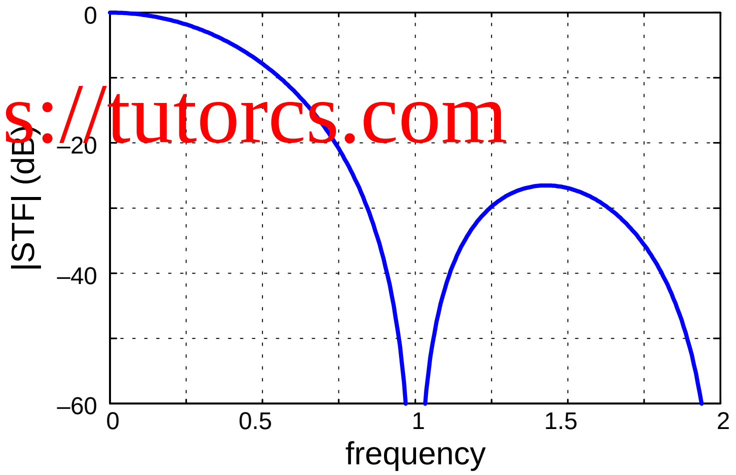
See Also

`evalMixedTF` is a more advanced version of this function which is used to evaluate the individual feed-in transfer functions of a CT modulator.

Example

Plot the STF of the 2^{nd}-order CT system depicted on page 15.

```
Ac = [ 0 0; 1 0];      Bc = [1 -1; 0 -1.5];
Cc = [0 1];            Dc = [0 0];
LFc = ss(Ac, Bc, Cc, Dc);
L0c = zpk(ss(Ac,Bc(:,1),Cc,Dc(1)));
tdac = [0 1];
[LF,Gp] = mapCtoD(LFc,tdac);
ABCD = [LF.a LF.b; LF.c LF.d];
H = calculateTF(ABCD);
% Yields H=(1-z^-1)^2
f = linspace(0,2,300);
STF = evalTFP(L0c,H,f);
plot(f,dbv(STF));
```



simulateESL

Synopsis: `[sv,sx,sigma_se,max_sx,max_sy]`
`= simulateESL(v,mtf,M=16,dw=[1…],sx0=[0…])`

Simulate the element selection logic (ESL) of a multi-element DAC using a particular mismatch-shaping transfer function (`mtf`).

[1] R. “Noise-shaped multibit D/A convertor employing unit elements,” *El*, no. 20, pp. 1712-1713, Sept. 28 1995.

Argum

v ning the number of elements to enable. Note that the output of nust be offset and scaled in order to be used here as *v* must be in $\sum_i^M dw(i)]$.

mtf shaping transfer function, given in zero-pole form.

M elements.

dw A vector containing the weight associated with each element.

sx0 An $n \times M$ matrix containing the initial state of the element selection logic.

Output

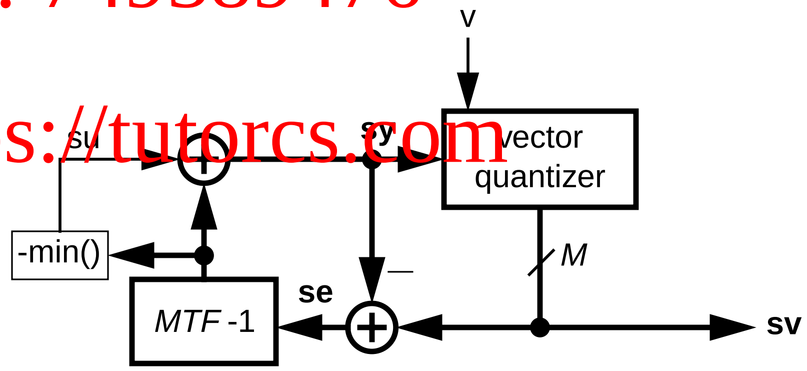
sv The selection vector: a vector of zeros and ones indicating which elements to enable.

sx An $n \times M$ matrix containing the final state of the element selection logic.

sigma_se The rms value of the selection error, $\boldsymbol{se} = \boldsymbol{sv} - \boldsymbol{sy}$. *sigma_se* may be used to analytically estimate the power of in-band noise caused by element mismatch.

max_sx The maximum value attained by any state in the ESL.

max_sy The maximum value attained by any component of the (un-normalized) “desired usage” vector.

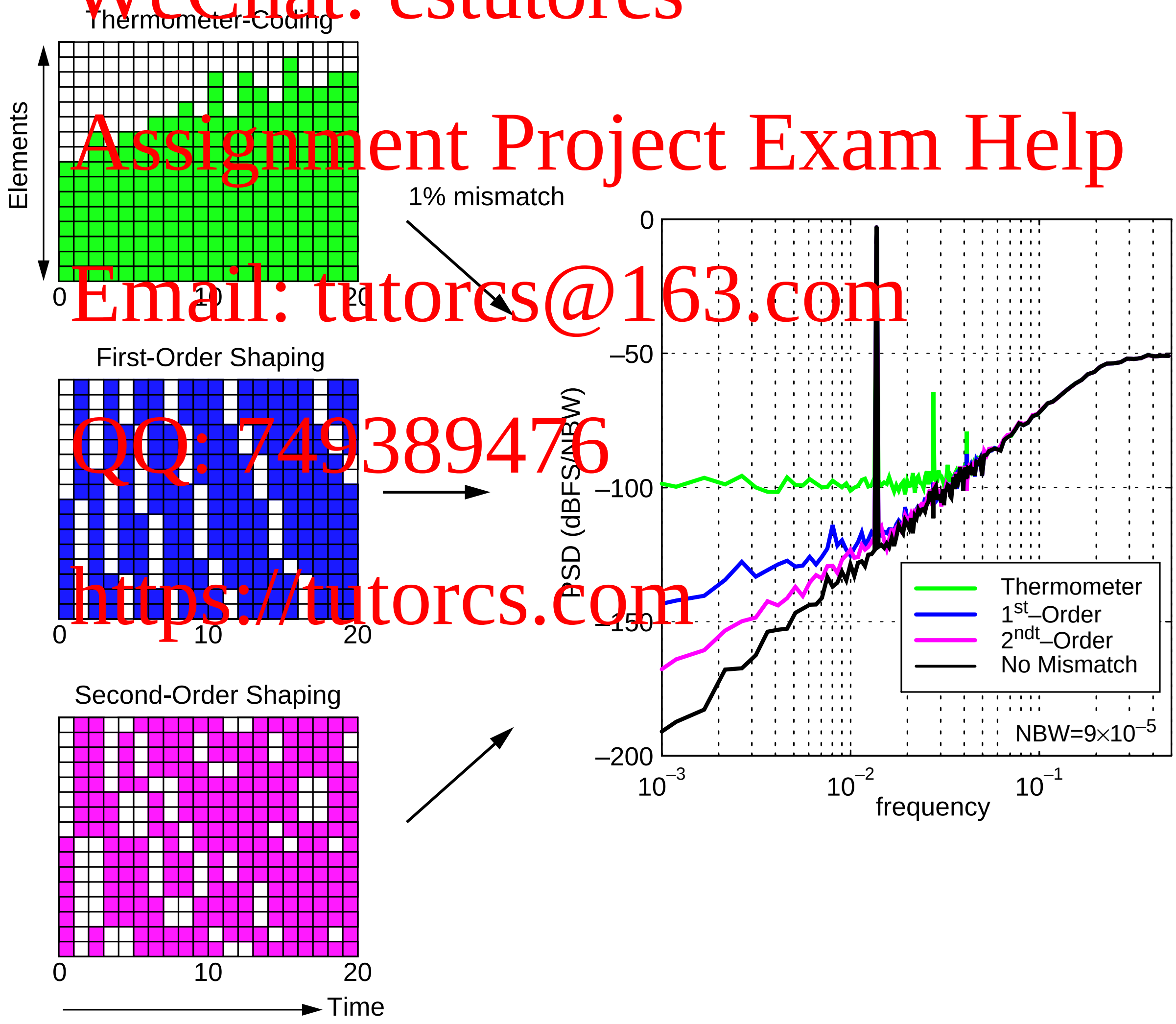


Block diagram of the Element Selection Logic

Example (From `dsdemo5`)

