Optilux Toolbox

P. Serena, M. Bertolini, A. Vannucci

University of Parma, Italy

Department of Information Engineering

March 2009

Contents

Chapter 1

Getting Started

In this Chapter we introduce the Optilux toolbox and its main features.

- What Is Optilux Toolbox?: Introduces the toolbox and describes the types of problems it is designed to solve.
- GNU General Public License: Describes the license that comes with Optilux.
- Style rules: Describes the basic style rules of this toolbox.
- Optilux structure: The core of Optilux.
- Description: How Optilux work with signals.
- Glossary: A glossary of the main terms.

1.1 What is Optilux?

1.1.1 Introduction

Optilux is an open source collection of tools that provide advanced techniques to design, simulate, and analyze optical communication systems. Optilux is implemented as a Matlab/Octave toolbox and efficiently exploits the MEX interface to speed up computation.
The toolbox includes routines for describing many aspects of optical systems, including:

- Bit pattern generators
- Multi-level modulation formats
- Optical fibers in the nonlinear regime
- Karhunen-Loéve methods for performance evaluation
- Monte Carlo estimation
- Performance optimization
- Polarizations effects

1.1.2 Philosophy

Optilux is free software. This means that everyone is free to use it and free to redistribute it. The precise conditions can be found in the GNU General Public License, version 3, that comes with Optilux. A crucial aspect of free software is that users are free to cooperate. It is absolutely essential to permit users who wish to help each other to share their bug fixes and improvements with other users. Hence, everyone is invited to contribute to Optilux.

1.1.3 Functions

All the toolbox functions are M-files that implement specialized Optilux algorithms. Following the flow of the information into an optical system, the functions are organized in the following categories:

- Transmitter blocks (e.g. laser, pattern generation, modulators)
- Channel blocks (e.g. fiber, optical amplifiers)
- Receiver blocks (e.g. eye evaluation, bit-error rate measurements)

A detailed list of the functions can be found here.
Optilux works both under Octave and Matlab. Under Octave some functions require packages of octave-forge.

1.2 GNU General Public License

Optilux is released under the GNU General Public License, version 3, or GNU GPLv3 for short. A FAQ list about GPL can be found at http://www.gnu.org/licenses/gpl-faq.html. With a GNU GPLv3 you are free to make modifications to any function of Optilux and use them privately, without ever releasing them. But if you release the modified version to the public in some way, the GPLv3 requires you to make the modified source code available to the program's users, under the GPLv3.
Each m-file of Optilux contains the following body of the license:

```
% This file is part of Optilux, the optical simulator toolbox.
% Copyright (C) 2009 Paolo Serena, <serena@tlc.unipr.it>
%
% Optilux is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation; either version 3 of the License, or
% (at your option) any later version.
%
% Optilux is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

1.3 Style rules

A good software is first of all clear to the reader. To this aim Optilux follows some general style rules that hopefully should help the readability of the code. To support the spirit of the free cooperation which is at the base of this open source project, any user that wants to share his functions with the Optilux community is encouraged to follow the rules:

1. All the global variables characterizing the system must be fields of a unique struct-global variable called `GSTATE`. The fields must be upper case. Please, do not add other fields to existing ones if not strongly motivated.
2. Each function is followed by its help description, which follows the Matlab rules: i) the first line is a brief description of the function, ii) all input/output variables in the help must be upper case, iii) at the end of the help a list of reference functions is present, iv) all lines except the first begins with a tab, v) each possible syntax of the function call is described separately, etc.
3. All functions must work both under Matlab and Octave.

4. All primary functions that run non-trivial operations must write a brief summary into a file that must be called `simul_out` and must be place in `GSTATE.DIR`.

5. If a function updates a field of `GSTATE` that is not the electric field, please note it in the summary file `simul_out`.

6. Please, if possible, call the integer variables by starting them with `i,j,k,l,m,n`. E.g. use `nfield = 5` instead of `field = 5`.

7. All variables must have English names.

8. If a new function requires a long list of arguments, please collect them into a struct variable. The fields of the struct must be lower case.

9. Please, insert at the beginning of each function the main test conditions on the input variables to avoid common errors. E.g. if the input variable `x` is a probability check that satisfies `0 <= x <= 1`.

10. It is very kind from you if you write a `.m` example file to show your new functions.

11. When you add a new function that is stable, do the following: i) check its efficiency, for instance using the Matlab profiler; ii) update the file `Contents.m`; iii) let your function available to all other people.

1.4 Optilux Structure

1.4.1 The code

Optilux is a collection of .m files, each representing a specific block of an optical system. Hence programming in Optilux follows standard rules where the top-to-bottom flow on a .m file corresponds to moving over the distance of the optical system. For the sake of easiness, the most important and used variables have been collected into a global one, called `GSTATE`. `GSTATE` is a struct variable and must be declared as global by each function that shares it.

Note: Remember that both `GSTATE` and its fields have capital letters.

1.4.2 The global GSTATE

`GSTATE` is a struct variable with the following fields:

Argument	**Description**	**Initialized by function**
`NSYMB`	The number of symbols of the digital signal under transmission. For binary transmissions this is also the number of bits.	reset_all
`NT`	The number of discrete points x symbol.	reset_all
`NCH`	The number of channels for a wavelength division multiplexing (WDM) transmission.	reset_all
`FN`	The vector of frequencies, normalized to the symbol rate. See here for more details.	reset_all
`SYMBOLRATE`	The signal symbol rate in [Gsymbols/s]. For a binary transmission this is also the bit rate [Gbit/s].	electricsource
`DISP`	The running cumulated dispersion [ps/nm] along the system.	create_field
`FIELDX`	The X component of the electric field. In absence of polarization effects the electric field is assumed polarized on the X component.	create_field
`FIELDY`	The Y component of the electric field. Unlike `GSTATE.FIELDX`, the Y component may not exists leading to an empty field for `GSTATE.FIELDY`.	create_field
`FIELDX_TX`	A copy of `GSTATE.FIELDX`. Useful for back-to-back measurements.	create_field
`FIELDY_TX`	A copy of `GSTATE.FIELDY`. Useful for back-to-back measurements.	create_field
`DELAY`	The running cumulated delay along the optical line by the X and Y polarizations. The delay is normalized to the inverse of the symbol rate, i.e. 1 is one symbol time. Note: by default Optilux bases its measurements at the receiver by estimating the delay through a correlation method instead of using this theoretical delay.	create_field
`LAMBDA`	The WDM channel wavelengths [nm].	lasersource
`POWER`	The transmitted peak power of the WDM channels [mW].	lasersource
`PRINT`	A flag true or false for printing functions summary to file.	reset_all
`DIR`	The output directory. Such option exists only with `GSTATE.PRINT=true`.	reset_all

1.4.3 MeX files

Optilux makes use of mex files to speed up the simulation. To reach a trade off between speed and code reliability, only simple, but time consuming, portions of codes are compiled as mex files. So far the following functions are implemented in mex files:

fastexp	`fastexp(x)` is `exp(i*x)`
saddle	saddle point search
cmaadaptivefilter	Continuous modulus algorithm
easiadaptivefilter	EASI algorithm

A mex file can improve the speed of the code significantly. For instance, observe the following line codes:

