Scilab Comm Toolbox Reference Card


Conventions
- Normalized frequency (between -0.5 and 0.5): $\nu$ (Nyquist is $\nu = 1/2$)
- $f_s$ = Sampling frequency
- $f_i$ = Intermediate frequency
- $f_{symb}$ = Symbol frequency

1. Modulations

\begin{verbatim}
wf = wf_init(name[,opt,ps]) Initialize a waveform object. Parameters can be:
   - 'bpsk' Binary Phase Shift Keying
   - 'qpsk' Quadrature Phase Shift Keying
   - 'psk' - Phase Shift Keying
   - 'psk' [M=2] Phase Shift Keying, M is the number of phase positions (2 for BPSK)
   - 'opsk' Offset QPSK
   - 'fsk' [M=2, index=0.5, filt='n', BT=0.8] Frequency Shift Keying, filt can be 'n' (no pulse shaping, e.g. NRZ), or 'g' (gaussian), in which case BT is the Bandwidth-Time product
   - 'gfsk' [index=0.5, BT=0.8] Gaussian 2-levels FSK
   - '4gfsk' [index=0.5, BT=0.8] Gaussian 4-levels FSK
   - 'ask' Minimum Shift Keying (FSK with index = 0.5)
   - 'gask' [BT=0.8] Gaussian filtered MSK
   - 'ask' [M=2] Amplitude Shift Keying, M is the number of levels.
   - 'ook' On-off keying.
   - 'qam16', 'qam64', 'qam256' Quadrature Amplitude Modulations

ps is the pulse shaping filter specification, it can be:
   - 'none' No pulse shaping (e.g. dirac directly transmitted)
   - 'nrz' Moving average filter (rectangular pulses)
   - 'rc', ntap, Raised cosine, with specified roll-off factor
   - 'errc', ntap, Raised cosine, with specified roll-off factor
   - 'gaussian', ntap, BT Gaussian filter (applied to NRZ). BT is the bandwidth-time product.

wf = wf_set_filter(wf, ps) Setup the pulse shaping filter for a given modulation (if not already specified in wf_init). The parameters in ps are the same as above.
\end{verbatim}

2. Graphical functions

\begin{verbatim}
plot_eye(x,T) Eye diagram (T is the symbol period)
plot_ber(EbNo, ber[,format]) Bit error rate plot (empirical)
plot_ber(wf[,format]) Bit error rate plot (theoretical)
plot_const(x, 'i') Constellation plot. x = complex vector or modulation object. With 'i' option : force isometric scale.
plot_psd(x[fs]) Power Spectral Density (PSD). fs = sampling frequency.
plot_impulse(h, plot_impulse(h[,opt]) Plot impulse response
plot_scurve(ted) Plot the S-curve of a Timing Error Detector (TED)
plot_demod(dbg) Plot the intermediate results of a demodulation process (carrier and clock recovery, ...).
\end{verbatim}

3. Channel simulation

\begin{verbatim}
y = awgn(x, sigma[,opt='c'/'r']) Additive white gaussian noise (complex or real)
prim = chn_simu_prm() Default simulation parameters (create a structure)

y = chn_simu(x, prim) Proceed to channel simulation

chn = fading_chn_init('rcc', 'rayleigh', fd, fs, [k]) Create a fading channel

y = fading_chn_process(chn, x) Simulate the effect of a fading channel

y = iqi_simu(x, phi, g) Simulate I/Q imbalance effect
n = bnoise_power(bw, T) Compute thermal noise power, in dBm (bandwidth in Hz and temperature in °C)

[b1, b2, nerr, ber] = cmp_bits(b1, b2) Alignment of 2 bits vectors and counting the number of errors.
\end{verbatim}

4. Channel extraction

4.1 Single channel extraction

\begin{verbatim}
dn = downconvert_process(su[,opt='r'/'c']) Initialize a down-conversion object (conversion from IF to baseband), for real or complex signals (for real signal, an image filter is automatically created). $\nu$ is the normalized IF frequency.
\end{verbatim}

5. Clock recovery

\begin{verbatim}
cr = clock_rec_init(osf[,ted,itrp,tc]) Creates a clock recovery object, with given TED and interpolator. OSF is the input over-sampling factor. tc is the time constant of the loop.