Compare the usage patterns and example spectra for a 16-element DAC driven with thermometer-coded, 1st-order mismatch-shaped and 2nd-order mismatch-shaped data generated by a 3rd-order multi-bit modulator.

```
>> ntf = synthesizeNTF(3,[],[],4);
>> M = 16;
>> N = 2^14;
>> fin                2));
>> u =                /N)*fin*[0:N-1]);
>> v =                +1);
>> v =                                        % scale v to [0,M]
>> sv0
>> mtf                                        %First-order shaping
>> sv1                1,M);
>> mtf                0 ], 1, 1);            %Second-order shaping
>> sv2                2,M);
>> ue                 );                      % 1% mismatch
>> dv0
>> spec0 = fft(dv0.*ds_hann(N))/(M*N/8);
>> plotSpectrum(spec0,fin,'g');
...
```



synthesizeQNTF

Synopsis: `ntf = synthesizeQNTF(order=3,OSR=64,f0=0,NG=-60,ING=-20,n_im=order/3)`
Synthesize a noise transfer function (NTF) for a quadrature delta-sigma modulator.

Arguments

order e NTF.

OSR ing ratio.

f0 uency of the modulator. *f0*≠0 yields a bandpass modulator; e center frequency at $f_s/4$.

NG d noise gain (dB).

ING -band noise gain (dB).

n_im ge-band zeros.

Output

ntf The modulator NTF, given as an LTI object in zero-pole form.

Bugs

ALPHA VERSION. This function uses an experimental ad hoc method that is neither optimal nor robust.

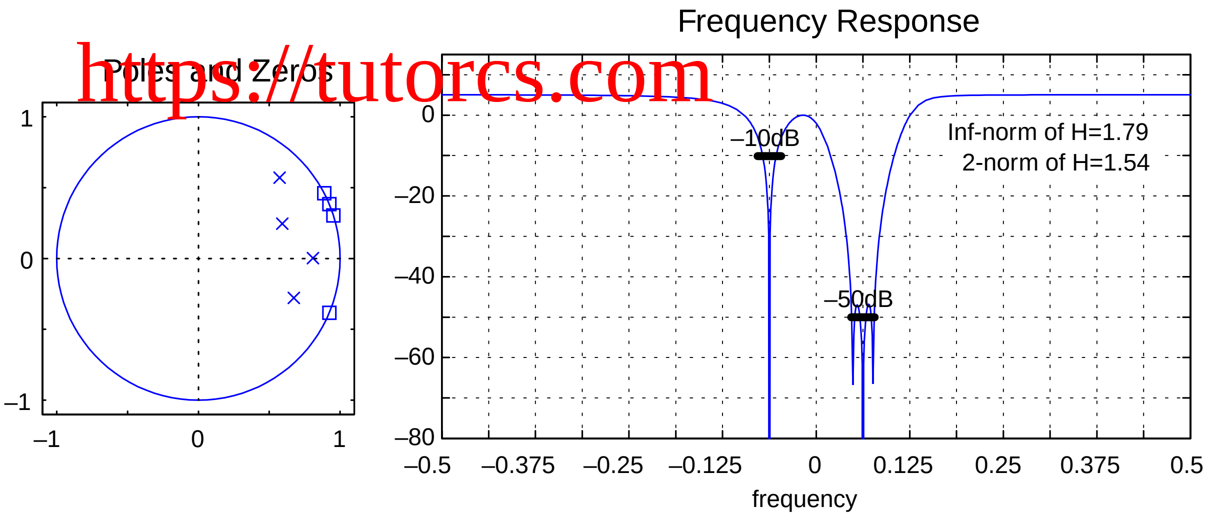
Example (From `dsexample4`)

Fourth-order, $OSR = 32$, $f_0 = 1/16$ bandpass NTF with an rms in-band noise gain of –50 dB and an image-band noise gain of –10 dB.

```
>> ntf = synthesizeQNTF(4,32,1/16,-50,-10);

Zero/pole/gain:
         (z-(0.953+0.3028i)) (z^2 - 1.848z + 1) (z-(0.888+0.4598i))
-------------------------------------------------------------------------------------
 (z-(0.8088+0.002797i)) (z-(0.5913+0.2449i)) (z-(0.6731-0.2788i)) (z-(0.5739+0.5699i))

Sampling time: 1
```



simulateQDSM

Synopsis: `[v,xn,xmax,y] = simulateQDSM(u,ABCD|ntf,nlev=2,x0=0)`

Simulate a quadrature delta-sigma modulator with a given input. For improved simulation speed, use `simulateDSM` with a 2-input/2-output ABCDr argument as indicated in the example below.

Arguments

u The input sequence to the modulator, given as a $1 \times N$ row vector. Full-scale corresponds to an input of magnitude *nlev*–1.

ABCD A state-space description of the modulator loop filter.

ntf The modulator NTF, given in zero-pole form. The modulator STF is assumed to be unity.

nlev The number of levels in the quantizer. Multiple quantizers are indicated by making *nlev* an array.

x0 The initial state of the modulator.

Output

v The samples of the output of the modulator, one for each input sample.

xn The internal states of the modulator, one for each input sample, given as an $n \times N$ matrix.

xmax The maximum absolute values of each state variable.

y The samples of the quantizer input, one per input sample.

Example

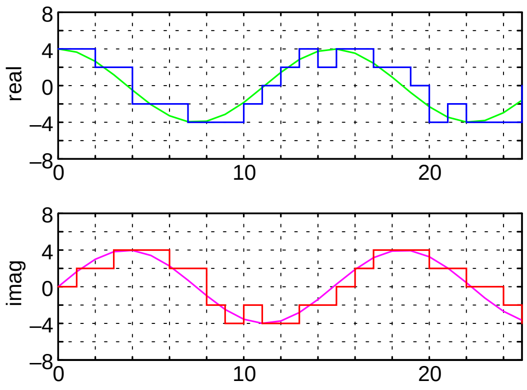
Simulate a 4th-order 9-level $f_0 = 1/16$ quadrature modulator with a half-scale sine-wave input and plot its output in the time and frequency domains.

```
nlev = 9; f0 = 1/16; osr = 32; M = nlev-1;
ntf = synthesizeQNTF(4,osr,f0,-50,-10);
N = 64*osr; f = round((f0+0.3*0.5/osr)*N)/N;
u = 0.5*M*exp(2i*pi*f*[0:N-1]);
v = simulateQDSM(u,ntf,nlev);
```