```
x=rand(1,1e5);
tic;
for k=1:1e3
    y=fastexp(x);
end
toc
>> Elapsed time is 8.187043 seconds.

x=rand(1,1e5);
tic;
for k=1:1e3
    y=exp(i*x);
end
toc
>> Elapsed time is 13.872832 seconds.

x=rand(1,1e5);
tic;
for k=1:1e3
    y=complex(cos(x),sin(x));
end
toc
>> Elapsed time is 12.031231 seconds.
```

All pieces of code give the same result, but the computational gain of `fastexp` is significant. Mex files must be compilated before using it. A simple way to compile all files once is to use comp_mex in your Optilux directory.

1.4.3.1 Notes

The compilation of `.c` files under Matlab^{TM}can fail because Matlab is unable to find the correct c++-standard-library. In this case edit your `mexopts.sh` file (you can create it in `~/.matlabxx/mexopts.sh` by running `mex -setup` on the command line) and replace the line:
`CLIBS="$RPATH $MLIBS -lm -lstdc++"`
by, (e.g. if you have Matlab in /opt/matlab)
`CLIBS="$RPATH $MLIBS -lm /opt/matlab/sys/os/glnx86/libstdc++.so.6"`
i.e. teach Matlab to use its stdc library.

Another solution under Debian-based Linux systems is to run the following:
`apt-get install build-essential`
This package contains an informational list of packages which are considered essential for building Debian packages.

1.5 Signals description

Aim of this Section is to clarify the relation between an analog signal and its discrete version, with particular emphasis on the numerical details for describing a signal both in the time or frequency domain.

1.5.1 Time domain representation

Optilux works with discrete time signals. A discrete time signal is an indexed sequence of real or complex numbers. A widely used signal in Optilux is the discrete version of a digital modulation signal, i.e. a sequence of information symbols emitted at rate R [symbols/s]. R plays a fundamental role for a digital signal and hence in Optilux is saved into the global variable `GSTATE.SYMBOLRATE` in [Gsymbols/s]. A digital modulation signal in Optilux is a vector containing `NSYMB` symbols, where each symbol is described by `NT` points. The length of such a vector is therefore `NFFT=NSYMB*NT` points. Remember that, since Optilux makes use of the Fast Fourier Transform algorithm, such signals are intrinsically periodic of period `NFFT`. Both variables `NT` and `NSYMB` are saved into the global `GSTATE.NT` and `GSTATE.NSYMB`, respectively.

Some notes about the time:

- A time signal in Optilux is an indexed vector. The distance between two indexes, i.e. `1`, corresponds to a time interval of:
  `1/(GSTATE.SYMBOLRATE*GSTATE.NT*1e9) [s]`
  or
  `1/GSTATE.NT [symbols].`
- Since Optilux works with circularly periodic signals, the concept of starting time or clock `0` is arbitrary. For convention, when plotting signals vs. the time variable Optilux associates the first index to clock `0`. In such cases we use the following time vector:
  `time=0:1/GSTATE.NT:GSTATE.NSYMB-1/GSTATE.NT;`
- A signal function of time is of length `NFFT`.
- Optilux works with FFTs, hence we strongly recommend to use signals of length `NFFT` factorisable in small integers. For instance working with signals of length a power of 2 is the best (fastest) solution, while working with signals of length equal to a to a large prime number is the worst or slowest solution.

1.5.2 Frequency domain representation

The frequency, i.e. the fundamental variable of the Fourier domain, is contained in the global variable `GSTATE` as `GSTATE.FN`. Given a signal $x(t)$, being t the time, the Fourier transform $\widetilde{X}(f)$, being f the frequency (`GSTATE.FN`), is defined as:

$$\widetilde{X}(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt$$

`GSTATE.FN` in Optilux is the following vector (see reset_all):