[cr,y] = clock_rec_process(cr, x) Proceed to clock recovery. Output signal y is resampled at symbol frequency and synchronized with the detected clock.

dt = delay_estim(x0, x1) Utility function to estimate accurately (subsample precision) the delay between two signals

y = frac_delay(x, delay[,itrp]) Utility function to apply a fractional delay (subsample precision) to a signal

y = resample(x, R[,itrp]) Resampling a signal at a different frequency (R is the decimation / upsampling ratio). Using by default a cardinal spline interpolator.

itrp = itrp_init('lagrange'/'linear'/'c spline') Initialization of an interpolator object (Lagrange polynomial interpolation, linear interpolation or cardinal spline)

ted = ted_init([type='gardner'/'mm'/'el']) Creation of a Timing Error Detector (TED). Supports Garner, Mueller and Müller or Early-Late gate.

[y, u] = scurve(ted) Compute the S-curve of a Timing Error Detector (ted)
\end{verbatim}
6. Carrier recovery

Carrier recovery is a process that can be decomposed into: phase error detection (ped), phase and frequency error estimation through second order PLL, and carrier correction.

cr=carrier_rec_init(ped[, 1f]) Initialization of a carrier recovery loop object, with given phase error detector and loop filter.

[cr,z2]=carrier_rec_process(cr,z1) Proceed to carrier recovery.

1f=1f_init(1, tc) Creation of a first order loop filter (a first order loop enables to recover the phase of the carrier).

1f=1f_init(2, BL, eta) Creation of a second order loop filter (a second order loop enables to recover both the phase and the frequency of the carrier).

1f=1f_reset(1f, phi[, nu]) Reset the current phase (and optionally frequency) of a loop filter object.

ped=ped_init('psk'|'ask'|'costa'|'atan',M[,tc]) A ped (Phase Error Detector) is a unit that computes the current phase error between the incoming signal and the local carrier (output of a second order PLL). Supported loop types are the following: squaring loop, tan loop, and Costa loop.

7. Symbol generation

b = prbs(n) Pseudo-random binary sequence, n bits
b = ts01(n) Alternating header (01010101...), n bits

7.1 LFSR (Linear Feedback State Registers)

LFSR are useful to build BER testers (easy synchronization between receiver and transmitter).

lf=lf_init(deg) Initialize a LFSR receiver object. Polynomial degree can be from 1 to 17. The higher the degree, the higher the sequence length.

lf=lf_init(1f, phi[, nu]) Create a first order loop filter (a first order loop enables to recover the phase of the carrier).

lf=lf_reset(lf, phi[, nu]) Reset the current phase (and optionally frequency) of a loop filter object.

[cr,z2]=carrier_rec_process(cr,z1) Proceed to carrier recovery.

7.2 Symbol generation

b = prbs(n) Pseudo-random binary sequence, n bits
b = ts01(n) Alternating header (01010101...), n bits

8. Pulse shaping

x=symmap(b,k[,enc='b'|'p'|'q']) Maps a binary sequence (0 and 1) to a symbol sequence, using binary, phase (PSK) or quadrature (QAM) encoding

b=symdemap(b,k[,enc='b'|'p'|'q']) Demaps a symbol sequence to a binary sequence.

y=upsample(x,osf) Upsample the input signal by the factor osf

y=downsample(x,osf) Downsample the input signal by the factor osf

9. Theoretical limits

ber=berawgn(wf,ebno[,M[,opt]]) Computes Bit Error Rate (ber) for Additive White Gaussian Noise (AWGN) channel.

c=channel_capacity(snr,B) Computes theoretical maximum channel capacity, in bit/s.

10. Channel equalization

eq = equalizer_init(wf[,K,errf,struct,gain,N1,N2]) Creation of an equalizer object. The structure can be 'dde' (Decision Directed Equalization) or 'dfe' (Decision Feedback Equalization), and the error function can be 'lms' (Least Mean Square), or 'cma' (Constant Modulus Algorithm).

[x,y]=equalizer_process(eq,x) Equalization of an input signal

g=equalizer_zfe(h,n) Zero-forcing equalizer (ZFE) FIR filter computation

11. Miscellaneous functions

g=iir1_design(tp,val) Compute the coefficient of a single pole lowpass IIR filter. Parameters can be 'fc', cut-off frequency or 'tc', time constant.

[x,fs,fsymb] = sct_test_signal([opt]) Generation of an RF test signal (default is a 2 ms (2000 samples) BPSK signal, sampled at 1 MHz, with IF at 200 KHz and symbol frequency of 10 kbps).