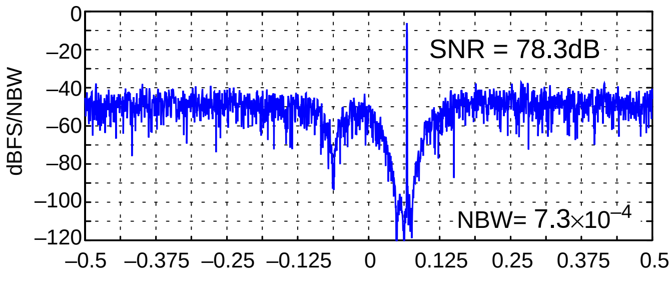
faster code

```
ABCD = realizeQNTF(ntf,'FF');
ABCDr = mapQtoR(ABCD);
ur = [real(u); imag(u)];
vr=simulateDSM(ur,ABCDr,nlev*[1;1]);
v = vr(1,:) + 1i*vr(2,:);
```

```
t = 0:25;
subplot(211)
plot(t, real(u(t+1)),'g');
hold on;
stairs(t,real(v(t+1)),'b');
figureMagic(…)
ylabel('real');
```



```
spec = fft(v.*ds_hann(N))/(M*N/2);
spec = [fftshift(spec) spec(N/2+1)];
plot(linspace(-0.5,0.5,N+1), dbv(spec))
figureMagic([-0.5 0.5],1/16,2, [-120 0],10,2)
ylabel('dBFS/NBW')
[f1 f2] = ds_f1f2(osr,f0,1);
fb1 = round(f1*N); fb2 = round(f2*N);
fb = round(f*N)-fb1;
snr = calculateSNR(spec(N/2+1+[fb1:fb2]),fb);
text(f,-10,sprintf(' SNR = %4.1fdB\n',snr));
text(0.25, -105, sprintf('NBW=%0.1e',1.5/N));
```



realizeQNTF

Synopsis: `ABCD = realizeQNTF(ntf,form='FB',rot=0,bn)`
Convert a quadrature noise transfer function (NTF) into an ABCD matrix for the desired structure.

Arguments

ntf The modulator NTF, given in zero-pole form (i.e. a zpk object).

form [illegible] ying the modulator topology.

[illegible] eedback

[illegible] rallel feedback

[illegible] eedforward

[illegible] rallel feedforward

[illegible] B with delaying quantizer. NOT SUPPORTED YET

[illegible] F with delaying quantizer. NOT SUPPORTED YET

rot *rot* =1 means rotate states to make as many coefficients as possible real

bn The coefficient of the auxiliary DAC for *form* = ’FF’

Output

ABCD State-space description of the loop filter.

Example (From `dsexample4`)
Determine coefficients for the parallel feedback (PFB) structure..

```
>> ntf = synthesizeQNTF(5,32,1/16,-50,-10);
>> ABCD = realizeQNTF(ntf,'PFB',1)
ABCD =
  Columns 1 through 4
   0.8854 + 0.4648i        0                  0                  0
   0.0065 + 1.0000i   0.9547 + 0.2974i        0                  0
        0             0.9715 + 0.2370i   0.9088 + 0.4171i        0
        0                  0             0.8797 + 0.4755i   0.9376 + 0.3477i
        0                  0                  0                  0
        0                  0                  0            -0.9916 - 0.1294i
  Columns 5 through 7
        0             0.0025             0.0025 + 0.0000i
        0                  0             0.0262 + 0.0000i
        0                  0             0.1791 + 0.0000i
        0                  0             0.6341 + 0.0000i
   0.9239 - 0.3827i        0             0.1743 + 0.0000i
  -0.9312 - 0.3645i        0                  0
```

mapQtoR

Synopsis: `ABCDr = mapQtoR(ABCD)`
Convert a quadrature matrix into its real (IQ) equivalent.

Arguments

ABCD [illegible] rix describing a quadrature system.

Output

ABCDr [illegible] orresponding to ABCD. Each element *z* in ABCD is replaced by a [illegible] BCDr. Specifically

$$[illegible] \begin{matrix} -y \\ x \end{matrix}\Bigg] \text{ where } x = Re(z) \text{ and } y = Im(z)\,.$$

mapRtoQ

Synopsis: `[ABCDq ABCDp] = mapR2Q( ABCDr )`
Map a real `ABCD` to a quadrature `ABCD`. `ABCDr` has its states paired (real, imaginary).

Arguments

ABCDr A real matrix describing a quadrature system.

Output

ABCDq The quadrature (complex) version of `ABCDr`.

ABCDp The mirror-image system matrix. `ABCDp` is zero if `ABCDr` has no quadrature errors.

calculateQTF

Synopsis: `[ntf stf intf istf] = calculateQTF(ABCDr)`
Calculate the noise and signal transfer functions for a quadrature modulator.

Arguments

ABCDr [illegible] ce description of the modulator loop filter. I/Q asymmetries may [illegible] he description. These asymmetries result in non-zero image [illegible] ns.

Output

ntf, stf [illegible] signal transfer functions.

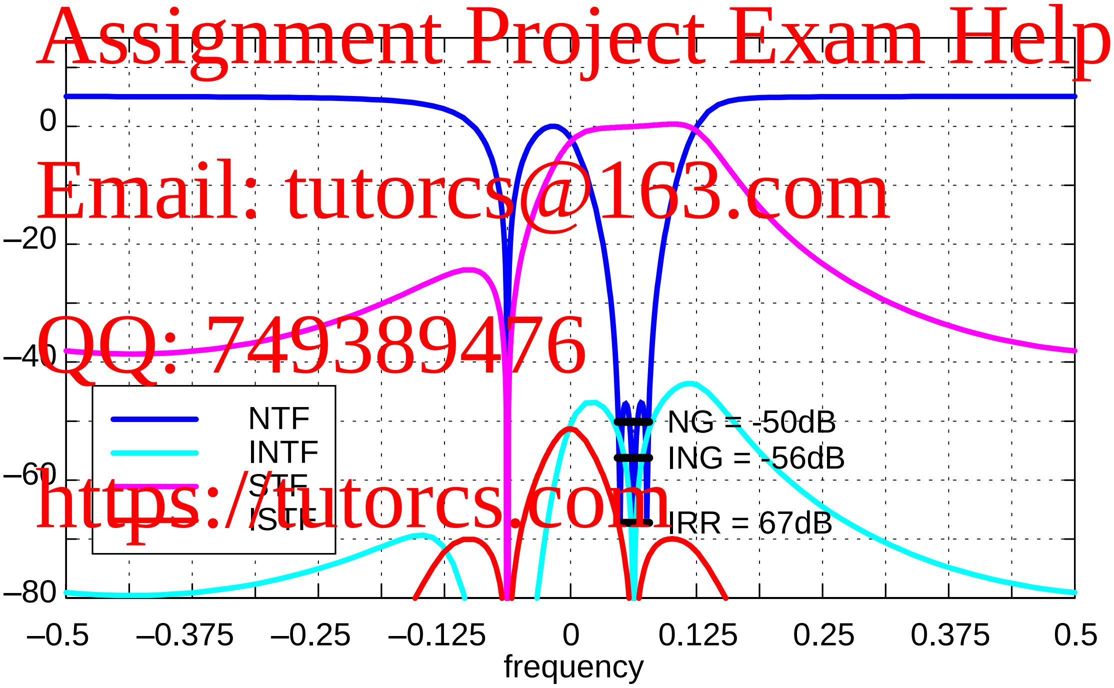
intf, istf [illegible] e and image signal transfer functions.

All tran[illegible] ed as LTI systems in zero-pole form.

Example (Continuing from `dsexample4`)

Examine the effect of mismatch in the first feedback.

```
>> ABCDr = mapQtoR(ABCD);
>> ABCDr(2,end) = 1.01*ABCDr(2,end);    % 0.1% mismatch in first feedback
>> [H G HI GI] = calculateQTF(ABCDr);
```



simulateQESL

Synopsis: `[sv,sx,sigma_se,max_sx,max_sy] = simulateQESL(v,mtf,M=16,sx0=[0…])`
Simulate the element selection logic (ESL) for a quadrature differential DAC.