```
GSTATE.FN=FFTSHIFT(-NT/2:1/NSYMB:NT/2-1/NSYMB);
```

being `NT=GSTATE.NT` and `NSYMB=GSTATE.NSYMB`.

Some notes about the frequency:

- The frequency is normalized to `GSTATE.SYMBOLRATE`. Hence, the frequency in Optilux is dimensionless. It turns out that all filter bandwidths in Optilux are normalized as well. For instance, an optical filter with bandwidth 20 GHz applied to a system working with a symbol rate of 10 Gbs in Optilux has bandwidth B= 20 GHz/10 Gbs = 2. Clearly, such a policy is just a question of style.

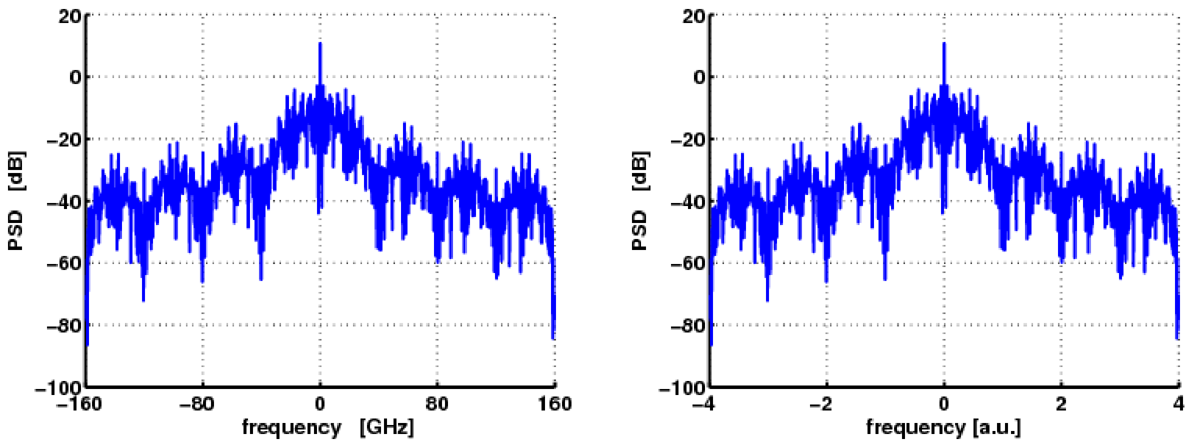

Figure 1.1: Left: PSD vs. absolute frequency [GHz]. Right: The same PSD vs. normalized frequency [a.u.]. Symbol rate: 40 Gsymb/s.

- The lowest discrete frequency (resolution) is `1/GSTATE.NSYMB`. The largest discrete frequency (Nyquist frequency), in absolute value, is `GSTATE.NT/2`.
- `GSTATE.FN` is of length `NFFT`. Note that the definition of `GSTATE.FN` yields the frequency zero at index `1, NFFT/2` negative frequencies and `NFFT/2-1` positive frequencies.
- The presence of `FFTSHIFT` in the frequency definition allows to easily generate spectra in FFT notation.
- The non-normalized frequency in [1/s] is `GSTATE.FN*GSTATE.SYMBOLRATE*1e9`.
- Any time signal is intrinsically periodic in time and in frequency with period `NFFT`.
- See here for more details about the choice of `GSTATE.NT.`

The following Fig. 1.1 shows the power spectral density (PSD) of a typical on-off keying (OOK) signal modulated with symbol rate 40 Gbit/s. The left figure shows the frequency in absolute units, the right one with the frequency used by Optilux.

1.5.3 Electric field

Any optical signal, like the electric field used by Optilux, is a bandpass signal, i.e. its frequency spectrum has energy concentrated around an high frequency sinusoidal carrier. Generally speaking, the Fourier transform of the generic optical field $\mathbf{A_{bp}}(t)$ has zero or negligible energy for frequencies f satisfying $|f| < f_c - B$ and $|f| > f_c + B$. The carrier frequency f_c is usually expressed in terms of the central wavelength $\lambda_c = c/f_c$, being c the speed of light. Since usually the bandwidth B is much smaller than f_c, it is customary to express $\mathbf{A_{bp}}(t)$ in the following way:

$$\mathbf{A_{bp}}(t) = \mathrm{Re}\left\{\mathbf{A_{lp}}(t)\exp\left(j2\pi f_c t\right)\right\} \tag{1.1}$$

which emphasizes the role of the carrier frequency. $\mathbf{A_{lp}}(t)$ in (1.1) is a low-pass signal of bandwidth B, and is the one on which actually Optilux works. $\mathbf{A_{lp}}(t)$ in the general case is a 2x1 vector $\mathbf{A_{lp}}(t) = [A_x(t); A_y(t)]$ in a reference system whose main axes are usually called x and y, respectively. In absence of polarization effects $\mathbf{A_{lp}}$ is a scalar function satisfying $A_{lp}(t) \equiv A_x(t)$, i.e. Optilux assumes the field aligned with the x axis. The discrete version of $\mathbf{A_{lp}}(t)$ is saved in Optilux into the global variable `GSTATE.FIELDX` for A_x and into `GSTATE.FIELDY` for A_y. Both fields are of size `[NFFT,GSTATE.NCH]` or `[NFFT,1]`, depending on the propagation type (separate fields or unique field, see create_field). The number of rows is the number of discrete samples `NFFT=GSTATE.NT*GSTATE.NSYMB`.

`GSTATE.FIELDX` and `GSTATE.FIELDY` are initialized by reset_all to empty and created by create_field. `GSTATE.FIELDX` is always not empty after create_field, while `GSTATE.FIELDY` remains empty in absence of polarization effects.

Note: Remember that longer signals in the time domain (`GSTATE.NSYMB` $\gg$1) have greater resolution in the frequency domain, i.e. neighboring discrete frequencies are closer. On the contrary, by increasing the Nyquist frequency (`GSTATE.NT`) it is possible to increase the resolution in the time domain, i.e. the number of points x symbol.

1.6 Glossary

A list of acronyms:

ASE	Amplified spontaneous emission noise
BPSK	Binary-PSK
C-NLSE	Coupled-NLSE
DBS	De Bruijn sequence
DCF	Dispersion compensating fiber
DOP	Degree of polarization
DPSK	Differential-PSK
DQPSK	Differential quadrature phase shift keying
DSP	Digital signal processing
DWDM	Dense-WDM
ECP	Eye closure penalty
FWM	Four wave mixing
GVD	Group velocity dispersion
KL	Karhunen Loéve
LPF	Low pass filter
MC	Monte Carlo
MZ	Mach Zehnder
NF-DPSK	Narrow-filter DPSK
NL	Non linear
NLSE	Nonlinear Schrödinger equation
NRZ	Non-return to zero
OBPF	Optical band pass filter
OOK	On-off keying
OSNR	Optical signal to noise ratio
PDM	Polarization division multiplexing

P-DPSK	Partial DPSK
PMD	Polarization mode dispersion
