Communication Toolbox for SCILAB

User guide

J. A.

Version 0.1, January 15, 2017

Toolbox webpage: http://www.tsdconseil.fr/log/sct
Contents

1 Introduction
2 Modulations
3 Plotting functions
4 Symbols generation
5 Channel simulation
6 Theoretical limits
7 Other modules (equalization, channelization, carrier & clock recovery)
8 Work in progress
<table>
<thead>
<tr>
<th>1</th>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Modulations</td>
</tr>
<tr>
<td>3</td>
<td>Plotting functions</td>
</tr>
<tr>
<td>4</td>
<td>Symbols generation</td>
</tr>
<tr>
<td>5</td>
<td>Channel simulation</td>
</tr>
<tr>
<td>6</td>
<td>Theoretical limits</td>
</tr>
<tr>
<td>7</td>
<td>Other modules (equalization, channelization, carrier &amp; clock recovery)</td>
</tr>
<tr>
<td>8</td>
<td>Work in progress</td>
</tr>
</tbody>
</table>
Introduction

Content and organization of this manual

- Description of main features
- Usage examples
- For complete API documentation: see the web page http://www.tsdconseil.fr/log/sct, or install the atom package, and look in the help (help sct in Scilab terminal).
- If you need help for understanding the underlying concepts, please have a look at the proposed training: Techniques in Software Defined Radio (SDR)
- If you need help for using Scilab, please have a look at the proposed Scilab trainings: SCILAB basics, and SCILAB advanced functions
Introduction

What is it?

- A **small** toolbox for telecommunications (only basic functions are provided, really cannot be compared to, e.g., the Matlab communication toolbox)
- 100 % Scilab code, no primitive / C / Fortran functions
- **Open source** (CeCILL V2.1 license)
  - Contributions welcome!
- **Goals / usage examples:**
  - Facilitation of prototyping modulators / demodulators for RF links
  - Analysing / plotting of data obtained from other programs
  - Generating data for real time implementation (e.g. taps of pulse shaping filters)
- **Some things it cannot do:**
  - Fast or real-time processing (yet, until SCILAB has, *at last*, a JIT / Just In Time compiler)
  - XCos simulation
Introduction

Toolbox sub-modules

- **modulations**: Classical waveforms (FSK, PSK, etc.), modulation and demodulation algorithms.

- **graphics**: Some plotting functions (for constellation / scatter plots, bit error rates, eye diagrams, ...).

- **sym-gen**: Several functions to generate binary sequences (alternating 010101, pseudo-random, LFSR)

- **simulation**: Propagation channels simulation (AWGN channel, fading, etc.)

- **limits**: Computing of several theoretical limits (channel capacity, theoretical ber, etc.)

- **pulse-shaping**: Pulse shaping filters (NRZ, SRRC, Gaussian, etc.), symbols mapping and demapping.

- **channelization**: Channelization (frequency multiplexing / demultiplexing) and baseband conversion.

- **equalization**: Channel equalization (for non flat channels).

- **clock-rec**: Clock recovery and interpolation functions (Lagrange, cardinal spline), signal resampling, ...

- **carrier-rec**: Carrier recovery (phase error detection, loop filters, ...)
Introduction

Coding style (1)

- In communications, there are a lot of different algorithms / modulations / techniques for doing the same thing. Each has its own advantages and drawbacks, but, usually, the same inputs and outputs.

- So some part of the library is written in a **object oriented style**, meaning that **different objects share the same interface**, for instance:
  - The different waveform types (BPSK, QAM, FSK, ...), the different interpolators (linear, Farrow, splines), the different timing error detectors (Gardner, ...), ...
  - For instance: \( \text{wf} = \text{wf\_init('qpsk')} \) and \( \text{wf} = \text{wf\_init('qam64')} \) will initialize two different waveform objects (actually, Scilab structures), but **the fields are common** and **the same functions can be called on them** (plot\_ber, plot\_const, demod\_init, ...).
Introduction
Coding style (2)

Usually, for a given functionnality (let's say it's called `func`), there are two functions:

```plaintext
func = func_init(...)  
```

**Initialization** of an object of type `func`

```plaintext
[func, output] = func_process(func, input)
```

**Processing** of some data input (typically a vector of samples), computing some **output** (usually a vector too), and **update** of the object state (`func`).