Arguments

v A ... the digital input values.

mtf shaping transfer function, given in zero-pole form.

M elements. There is a total 2*M* elements.

sx0 ix whose columns are the initial state of the ESL.

Output

sv ector: a vector of zeros and ones indicating which elements to

sx An $n \times M$ matrix containing the final state of the ESL.

sigma_se The rms value of the selection error, $\mathbf{se} = \mathbf{sv} - \mathbf{sy}$. *sigma_se* may be used to analytically estimate the power of in-band noise caused by element mismatch.

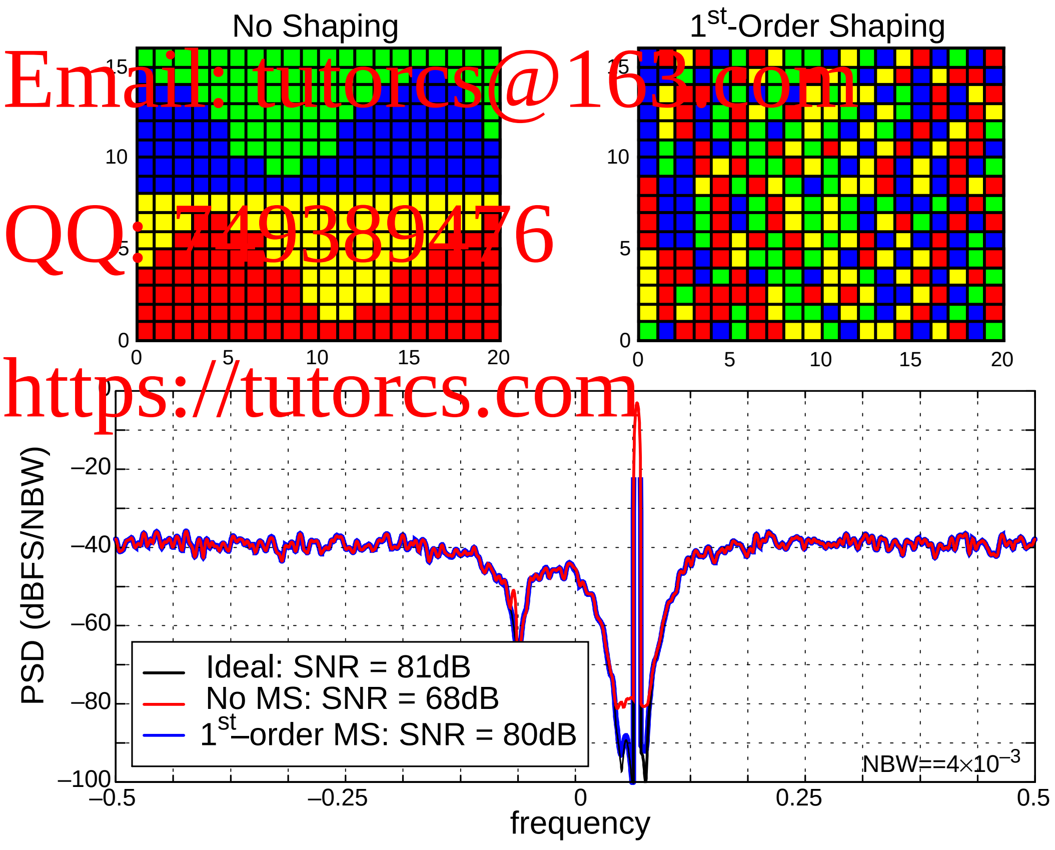
max_sx The maximum absolute value attained by any state in the ESL.

max_sy The maximum absolute value attained by any component of the input to the VQ.

Example

Apply mismatch-shaping to the modulator from dsexample4.

```
>> mtf1 = zpk(exp(2i*pi*f0),0,1,1);          %First-order complex shaping
>> sv1 = simulateQESL(v,mtf1,M);
```



designHBF

Synopsis: `[f1,f2,info]=designHBF(fp=0.2,delta=1e-5,debug=0)`

Design a hardware-efficient linear-phase half-band filter for use in the decimation or interpolation filter associated with a delta-sigma modulator. This function is based on the procedure described by Saramäki [1]. Note that since the algorithm uses a non-deterministic search procedure, successive cal[illegible] signs.

[1] T. [illegible] FIR filters as a tapped cascaded interconnection of identical su[illegible] *ions on Circuits and Systems*, vol. 34, pp. 1011-1029, 1987.

Argum[illegible]

fp [illegible]ssband cutoff frequency.

delta [illegible]topband ripple in absolute value.

Output[illegible]

f1,f2 Prototype filter and subfilter coefficients and their canonical-signed digit (csd) representation.

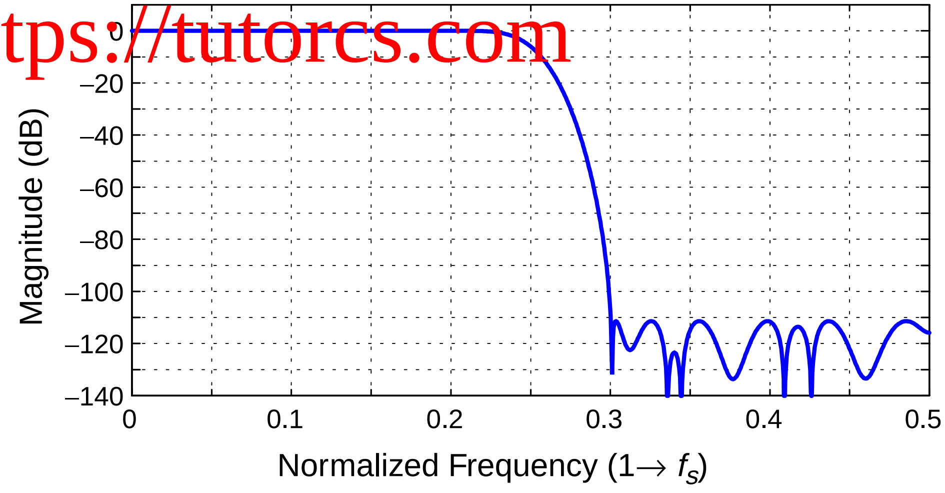
info A vector containing the following information data (only set when `debug=1`):
complexity The number of additions per output sample.
n1,n2 The length of the f1 and f2 vectors.
sbr The achieved stop-band attenuation (dB).
phi The scaling factor for the F2 filter.