PMF	Polarization maintaining fiber
PRBS	Pseudo random binary sequence
PRQS	Pseudo random quaternary sequence
PSBT	Phase-shaped binary transmission
PSK	Phase shift keying
QPSK	Quadrature-PSK
RZ	Return-to-zero
SMF	Single mode fiber
SOP	State of polarization
SP	Sensitivity penalty
SPM	Self phase modulation
SSFM	Split step Fourier method
WDM	Wavelength division multiplexing
XPM	Cross phase modulation

Chapter 2

List of Functions

This chapter shows the main functions of Optilux.

2.1 Functions - by Category

Fundamental functions	Mandatory blocks
Pattern and Coding	Pattern generation and elaboration
Transmitter Side	Components before propagation in the channel
Channel side	Channel components, including optical fibers
Receiver Side	Components after propagation in the channel
Utility functions	Utility functions
MeX and MeX support functions	Functions compiled with MeX

2.1.1 Fundamental functions

reset_all	Reset all global variables and initialize the simulation.
create_field	Create the electric field.

2.1.2 Pattern and coding

pattern	Create the sequence pattern with rules.
pat_encoder	Symbols encoder.
pat_decoder	Symbols decoder.
pat2stars	Convert an M-ary pattern into a complex constellation.
samp2pat	Convert received samples into a pattern.
stars2pat	Convert a complex constellation into a pattern.

2.1.3 Transmitter side

electricsource	Create the electric modulating signal.
lasersource	Multichannel laser transmitter.
linear_modulator	Modulate the optical field with a linear modulator.
mz_modulator	Modulate the optical field with a Mach-Zehnder Interferometer.
phase_modulator	Modulate the optical field with a phase modulator.
qi_modulator	Modulate the optical field using a QI Mach-Zehnder modulator.

2.1.4 Channel side

fiber	Optical fiber in the nonlinear regime.
fibergui	Optical fiber in the nonlinear regime (GUI tool).
ampliflat	Ideal Optical amplifier with ASE noise.
inverse_pmd	Inverse PMD matrix.
optfilter	Optical filter.

2.1.5 Receiver side

receiver_ook	Complete OOK receiver (POST fiber+OBPF+photodiode+LPF).
receiver_dpsk	Complete DPSK receiver (POST fiber+OBPF+MZ+LPF).
receiver_dqpsk	Complete DQPSK receiver (POST fiber+OBPF+MZs+LPF).
receiver_cohmix	Complete coherent mixer receiver.
eval_eye	Evaluate the eye opening for a non-coherent transmission.
ber_kl	Evaluate the ber for a non-coherent receiver by Karhunen-Loève method.
best_eye	Search algorithm for the best eye opening.
best_sp	Search algorithm for the best OSNR penalty vs. back-to-back.
ber_estimate	Bit-error rate estimate by Monte Carlo simulation.
mc_estimate	Monte Carlo estimation of a random variable mean and variance.
cmaadaptivefilter	Polarization demultiplexing filter using CMA algorithm.
dsp4cohdec	Digital signal processing for a coherent receiver.
easiadaptivefilter	Source separation filter using EASI algorithm.

2.1.6 Utility functions

nmod	N-modulus of an integer.
fastshift	Fast but simplified circular shift.
pow2phi	Convert power into nonlinear phase.
phi2pow	Convert nonlinear phase into power.
avg_power	Evaluate the average energy per symbol.
corr_delay	System delay by cross-correlation measurement
evaldelay	Evaluate the group-delay of the filter.
myfilter	Filter device in the frequency domain.
lpfilter	Filtering with a lowpass filter.
pol_scrambler	Rotates the SOP of signal samples on the Poincaré sphere.
dop_meter	Compute the Degree of Polarization of the optical field.
set_sop	Set the average State Of Polarization of the transmitted signal.
polarizer	Linear optical polarizer.
plotfield	Plot the optical field.
plotfile	Plot file from disk.
printfield	Print the optical field to file.
ber2q	Convert the bit-error rate in Q-factor.
mdoc	Display Optilux HTML documentation in the browser.
fprintmsg	Write a message into the file simul_out.
checkfields	Check for valid input fields
offmat	Run Matlab/Octave simulations offline.

2.1.7 MeX and MeX support functions

fastexp	Calculate `exp( i * x )` quickly.
saddle	Evaluate the MGF saddle point.
cmaadaptivefilter	Polarization demultiplexing filter using CMA algorithm.
easiadaptivefilter	Source separation filter using EASI algorithm.
comp_mex	Compile all `.c` files into the directory.

2.2 Examples

In the Optilux package the sub-directory `/examples` contains many examples describing the Optilux syntax and the main functions.

2.3 myfunction

Write here the first comment line (the H1 line) of the help of your function. In the following "Syntax" section write all the possible calls to your function (`myfunction` in this example)[1]

2.3.1 Syntax