For instance:
- `mod_init`, and `mod_process` for signal modulation,
- `demod_init`, and `demod_process` for signal demodulation,
- `fading_chn_init` and `fading_chn_process` for fading channel simulation
- etc.
Contents

1 Introduction
2 Modulations
3 Plotting functions
4 Symbols generation
5 Channel simulation
6 Theoretical limits
7 Other modules (equalization, channelization, carrier & clock recovery)
8 Work in progress
Modulations

Functions list

This is the main module of the toolbox. It enables to **modulate** and **demodulate** a binary stream to / from a RF / IF signal, including the classical steps of **downconversion** (and image rejection), **carrier and clock recovery**, **matched filter**, **symbol mapping** / **demapping**, . . .

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wf=wf_init(...)</code></td>
<td>Definition of a waveform</td>
</tr>
<tr>
<td><code>wf=wf_set_filter(wf,...)</code></td>
<td>Specification of a pulse shaping filter</td>
</tr>
<tr>
<td><code>mod=mod_init(wf,fs,fi,fsb)</code></td>
<td>Creation of a modulator object</td>
</tr>
<tr>
<td><code>[mod,x]=mod_process(mod,b)</code></td>
<td>Modulation</td>
</tr>
<tr>
<td><code>dmd=demod_init(wf,fs,fi,fsb)</code></td>
<td>Creation of a demodulator object</td>
</tr>
<tr>
<td><code>[dmd,b]=demod_process(dmd,x)</code></td>
<td>Demodulation</td>
</tr>
</tbody>
</table>
Modulations

Waveform definition

\[
wf = \text{wf\_init}('\text{mod\_name}',[],\text{params})
\]

Returns a **waveform object**, completely defining the modulation used. This waveform object can then be used as a parameter for a lot functions of the SCT (modulation / demodulation, ber computation, constellation plot, ...).

**Example**

\[
wf = \text{wf\_init}('\text{psk}',8);
\text{plot\_const}(wf);
\]
Modulations

Waveform definition (linear modulations)

**Amplitude modulations**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wf_init('ask',M)</code></td>
<td>Amplitude Shift Keying.</td>
</tr>
<tr>
<td><code>wf_init('ook')</code></td>
<td>On-Off Keying</td>
</tr>
</tbody>
</table>

**Phase modulations**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wf_init('psk',M)</code></td>
<td>General Phase Shift Keying, M phases.</td>
</tr>
<tr>
<td><code>wf_init('bpsk')</code></td>
<td>Binary Phase Shift Keying</td>
</tr>
<tr>
<td><code>wf_init('qpsk')</code></td>
<td>Quadrature Phase Shift Keying</td>
</tr>
</tbody>
</table>

**Phase / amplitude modulations**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wf_init('qamxxx')</code></td>
<td>Quadrature Amplitude Shift Keying</td>
</tr>
</tbody>
</table>
## Modulations

Waveform definition (frequency modulations)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wf_init('fsk', ind, filt, BT)</code></td>
<td>General Frequency Shift Keying. <code>ind</code> is the modulation index, <code>filt</code> is the pulse shaping filter ('nrz' or 'gaussian'), <code>BT</code> is the bandwidth-time product.</td>
</tr>
<tr>
<td><code>wf_init('gfsk', ind, BT)</code></td>
<td>Same as 'fsk', but with gaussian filter implied.</td>
</tr>
<tr>
<td><code>wf_init('msk', filt, BT)</code></td>
<td>Minimum Shift Keying</td>
</tr>
<tr>
<td><code>wf_init('gmsk', BT)</code></td>
<td>Gaussian Minimum Shift Keying</td>
</tr>
</tbody>
</table>
Modulations

Waveform definition (pulse shaping filter)

wf = wf_set_filter('type', params)

Change the pulse shaping filter of the waveform (default is NRZ).
- 'none' No pulse shaping (e.g. dirac directly transmitted!)
- 'nrz' Moving average filter (rectangular pulses)
- 'rc', ntaps, r Raised cosine, with specified roll-off factor
- 'srrc', ntaps, r Square root raised cosine, with specified roll-off factor
- 'gaussian', ntaps, BT Gaussian filter (applied to NRZ). BT is the bandwidth-time product.