Example

Design of a lowpass half-band filter with a cut-off frequency of $0.2f_s$, a passband ripple of less than 10^{-5} and a stopband rejection of at least 10^{-5} (-100 dB).

```
>> [f1,f2] = designHBF(0.2,1e-5);
>> f = linspace(0,0.5,1024);
>> plot(f, dbv(frespHBF(f,f1,f2)))
```

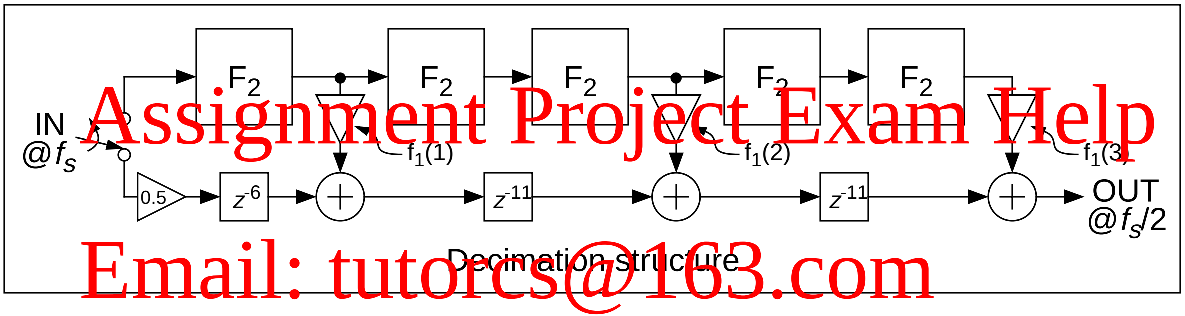
A plot of the filter response is shown below. The filter achieves 109 dB of attenuation in the stopband and uses only 124 additions (no true multiplications) to produce each output sample.



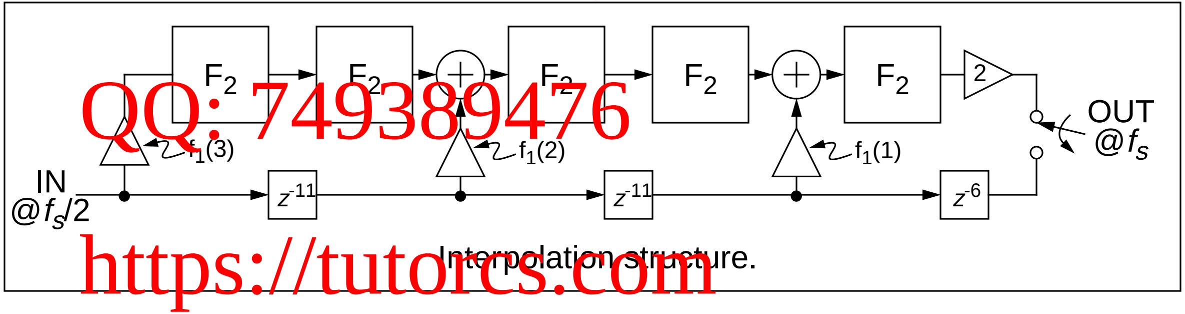
The structure of this filter as a decimation or interpolation filter is shown below. The coefficients and their signed-digit decompositions are

```
 [f1.val]' =      [f2.val]' =         >> f1.csd              >> f2.csd
      0.9453           0.6211         ans =                  ans =
     -0.6406          -0.1895               0    -4    -7          -1     -3     -8
      0.1953           0.0957               1    -1     1           1      1     -1
                      -0.0508         ans =                  ans =
                       0.0269              -1    -3    -6          -2     -4     -9
                      -0.0142              -1    -1    -1          -1      1     -1
                                      ans =                  ans =
                                           -2    -4    -7          -3     -5     -9
                                            1    -1     1           1     -1      1
                                                             ans =
                                                                   -4     -7     -8
                                                                   -1      1      1
                                                             ans =
                                                                   -5     -8    -11
                                                                    1     -1     -1
                                                             ans =
                                                                   -6     -9    -11
                                                                   -1      1     -1
```

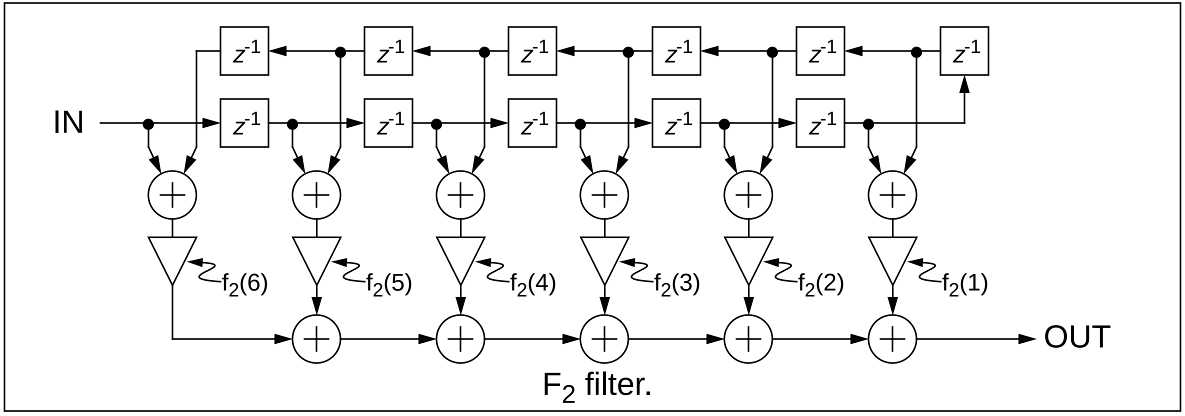
In the si he first row contain the powers of two while the second row gives their signs. For example, $f_1(1) = 0.9453 = 2^0 - 2^{-4} + 2^{-7}$ and $f_2(1) = 0.6211 = 2^{-1} + 2^{-3} - 2^{-8}$. Since the filter coefficients for this example use only 3 signed digits, each multiply-accumulate operation shown in the diagram below needs only 3 binary additions. Thus, an implementation of this 110th-order FIR filter needs to perform only 3×3 + 5×(3×6 + 6 -1) = 124 additions at the low ($f_s/2$) rate.



Decimation structure



Interpolation structure.



F_2 filter.

simulateHBF

Synopsis: `y = simulateHBF(x,f1,f2,mode=0)`

Simulate a Saramaki half-band filter (see `designHBF` on page 25) in the time domain.

Arguments

x

f1,f2 …nts. f1 and f2 can be vectors of values or struct arrays like those …`designHBF`.

mode …determines whether the input is filtered, interpolated, or deci-…g to the following:

…ng, no interpolation or decimation.

…s interpolated

…is decimated, even samples are taken.

3 The output is decimated, odd samples are taken.

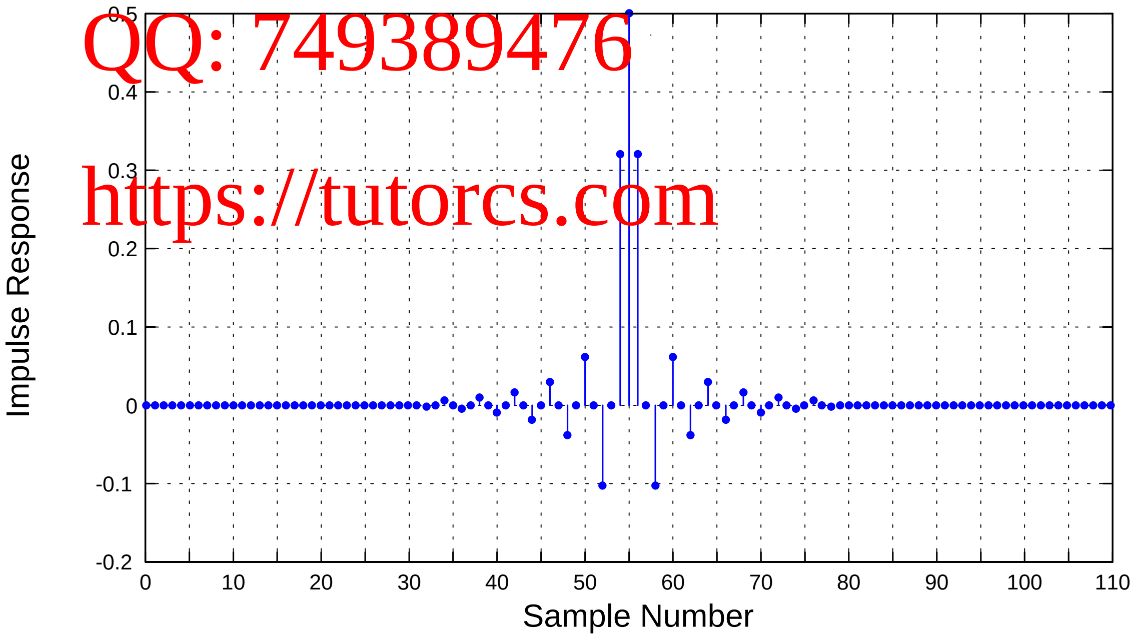
Output

y The output data.