```
d = myfunction(A)
d = myfunction(A,B)
[x,y] = myfunction(A)
```

2.3.2 Description

`d = myfunction(A)` write here the description of the first syntax form

Note: If you have some notes about the function, use the table environment with one cell. Set properly the cell width as a % of the line width (in this example it is 80%).

`d = myfunction(A,B)` description of the second syntax form.
`[x,y] = myfunction(A)` description of the third syntax. And so on.

If you have a flag with many values use the following syntax:
`flag` can be (use the itemize environment):

- `'option1'` Computes the absolute value...
- `'option2'` Evaluates the...

The following subsections remarks, example, details are optional.

2.3.3 Remarks

If you have some important remarks. E.g. limitations, input types, approximations.

2.3.4 Example

Add here some examples. Use the typewriter character for any piece of code. E.g.

```
A=sin(2x);
d = myfunction(A);
fprintf('x=%.3f yields d=%.3f\n',x,d);
```

2.3.5 Details

A list of some details. You can also call this subsection theory, algorithm, or with other names that are useful to focus on specific aspects of the function. This subsection may have other subsubsections.

The final subsection “See Also” is mandatory except for very simple functions.

In the optilux directory you can find the template `myfunction.lyx` in LyX or `myfunction.tex` in LaTeX. Write your function doc following the template and then send it to the Optilux team for comparing in a next Optilux release. In the same files you can also find a template for adding a new Section to the Background part of the documentation. Any contribution is welcome.

[1]myfunction is a template for writing a function help in Optilux.

2.3.6 See Also

reset_all, lasersource

2.3.7 References

Cite the reference using the standard way and support them with a brief description. E.g. In [1] you can find a good tutorial about the group-velocity dispersion.

2.4 reset_all

Reset all global variables and initializes the simulation.

2.4.1 Syntax

```
RESET_ALL(NSYMB,NT,NCH)
RESET_ALL(NSYMB,NT,NCH,OPT1)
WRN=RESET_ALL(NSYMB,NT,NCH,OPT1)
RESET_ALL(NSYMB,NT,NCH,OPT1,OPT2)
```

2.4.2 Description

Note: THIS FUNCTION MUST BE CALLED IN EACH SIMULATION.

`RESET_ALL(NSYMB,NT,NCH)` initializes the global struct variable `GSTATE`.

The fields of the structure `GSTATE` are the following:

- `GSTATE.NSYMB`: Number of symbols = `NSYMB`. For binary transmissions `NSYMB` is the number of bits.
- `GSTATE.NT`: Number of discrete points x symbol = `NT`. See here for more details about the choice of `GSTATE.NT`.
- `GSTATE.NCH`: Number of channels = `NCH`.
- `GSTATE.FN`: Frequency normalized to the symbol rate (R), i.e. `GSTATE.FN` = freq/R with freq the frequency in [Hertz], R the symbol rate in [baud]. See _description for more details.
- `GSTATE.SYMBOLRATE`: Symbol rate (R) in [GBaud] equal for all channel. See the examples for how to manage different channel symbol rates. The symbolrate will be initialized to a numerical value in electricsource.
- `GSTATE.PRINT`: true/false. True: Print functions details to file.

`RESET_ALL(NSYMB,NT,NCH,OPT1)` creates the output directory `OPT1` that will collect a summary of each function operation within the file `simul_out`. The file `simul_out` is appended each call. The output directory will also contain any signal printed to file by Optilux. In presence of `OPT1` there is the additional field:

- `GSTATE.DIR = OPT1`

`WRN=RESET_ALL(NSYMB,NT,NCH,OPT1)` sets `WRN=true` if the dimension of `simul_out` exceeds 50 Mbytes, otherwise `WRN=false`.

RESET_ALL(NSYMB,NT,NCH,OPT1,OPT2) with OPT2='noprint' creates the output directory OPT1 but all functions called by Optilux will not print any detail in simul_out.

RESET_ALL initializes the fundamental constants (like Planck's, speed of light, etc) in the global variable CONSTANTS.
There are other fields of GSTATE that are set to empty by RESET_ALL and will be initialized by create_field. They are:

- GSTATE.FIELDX: x-component of the electric field.
- GSTATE.FIELDY: y-component of the electric field.
- GSTATE.FIELDX_TX: copy of GSTATE.FIELDX. Useful for back-to-back measurements.
- GSTATE.FIELDY_TX: same as GSTATE.FIELDX_TX, but for y polarization.
- GSTATE.DELAY: Overall delay cumulated in the optical line, normalized to 1/R. Size: [2,NCH] if the y-component is empty, else [1,NCH].
- GSTATE.DISP: cumulated dispersion [ps/nm] in the system. Size: [2,NCH] if the y-component is empty, else [1,NCH].

Other fields of GSTATE will be initialized by lasersource. They are:

- GSTATE.LAMBDA: Channels wavelength [nm]. Size: [1,NCH]
- GSTATE.POWER: Transmitted Signal PEAK power [mW]. Size: [1,NCH]

2.4.3 See also

create_field, electricsource, lasersource

2.5 create_field

create the electric field

2.5.1 Syntax