Example

wf = wf_init('qpsk');
wf = wf_set_filter(wf, 'srrc', 50, 0.2);
mod = mod_init(wf, 10, 0, 1);
[mod, x] = mod_process(mod, prbs(1000));
clf(); plot_const(x);
Modulations

Modulation of a binary sequence to RF / IF

```plaintext
mod=mod_init(wf,fs,fi,fsymb[,mode])
[mod,x]=mod_process(mod,b)
```

Instanciation of modulator object (`mod_init`) and modulation of a bitstream (`mod_process`).

- $f_s$ is **sample frequency**, $f_i$ is **intermediate frequency** and $f_{symb}$ is **symbol frequency**.
- `mode` can be ’r’ (real modulation) or ’c’ (complex / I/Q modulation).

**Example**

```plaintext
// Creation of a MSK modulator,
// fs=0.5 MHz, fi=100 kHz, fsymb=20 kHz
mod = mod_init(’msk’,0.5e6,100e3,20e3);

// Modulation
[mod,x] = mod_process(mod,prbs(2000));

// Plots power spectral density
clf(); plot_psd(x);
```
Modulations

Demodulation of a RF / IF signal

demod=demod_init(wf,fs,fi,fsymb[,mode])
[demod,x]=demod_process(demod,b)

Instanciation of demodulator object (demod_init) and demodulation of a rf / if signal (demod_process).

Example

// Generation of a test signal
// b = bitstream, x = BPSK modulation
[x,fs,fi,fsymb,b] = sct_test_signal();

// Demodulation
demod = demod_init(’bpsk’,fs,fi,fsymb);
[demod,b2] = demod_process(demod,x);

// Plots correct bit stream
subplot(211); plot_binary(b);
// Plots demodulated bit stream
subplot(212); plot_binary(b2);
Contents

1 Introduction
2 Modulations
3 Plotting functions
4 Symbols generation
5 Channel simulation
6 Theoretical limits
7 Other modules (equalization, channelization, carrier & clock recovery)
8 Work in progress
These are some plotting functions specific to the telecommunications domain.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plot_eye(x,T)</td>
<td>Eye diagram</td>
</tr>
<tr>
<td>plot_const(x)</td>
<td>Constellation plot</td>
</tr>
<tr>
<td>plot_const(mod)</td>
<td>Theoretical constellation plot, for a given modulation</td>
</tr>
<tr>
<td>plot_ber(mod[,opt])</td>
<td>Plots theoretical bit error rate</td>
</tr>
<tr>
<td>plot_ber(x,ber[,opt])</td>
<td>Plots empirical bit error rate</td>
</tr>
<tr>
<td>plot_scurve(ted)</td>
<td>Plots the S-Curve of a timing error detector</td>
</tr>
<tr>
<td>plot_psd(x[,fs])</td>
<td>Plots power spectral density</td>
</tr>
</tbody>
</table>
Plotting functions
Eye diagram

plot_eye(x,T)

The eye diagram is the superposition in the time domain of the several chunks of the input signal, with a trigger. At the center of the diagram is the ideal sampling time, and with this view we can see how well the different symbol values are resolvable (the 'opening' of the eye).

$T$ is the symbol period.

Example

```scilab
osf = 128; // Oversampling factor/nsymb = 100; // Number of symbols
x = nrz(ts01(nsymb),osf);
x = ma(x, osf); // Moving average
x = awgn(x, 0.1); // Add some noise
plot_eye(x, osf);
```
Plotting functions

Scatter plot

plot_const(x[,opt])

Lissajou plot of a complex signal (also called I/Q, or quadrature signal), e.g. plots:

\[ \text{Im}(x) = f(\text{Re}(x)) \]

Example

```
mod = mod_init('psk',8); // 8PSK modulator
b = prbs(1e6); // 1 Mbits (random seq.)
x = mod_process(mod,b); // 8PSK modulation
x = awgn(x,0.1); // White noise
plot_const(x);
```
Plotting functions

Constellation plot

**plot Const**(wf)

Theoretical constellation plot of a waveform object.

**Example**

```
wf = wf_init('qam16');
clf();
plot_const(wf);
```
Plotting functions

Bit error rate plot

```scilab
plot_ber(EbNo,ber[,format]) // Empirical BER
plot_ber(wf[,format]) // Theoretical BER
```

Plots bit error rate, either theoretic (first form), or empirical (second form).