Example

Plot the impulse response of the HBF designed on the previous page.

```
>> N = (2*length(f1)-1)*2*(2*length(f2)-1)+1;
>> y = simulateHBF([1 zeros(1,N-1)],f1,f2);
>> stem([0:N-1],y);
>> figureMagic([0 N-1],5,2, [-0.2 0.5],0.1,1)
>> printmif('HBFimp', [6 3], 'Helvetica8')
```



predictSNR

Synopsis: `[snr,amp,k0,k1,sigma_e2] = predictSNR(ntf,OSR=64,amp=...,f0=0)`

Use the describing function method of Ardalan and Paulos [1] to predict the signal-to-noise ratio (SNR) in dB for various input amplitudes. This method is only applicable to binary modulators.

[1] S. H. Ardalan and J. J. Paulos, “Analysis of nonlinear behavior in delta-sigma modulators,” *IE[illegible]cuits and Systems*, vol. 34, pp. 593-603, June 1987.

Argum[illegible]

ntf [illegible] NTF, given in zero-pole form.

OSR [illegible]ing ratio. *OSR* is used to define the “band of interest.”

amp [illegible]sting the amplitudes to use. Defaults to [-120 -110...-20 -15 -10 -9 [illegible]here 0 dB means a full-scale (peak value = 1) sine wave.

f0 [illegible]uency of the modulator. *f0*≠0 corresponds to a bandpass modulator.

Output

snr A row vector containing the predicted SNRs.

amp A row vector listing the amplitudes used.

k0 The signal gain of the quantizer model; one value per input level.

k1 The noise gain of the quantizer model; one value per input level.

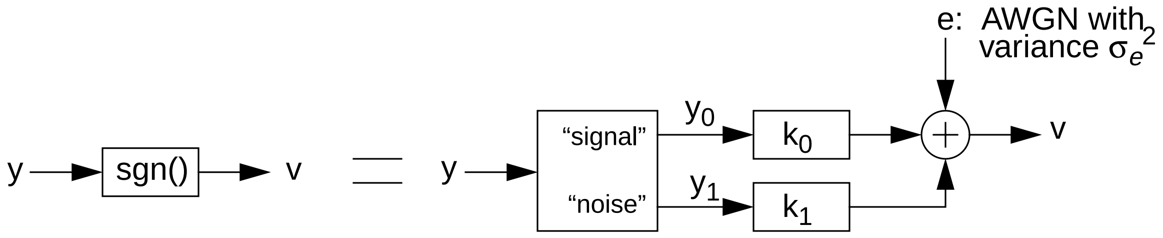
sigma_e2 The mean square value of the noise in the model of the quantizer.

Example

See the example on page 9.

The Quantizer Model:

The binary quantizer is modeled as a pair of linear gains and a noise source, as shown in the figure below. The input to the quantizer is divided into signal and noise components which are processed by signal-dependent gains k_0 and k_1. These signals are added to a noise source, which is assumed to be white and to have a Gaussian distribution (the variance σ_e^2 is also signal-dependent), to produce the quantizer output.



findPIS, find2dPIS (in the `PosInvSet` subdirectory)

Synopsis:

```
[s,e,n,o,Sc] = findPIS(u,ABCD,nlev=2,options)
options = [dbg=0 itnLimit=2000 expFactor=0.005 N=1000 skip=100
           qhullArgA=0.999 qhullArgC=.001]
s = find2dPIS(u,ABCD,options)
options = [dbg=0 itnLimit=100 expFactor=0.01 N=1000 skip=100]
```

Find a c [illegible] nt set for a delta-sigma modulator. `findPIS` requires compilation of the q [illegible] `IS` does not, but is limited to second-order systems.

Argum [illegible]

u [illegible] modulator. If *u* is a scalar, the input to the modulator is constant. [illegible] vector, the input to the modulator may be any sequence whose [illegible] he range $[u(1), u(2)]$.

ABCD [illegible] escription of the modulator loop filter.

nlev The number of quantizer levels.

dbg Set dbg=1 to get a graphical display of the iterations.

itnLimit The maximum number of iterations.

expFactor The expansion factor applied to the hull before every mapping operation. Increasing expFactor decreases the number of iterations but results in sets which are larger than they need to be.

N The number of points to use when constructing the initial guess.

skip The number of time steps to run the modulator before observing the state. This handles the possibility of “transients” in the modulator.

qhullArgA The ‘A’ argument to the qhull program. Adjacent facets are merged if the cosine of the angle between their normals is greater than the absolute value of this parameter. Negative values imply that the merge operation is performed during hull construction, rather than as a post-processing step.

qhullArgC The ‘C’ argument to the qhull program. A facet is merged into its neighbor if the distance between the facet’s centrum (the average of the facet’s vertices) and the neighboring hyperplane is less than the absolute value of this parameter. As with the above argument, negative values imply pre-merging while positive values imply post-merging.

Output

s The vertices of the set ($dim \times n_v$).

e The edges of the set, listed as pairs of vertex indices ($2 \times n_e$).

n The normals for the facets of the set ($dim \times n_f$).

o The offsets for the facets of the set ($1 \times n_f$).

Sc The scaling matrix which was used internally to “round out” the set.

Background

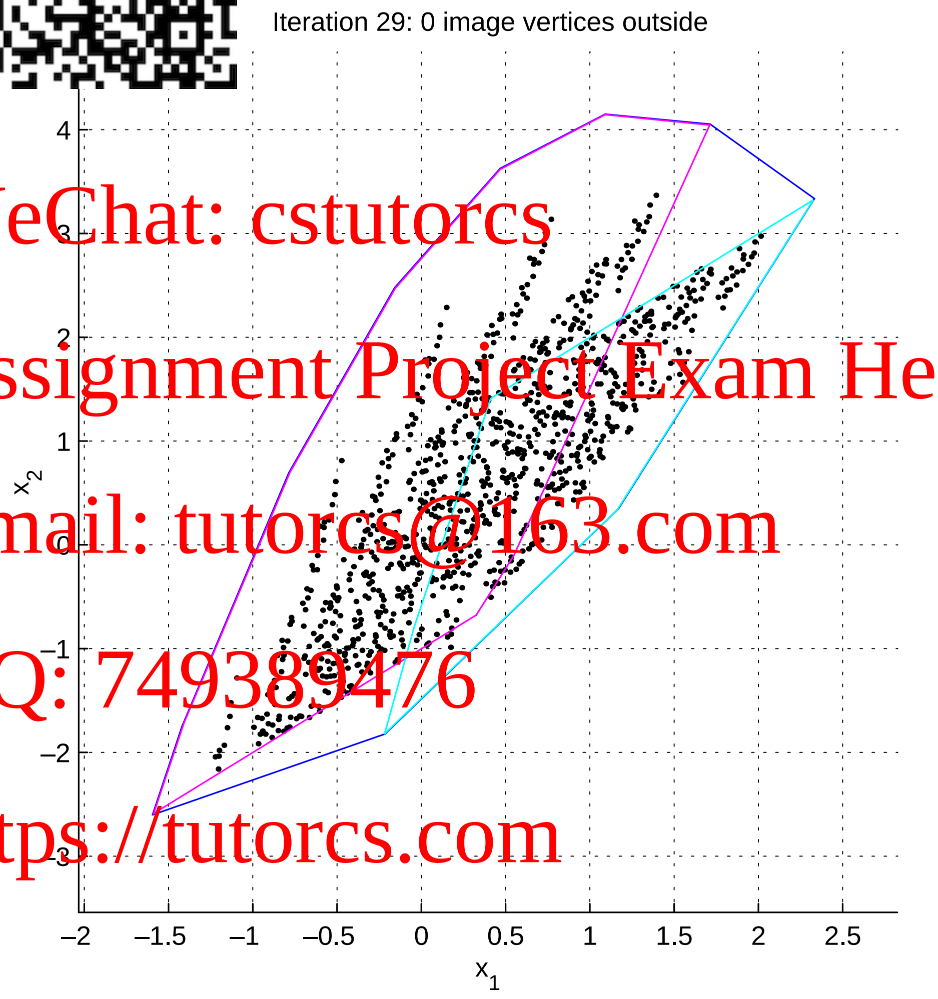
This is an implementation of the method described in [1].