```
CREATE_FIELD(FTYPE,SIGX)
CREATE_FIELD(FTYPE,SIGX,SIGY)
CREATE_FIELD(FTYPE,SIGX,SIGY,OPTIONS)
```

2.5.2 Description

Note: THIS FUNCTION MUST BE CALLED IN EACH SIMULATION THAT OPERATES ON THE ELECTRIC FIELD.

CREATE_FIELD(FTYPE,SIGX) creates the electric field. SIGX is a matrix [Nfft,Nch] containing on columns the x polarization of the electric fields to be multiplexed together. Nch is the overall number of channels, Nfft the number of FFT points. CREATE_FIELD creates new fields of the global variable GSTATE, GSTATE.FIELDX and its copy GSTATE.FIELDX_TX, respectively. GSTATE.FIELDX is the electric field that will be propagated in the optical system.

FTYPE can be 'sepfields' or 'unique'. With 'sepfields' GSTATE.FIELDX is a copy of SIGX. 'sepfields' is useful for propagation in optical fibers in absence of four wave mixing (FWM) and allows to obtain faster runs. 'unique' combines all channels into a unique channel and allows to account for FWM in optical fibers.

`'unique'` allows therefore more accurate results even if it is slow since requires larger value of `GSTATE.NT` (see reset_all) for accounting all channels. With option `'unique'` `CREATE_FIELD` acts as an ideal multiplexer and yields `GSTATE.FIELDX` of size `[Nfft,1]`. More information can be found here.

`CREATE_FIELD(FTYPE,SIGX,SIGY)` operates on the x `(SIGX)` and y `(SIGY)` polarization creating `GSTATE.FIELDX`, and `GSTATE.FIELDY` (and their copies, `GSTATE.FIELDX_TX` and `GSTATE.FIELDY_TX`, respectively). `SIGX` and `SIGY` must have the same size `[Nfft,Nch]`.

`CREATE_FIELD(FTYPE,SIGX,SIGY,OPTIONS)` accepts the optional parameter `OPTIONS`, containing:

- `OPTIONS.delay`: can be the string `'rand'` or a vector of double. In the first case a uniform distributed random delay between `[0,1]` is added to the channels before creating the electric field. In the second case the vector is used as delay. The values are normalized to `1/GSTATE.SYMBOLRATE`, i.e. the symbol time. See signal convention for more details.
- `OPTIONS.power`: if set to `'average'`, the power defined in lasersource is not the peak power (default) but the average power. In such a case `GSTATE.POWER` is changed accordingly.

Note: for hybrid symbol-rate systems, the delay is normalized to the current (last defined) symbol time (`1/GSTATE.SYMBOLRATE`) for all channels.

In absence of y polarization simply use:
`CREATE_FIELD(FTYPE,SIGX,[],OPTIONS)`

`CREATE_FIELD` initializes the global variable `GSTATE.DELAY` to zero or to the value imposed by `OPTIONS.delay`.
`CREATE_FIELD` initializes the global variable `GSTATE.DISP` to zero.

2.5.3 See also

reset_all, lasersource, electricsource

2.6 pattern

Create the sequence pattern with rules.

2.6.1 Syntax

```
PAT=PATTERN(PTYPE,NSEED,OPTIONS)
[PAT,BMAT] = PATTERN(PTYPE,NSEED,OPTIONS)
```

2.6.2 Description

`PAT=PATTERN(PTYPE,NSEED,OPTIONS)` returns in `PAT` a sequence of `Nsymb` integers representing the symbol-pattern for a digital modulation. `Nsymb=GSTATE.NSYMB` see reset_all. In absence of `OPTIONS` the pattern is a bit-pattern, i.e. a vector of 0 and 1.
`PTYPE` is the type of the pattern and can be one of the following:

- `'debruijn'`: creates a De Bruijn sequence (DBS) (each subsequence of length `log2(Nsymb)` appears exactly once in a DBS) [2, 3]. In the binary case, a DBS is a pseudo random binary sequence (PRBS) with an additional zero added to to the longest sequence of 0.
  `NSEED` is the DBS seed:
  - `0 <= NSEED < Nsymb/4` yields a unique DBS, i.e. it is not possible to obtain the same DBS with a different `NSEED`, neither with a circular shift.

  - `NSEED >= Nsymb/4` does not yield a unique DBS, but a circular random delayed version of a DBS with `NSEED < Nsymb/4`.
  - `NSEED` must be `< Nsymb/4*(Nsymb-1)`.

Note: It is not possible to have the same sequence for different `NSEED`.

- `'random'`: creates a uniform distributed random sequence using `rand`.
- <sequence of numbers>: create a periodic repetition of the sequence up to length `Nsymb`, and truncate when necessary.
- <file>: reads the pattern from 'file' using `LOAD`. 'file' contains the pattern for the channel.

`OPTIONS` is an optional parameter containing:

- `OPTIONS.alphabet`: is the alphabet of the pattern.
  - Example:
    `PAT=PATTERN('debruijn',0,OPTIONS)` with `OPTIONS.alphabet=4` returns: `PAT=[1 1 2 3 0 3 1 3 3 2 2 1 0 2 0 0]`, i.e. a DBS sequence with symbols `(0,1,2,3)` containing all couples of symbols exactly once.
  - Example: `PAT=PATTERN('random',OPTIONS)` with `OPTIONS.alphabet=8` may return: `PAT=[6 3 7 6 6 2 3 7 3 2 5 4 4 2 0 2]`

`[PAT,BMAT] = PATTERN(PTYPE,NSEED,OPTIONS)` returns in `BMAT` the binary representation of `PAT`. `BMAT` is a matrix of size `[Nsymb,ceil(log2(OPTIONS.alphabet))]`. The decimal representation of `BMAT(k,:)` is `PAT(k)`.
Example: For a QPSK modulation, the two columns of `BMAT` represent the in-phase and quadrature component of a pseudo random quaternary sequence (PRQS), [3].

Note: `PAT` and `BMAT` are of type double even for binary symbols.

2.6.3 Examples

2.6.3.1 Example 1

`pat=pattern('20104101')`
creates a periodic repetition of the sequence up to length `Nsymb`, and truncates when necessary.
For instance, with `Nsymb=16` returns `pat = [2 0 1 0 4 1 0 1 2 0 1 0 4 1 0 1]`. The sequence can be a string or a vector of double, e.g:
`pat=pattern('20104101')` and
`pat=pattern( [2 0 1 0 4 1 0 1])`
are identical

2.6.3.2 Example 2

`pat=pattern('filename')`
load the sequence from file `'filename'`. The pattern is obtained by concatenating the rows of the file. For instance with `Nsymb=8`, the pattern `pat=[3 0 1 0 1 2 1 0]` can be written into `'filename'` in the following equivalent forms:

```
3 0 1 0
             or     3 0 1 0 1 2 1 0 .
1 2 1 0
```

2.6.4 Details

`PAT=PATTERN('debruijn',1)` with `GSTATE.NSYMB=8` returns the following:
`PAT=[0 0 0 1 1 1 0 1];`
Such a sequence contains the following subsequences of length $\log_2(8) = 3$: `000,001,011,111,110,101,010,100`. The last two sequences can be found by periodic repetition of `PAT`.

2.6.5 See also

electricsource, pat_decoder

2.6.6 References

This function implements the algorithm proposed in [2]. An useful reference about quaternary De Bruijn sequences can be found in [3].

2.7 pat_encoder

Symbols encoder.

2.7.1 Syntax

```
PAT=PAT_ENCODER(PAT,MODFORMAT)
[PAT PATBIN]=PAT_ENCODER(PAT,MODFORMAT)
[PAT PATBIN]=PAT_ENCODER(PAT,MODFORMAT,OPTIONS)
```

2.7.2 Description

`PAT=PAT_ENCODER(PAT,MODFORMAT)` given the pattern `PAT` generated by pattern returns the encoded pattern [4].
`MODFORMAT` is the modulation format and can be `'ook'`, `'dpsk'`,`'nf-dpsk'`, `'psbt'`, `'dqpsk'`, `'nf-dqpsk'`. See the Glossary for a list of acronyms. Such function must be called before electricsource. With `'ook'` it is actually unnecessary to call this function (`PAT_ENCODER` does nothing).
In case of `'dqpsk'` or `'nf-dqpsk'`, the pattern is assumed to be quaternary unless `OPTIONS.binary` is set to `true`.
`[PAT PATBIN]=PAT_DECODER(PAT,MODFORMAT,OPTIONS)` returns in `PATBIN` also the binary representation of the M-ary pattern and thus `PATBIN` is a matrix `[NSYMB, log2(M)]`, being `M` the cardinality of the pattern.

In case of PDM coherent transmissions, `PAT_ENCODER` operates only on the pattern associated on one polarization, since the differential codings on the two polarizations are independent. Thus, you have to calculate the encoded pattern for both the x and y components of the signal.

2.7.3 See Also

pattern, pat_decoder, eval_eye

2.7.4 References

A detailed analysis of DPSK and QPSK modulation formats is available in [5]. PSBT and Enhanced PSBT (EPSBT) are described in [6, 7, 8]. NF-DPSK was presented in [9] and analyzed in [10, 11]. Finally NF-DQPSK was proposed in [12].

2.8 pat_decoder

Symbols decoder.

2.8.1 Syntax

```
PAT=PAT_DECODER(PAT,MODFORMAT)
[PAT PATBIN]=PAT_DECODER(PAT,MODFORMAT)
[PAT PATBIN]=PAT_DECODER(PAT,MODFORMAT,OPTIONS)
```

2.8.2 Description

`PAT=PAT_DECODER(PAT,MODFORMAT)`, given the pattern `PAT` generated by pattern, returns the decoded pattern [4].
`MODFORMAT` is the modulation format and can be `'ook', 'dpsk','nf-dpsk', 'psbt', 'dqpsk', 'nf-dqpsk'`. See the Glossary for a list of acronyms. Such function must be called before functions that operates on the received signal, like `eval_eye`, ber_kl, `ber_estimate`, etc. With `'ook'` it is actually unnecessary to call this function (`PAT_DECODER` does nothing).
In case of `'dqpsk', 'nf-dqpsk'`, the pattern is assumed to be quaternary unless `OPTIONS.binary` is set to `true`.

`[PAT PATBIN]=PAT_DECODER(PAT,MODFORMAT,OPTIONS)` returns in `PATBIN` also the binary representation of the M-ary pattern and thus `PATBIN` is a matrix `[NSYMB, log2(M)]`, being `M` the cardinality of the pattern.
In case of PDM coherent transmissions, `PAT_DECODER` operates only on the pattern associated on one polarization, since the differential codings on the two polarizations are independent. Thus, you have to calculate the decoded pattern for both the x and y components of the signal.

2.8.3 See Also

pattern, pat_encoder, eval_eye, ber_kl, ber_estimate

2.8.4 References

A detailed analysis of DPSK and QPSK modulation formats is available in [5]. PSBT and Enhanced PSBT (EPSBT) are described in [6, 7, 8]. NF-DPSK was presented in [9] and analyzed in [10, 11]. Finally NF-DQPSK was proposed in [12].

2.9 pat2stars

Convert an M-ary pattern into a complex constellation

2.9.1 Syntax