- `format` is an optional argument that will be given to the `plot` function.
- The `EbNo` must be in dB.
- `wf` is either a waveform object or a string

Example

```scilab
clf();
plot_ber('bpsk', 'b');
plot_ber('fsk', 'r+');
legend([’bpsk’,’fsk’]);
```
Plotting functions
Power spectral density

\texttt{plot\_psd(x[,fs])}

Plots the PSD of $x$, e.g. $10 \log_{10} \|X\|^2$, with $X = \text{DFT}(x)$. The PSD is evaluated by a single DFT of $x$ windowed with Hann window. If input is a real signal, plots only the positive frequencies (PSD for the negative frequencies are identical).

This function doesn't do much more than calling \texttt{fft}, \texttt{ffshift}, \texttt{abs}, \texttt{log10}, \texttt{window}, \texttt{plot}..., and is not configurable, but it is convenient when a simple psd view is needed. For more advanced PSD evaluation, see Scilab functions \texttt{cspect}, \texttt{pspect}, ...

\textbf{Example}

\begin{verbatim}
fs = 48e3;  // Sample rate = 48 KHz
f = 12e3;  // Signal freq. = 12 kHz
\texttt{t = linspace}(0,1,fs);  // one second @48KHz
x = \texttt{sin}(2*\texttt{\%pi}*f*t);  // Pure real tone
\texttt{plot\_psd(x,fs);}  
\end{verbatim}
1 Introduction
2 Modulations
3 Plotting functions
4 Symbols generation
5 Channel simulation
6 Theoretical limits
7 Other modules (equalization, channelization, carrier & clock recovery)
8 Work in progress
Symbols generation

Basic symbol sequences

\[ b = \text{prbs}(n) \]

Generates a Pseudo-Random Binary Sequence of \( n \) bits
(011000110101...)

\[ b = \text{ts01}(n) \]

Generates an alternating binary sequence of \( n \) bits (0101010101...)

User guide
SCILAB communication toolbox

Introduction
Modulations
Plotting functions
Symbols generation
Channel simulation
Theoretical limits
Other modules (equalization, channelization, carrier & clock recovery)
Work in progress
Symbols generation

Linear Feedback State Registers (LFSR)

LFSR are useful for building BER testers (automatic synchronization between receiver and transmitter: **the receiver can lock anywhere on the received bitstream**).

```plaintext
lfsr=lfsr_init(deg)
[b,lfsr]=lfsr_process(lfsr,n)
```

**LFSR transmitter functions**: creation (**lfsr_init**) and symbol generation (**lfsr_process**). `deg` is the polynomial degree (register width), `b` is the generated bitstream.

```plaintext
lfsr=lfsr_rx_init(deg)
lfsr=lfsr_rx_process(lfsr,b)
[locked,ber] = lfsr_rx_status(lfsr)
```

**LFSR receiver functions**. When the receiver is locked, the BER is continuously updated.
Introduction

Modulations

Plotting functions

Symbols generation

Channel simulation

Theoretical limits

Other modules (equalization, channelization, carrier & clock recovery)

Work in progress
Channel simulation

Goals:

- **Simulate some radio channel imperfections**
  (multipath, thermal noise, interferers)
- **Simulate some analog path imperfections**
  (DC offset and I/Q mismatch for quadrature mixers)
# Channel simulation

## Functions list

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>y = awgn(x, sigma)</code></td>
<td>Additive white gaussian noise</td>
</tr>
<tr>
<td><code>y = fading_chn_init(type, fd, fs[, ...])</code></td>
<td>Rayleigh and Rice channel models (narrowband systems)</td>
</tr>
<tr>
<td><code>y = fading_chn_process(chn, x)</code></td>
<td></td>
</tr>
<tr>
<td><code>y = iqi_simu(x, phi, g)</code></td>
<td>I/Q Imbalance simulation</td>
</tr>
<tr>
<td><code>y = chn_simu(x, prm)</code></td>
<td>Channel simulation (AWGN, clock and carrier errors, ...)</td>
</tr>
</tbody>
</table>

---

**User guide**

**SCILAB communication toolbox**

**Introduction**

**Modulations**

**Plotting functions**

**Symbols generation**

**Channel simulation**

**Theoretical limits**

**Other modules (equalization, channelization, carrier & clock recovery)**

**Work in progress**

---

[www.tsdconseil.fr](http://www.tsdconseil.fr)
Channel simulation

Additive white gaussian noise (AWGN)

```plaintext
y = awgn(x,sigma[,opt='r'|'c'])
```

For real noise (opt='r'):

\[ y = x + n, \text{ with } n : N(0, \sigma) \]