[1] R. Schreier, M. Goodson and B. Zhang “An algorithm for computing convex positively invariant sets for delta-sigma modulators,” *IEEE Transactions on Circuits and Systems I*, vol. 44, no. 1, pp. 38-44, January 1997.

Example

Find a positively-invariant set for the second-order modulator with an input of $1/\sqrt{7}$.

```
>> ABCD = [
     1     0     1    -1
     1     1     1    -2
     0     1     0     0];
>> s = find2dPIS(sqrt(1/7),ABCD,1)
s =
  Colu
   -1.      1700    2.3324    1.7129    1.0904    0.4672
   -2.      3498    3.3359    4.0550    4.1511    3.6277
  Colu
   -0.      4205   -1.5954
    2.      7462   -2.6019
```



designLCBP

Synopsis: `[param,H,L0,ABCD,x]=`
`designLCBP(n=3,OSR=64,opt=2,Hinf=1.6,f0=1/4,t=[0 1],form='FB',x0,dbg)`

Design a continuous-time LC-bandpass modulator consisting of resistors and LC tanks driven by transconductors and current-source DACs. Dynamic range and impedance scaling are not applied.

Argum

n tanks in the loop filter.

OSR ratio, $OSR = f_s/(2f_b)$

opt g whether optimized NTF zeros should be used or not.

H_inf out-of-band gain of the NTF. See `synthesizeNTF` on page 5.

f0 er frequency, default = $f_s/4$.

t mes of the DAC feedback pulse. Note that $t_1 = 0$ implies the use of a comparator with zero delay, which is usually an impractical situation.

form The specific form of the modulator. See the table and diagram below.

dbg Set this to a non-zero value to observe the optimization process in action.

Output

param A struct containing the (`n`, `OSR`, `Hinf`, `f0`, `t` and `form`) arguments plus the fields given in the table below.

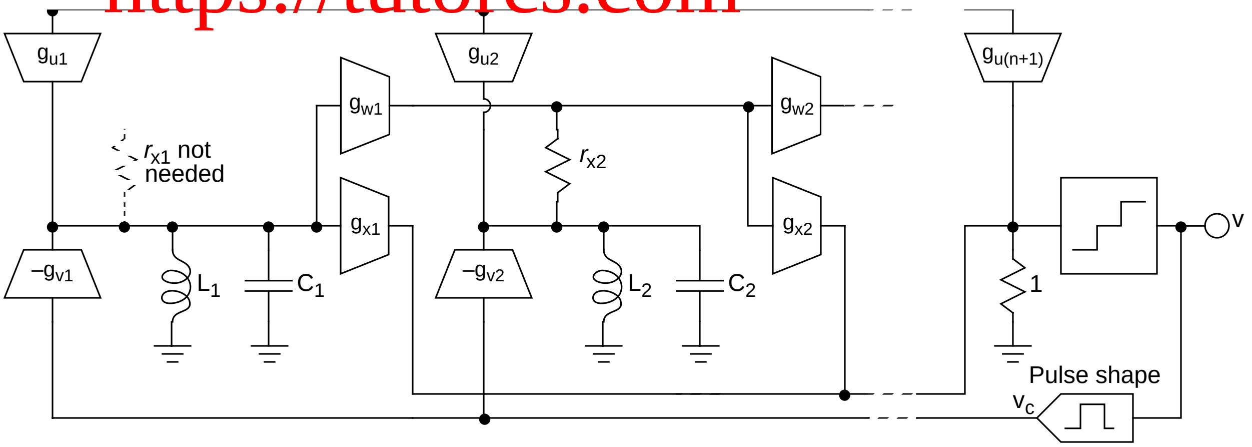
H The NTF of the equivalent discrete-time modulator.

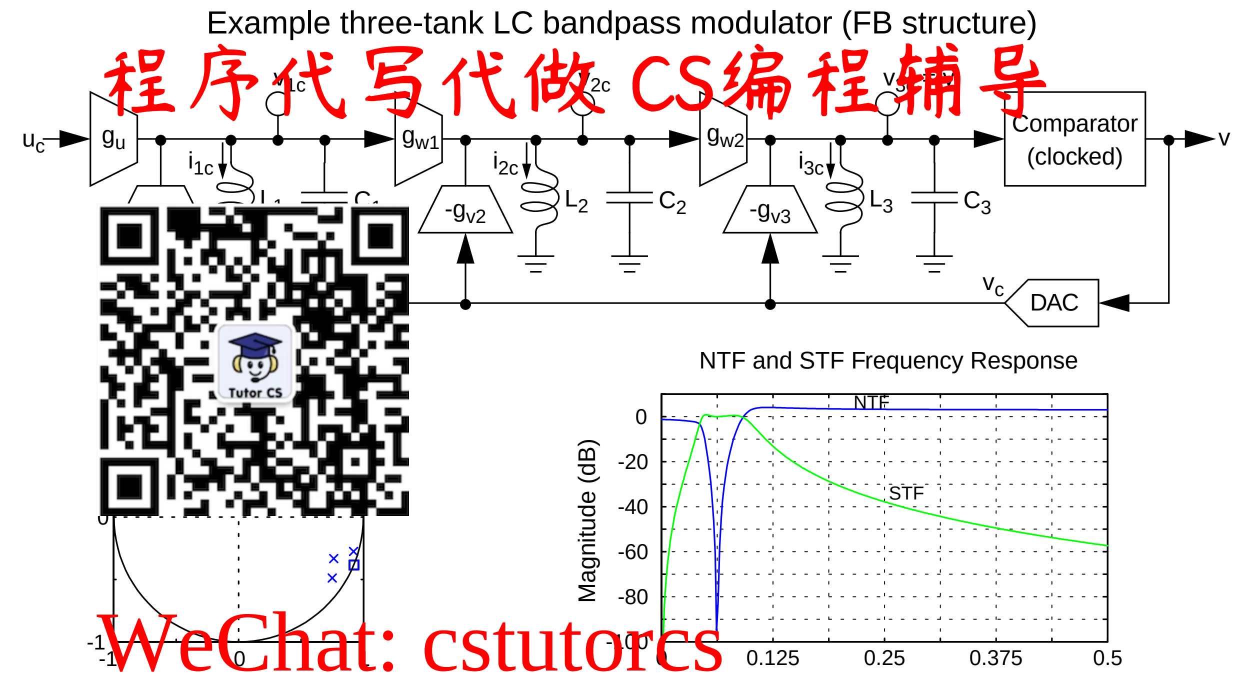
L0 The continuous-time loop filter; an LTI object in zero-pole form.

General LC topology and the coefficients subject to optimization for each of the supported forms:

Form	L	C	g_u: $1 \times (n+1)$	g_v: $1 \times n$	g_w: $1 \times (n-1)$	g_x: $1 \times n$	r_x: $1 \times n$
FB	$\frac{1}{\sqrt{2\pi f_0}}$		[1 0 …]*	$[x_1\ x_2 \ldots\ x_n]$	[1…]	[0 0… 1]	[0 …]
FF			[1 0 … 1]*	[1 0 …]	[1…]	$[x_1\ x_2 \ldots\ x_n]$	[0 …]
R			[1 0 … 1]*	$[x_1\ 0\ \ldots]$	[1…]	[0 0… 1]	$[0\ x_2 \ldots x_n]$

* scaled for unity STF gain at f_0.





Example

```
>> n = 3; OSR = 64; opt = 0; Hinf = 1.6; f0 = 1/16; t = [0.5 1]; form = 'FB';
>> [param,H,L0,ABCD,x] = designLCBP(n,OSR,opt,Hinf,f0,t,form);
>> plotPZ(H);
>> f = linspace(0,0.5,300); z = exp(2*pi*j*f);
>> ntf = dbv(evalTF(H,z)); stf = dbv(evalTFP(L0,H,f));
>> plot( f, ntf,’b’, f, stf,’g’);
>> figureMagic( [0,0.5],1/16,2, [-100 10],10,2);
gu = 1     0     0     0
gv = 0.653 2.708 3.708
gw = 1     1
gx = 0     0     1
rx = 0     0     0
```

See Also

`[H,L0,ABCD,k]=LCparam2tf(param,k=1)`

This function computes the transfer functions, quantizer gain and ABCD representation of an LC system. Inductor series resistance (`rl`) and capacitor shunt conductance (`gc`) can be incorporated into the calculations. Finite input and output conductance of the transconductors can be lumped with `gc`.

Bugs

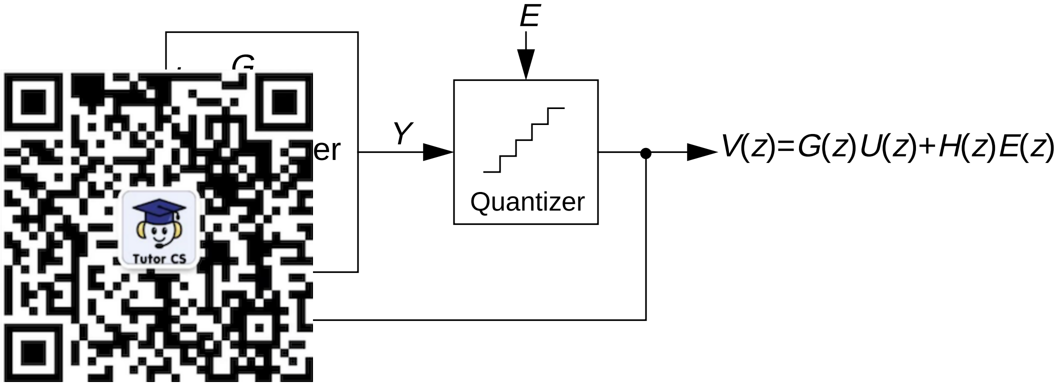
The use of the `constr/fmincon` function (from the optimization toolbox, versions 5 & 6) makes convergence of `designLCBP` erratic and unreliable. In some cases, editing the `LCObj*` functions helps. A more robust optimizer/objective function, perhaps one which supports a step-size restriction, is needed.

`designLCBP` is outdated, now that LC modulators which use more versatile stages in the back-end have been developed. See [1] for an example design.

[1] R. Schreier, J. Lloyd, L. Singer, D. Paterson, M. Timko, M. Hensley, G. Patterson, K. Behel, J. Zhou and W. J, Martin, “A 10-300 MHz IF-digitizing IC with 90-105 dB dynamic range and 15-333 kHz bandwidth,” *IEEE Journal of Solid-State Circuits*, vol. SC-37, no. 12, pp. 1636-1644, Dec. 2002.

Modulator Model Details

A delta-sigma modulator with a single quantizer is assumed to consist of quantizer connected to a loop filter as shown in the diagram below.



The Loop Filter

The loop filter is described by an ABCD matrix. For single-quantizer systems, the loop filter is a two-input, one-output linear system and ABCD is an $(n+1)\times(n+2)$ matrix, partitioned into A $(n\times n)$, B $(n\times 2)$, C $(1\times n)$ and D (1×2) sub-matrices as shown below:

$$ABCD = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]. \qquad (1)$$

The equations for updating the state and computing the output of the loop filter are

$$\begin{aligned} x(n+1) &= Ax(n) + B\begin{bmatrix} u(n) \\ v(n) \end{bmatrix} \\ y(n) &= Cx(n) + D\begin{bmatrix} u(n) \\ v(n) \end{bmatrix}. \end{aligned} \qquad (2)$$

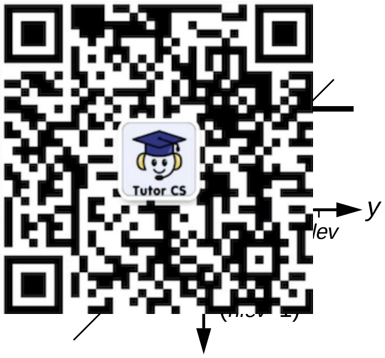
This formulation is sufficiently general to encompass all single-quantizer modulators which employ linear loop filters. The toolbox currently supports translation to/from an ABCD description and coefficients for the following topologies:

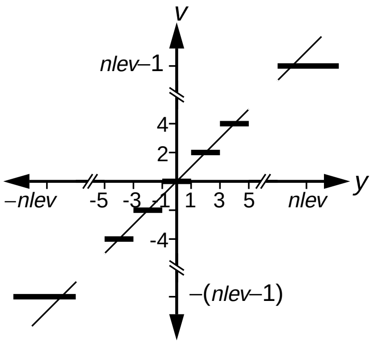
CIFB	Cascade-of-integrators, feedback form.
CIFF	Cascade-of-integrators, feedforward form.
CRFB	Cascade-of-resonators, feedback form.
CRFF	Cascade-of-resonators, feedforward form.
CRFBD	Cascade-of-resonators, feedback form, delaying quantizer.
CRFFD	Cascade-of-resonators, feedforward form, delaying quantizer

Multi-input and multi-quantizer systems can also be described with an ABCD matrix and Eq. (2) will still apply. For an n_i-input, n_o-output modulator, the dimensions of the sub-matrices are A: $n\times n$, B: $n\times(n_i+n_o)$, C: $n_o\times n$ and D: $n_o\times(n_i+n_o)$.

The Quantizer

The quantizer is ideal, producing integer outputs centered about zero. Quantizers with an even number of levels are of the mid-rise type and produce outputs which are odd integers. Quantizers with an odd number of levels are of the mid-tread type and produce outputs which are even integers.





Transfer curve of a quantizer with an even number of levels.

Transfer curve of a quantizer with an odd number of levels.