For complex noise (opt='c'):

\[ y = x + n_1 + n_2 \cdot i, \text{ with } n_1, n_2 : N(0, \sigma) \]

Example

```plaintext
// 100 random NRZ symbols
// (Tsymb = 10 samples)
x = nrz(prbs(100),10);
// Add some noise
y = awgn(x,0.1);
clf(); plot(y);
```
Channel simulation
Rayleigh and Rice channel

chn = fading_chn_init(’rayleigh’, fd, fs)
chn = fading_chn_init(’rice’, fd, fs, K)

Creation of a channel object, for narrowband systems, without (Rayleigh) or with (Rice) dominant line of sight, \( f_d = \) maximum Doppler shift, \( f_s = \) sampling frequency, \( K = \frac{\beta}{\sigma^2} \) (ratio of dominant path energy vs diffuse paths). \( K = \infty \Rightarrow \) Rayleigh channel, \( K = 0 \Rightarrow \) single path channel

y = fading_chn_process(chn, x)

Apply the time-varying fading and phase shift to the input signal \( x \).

Example

\[
x = \text{ones}(100,1);
chn = \text{fading\_chn\_init}(’r’,1,10);
y = \text{fading\_chn\_process}(chn, x);
\]

\[
\text{subplot}(1,2,1); \text{plot(abs(y))} \text{ subplot}(1,2,2); \text{histplot(abs(y))};
\]
Channel simulation
I/Q imbalance

I/Q imbalance typically occurs in analog quadrature mixers (zero or low FI architectures).

\[
y = iqi\_simu(x, \phi, g)
\]

With \(x\): reference signal, \(\phi\): phase imbalance (in radians), \(g\): relative gain imbalance (1 = balanced).

\[y = \alpha x + \beta x^*,\] with \(\alpha\) and \(\beta\) computed from the imbalance parameters.

Note: \(iqi\_blind\_est\) and \(iqi\_cor\) to correct for I/Q imbalance.
Contents

1 Introduction
2 Modulations
3 Plotting functions
4 Symbols generation
5 Channel simulation
6 Theoretical limits
7 Other modules (equalization, channelization, carrier & clock recovery)
8 Work in progress
Theoretical limits
Channel capacity

\[ C = \text{channel\_capacity}(\text{SNR\_dB}) \]

Computes capacity of a AWGN channel (Shannon-Hartley):

\[ C = \log_2 \left( 1 + \frac{S}{N} \right) = \log_2 \left( 1 + \frac{RE_b}{BN_o} \right) \]

with \( S/N = \) signal to noise power ratio, \( B = \) channel bandwidth, \( R = \) bit rate, \( C = \) channel capacity, in bit/s/Hz.

Example

```matlab
--> SNR = -10:.1:20;
--> C = channel_capacity(SNR);
--> plot(C);
--> C = channel_capacity(10)
3.46
```

Means that maximum transmission bandwidth is 3.46 bits/s/Hz for a SNR of 10 dB (with ECC).
Theoretical limits

Bit error rate VS EbNo (for a given modulation)

ber = berawgn(wf, EbNo)

Computes theoretical uncoded bit error rate (ber), for a given waveform wf and a given SNR per bit EbNo (dB).

Example

SNR_db = 3:16;
wf = wf_init('bpsk');
ber = berawgn(wf, SNR_db);
plot(SNR_db, ber);
Contents

1 Introduction
2 Modulations
3 Plotting functions
4 Symbols generation
5 Channel simulation
6 Theoretical limits
7 Other modules (equalization, channelization, carrier & clock recovery)
8 Work in progress
Other modules

- **Channel equalization**:
  - LMS or CMA error functions,
  - DDE or DFE structures.

- **Channelization**:
  - Baseband conversion and image filtering
  - Polyphase structures (frequency multiplexing / demultiplexing)

- **Carrier recovery**:
  - Carrier recovery loop, with first or second order loop filter, phase error detectors (Squaring loop, Costa loop, Tan loop, ...)

- **Clock recovery**:
  - Clock recovery loop, with interpolators (linear, Lagrange, cardinal spline), timing error detectors (Gardner, M&M, early-late gate), loop filter (1st order). General resampling and interpolating functions.
Work in progress

- **OFDM** signal simulation and decoding
- **Coherent FSK demodulation** (based on Viterbi decoder on the baseband signal path)