```
STARS=PAT2STARS(PAT,FORMAT)
STARS=PAT2STARS(PAT,FORMAT,OPTIONS)
```

2.9.2 Description

`STARS=PAT2STARS(PAT,FORMAT)` returns in `STARS` the values of the complex constellation associated with the M-ary `PAT` pattern. The supported modulation formats in `FORMAT` are (see the Glossary for a list of acronyms):

- `'ook'`
- `'psbt'`
- `'bpsk', 'dpsk', 'nf-dpsk'`
- `'qpsk', 'dqpsk', 'nf-dqpsk'`

The constellation is associated with the pattern through Gray coding, thus, for example, the QPSK constellation associated with the symbols `[0, 1, 2, 3]` is `[1, i, -i, -1]`.

`STARS=PAT2STARS(PAT,FORMAT,OPTIONS)` allows to modify the default behaviour.
Currently supported fields are:

- `OPTIONS.binary`: if set to `true`, `PAT2STARS` assumes the pattern is binary and thus is a matrix of size `[NSYMB, log2(M)]`, being `M` the cardinality of the alphabet. If `M=2`, this option is simply neglected.

2.9.3 See Also

stars2pat, pat_decoder

2.9.4 References

See [4] for more details. A detailed analysis of DPSK and QPSK modulation formats is available in [5]. PSBT and Enhanced PSBT (EPSBT) are described in [6, 7, 8]. NF-DPSK was presented in [9] and analyzed in [10, 11]. Finally NF-DQPSK was proposed in [12].

2.10 samp2pat

Convert received samples into a pattern

2.10.1 Syntax

```
PAT_RX = SAMP2PAT(X,S,OUTVALUE)
```

2.10.2 Description

`PAT_RX = SAMP2PAT(X,S,OUTVALUE)` takes the samples `OUTVALUE` returned by eval_eye or dsp4cohdec and converts them into a receiver pattern `PAT_RX`. Currently supported modulation formats are '`ook`', '`dpsk`', '`dqpsk`', '`qpsk`'.
`X` is a structure that describes the receiver, while `S` is a structure used to specify how to convert the samples into a pattern. Its fields are:

- `S.alphabet` = vector of the alphabet of the desired output pattern
- `S.thr` = vector of threshold values to apply in order to decide how to map a sample into an alphabet symbol. `S.thr` length must be equal to `S.alphabet` length minus one

2.10.3 See Also

eval_eye, dsp4cohdec, ber_estimate

2.11 stars2pat

Convert a complex constellation into a pattern

2.11.1 Syntax

```
PAT=STARS2PAT(PAT,FORMAT)
[PAT, BPAT]=STARS2PAT(PAT,FORMAT)
```

2.11.2 Description

`PAT=STARS2PAT(STARS,FORMAT)` returns in `PAT` the M-ary pattern associated with the complex constellation. The supported modulation formats in `FORMAT` are (see the Glossary for a list of acronyms):

- `'ook'`
- `'psbt'`
- `'bpsk', 'dpsk', 'nf-dpsk'`
- `'qpsk', 'dqpsk', 'nf-dqpsk'`

The association rule between constellation and pattern is the same as in pat2stars.

`[PAT BPAT]=STARS2PAT(STARS,FORMAT)` returns also the binary equivalent of the M-ary pattern.

2.11.3 References

See [4] for more details. A detailed analysis of DPSK and QPSK modulation formats is available in [5]. PSBT and Enhanced PSBT (EPSBT) are described in [6, 7, 8]. NF-DPSK was presented in [9] and analyzed in [10, 11]. Finally NF-DQPSK was proposed in [12].

2.11.4 See Also

pat2stars, pat_decoder

2.12 electricsource

Create the electric modulating signal.

2.12.1 Syntax

```
ELEC=ELECTRICSOURCE(PAT, FORMAT, SYMBRATE, PTYPE, DUTY, ROLL)
ELEC=ELECTRICSOURCE(...,INPOW, ...)
ELEC=ELECTRICSOURCE(..., ORD, ...)
ELEC=ELECTRICSOURCE(..., PAR, ...)
```

2.12.2 Description

`ELEC=ELECTRICSOURCE(PAT, FORMAT, SYMBRATE, PTYPE, DUTY, ROLL)` returns in `ELEC` the electric signal which is passed to one input of the modulator, using the pattern `PAT`, created using the `pattern` function.

`ELECTRICSOURCE` acts on a per channel basis, so if you want to create the driving signals for a N-channel WDM system you have to call `ELECTRICSOURCE` N times. `FORMAT` is a string that described the modulation format. The supported formats are (see the Glossary for a list of acronyms):

1. `'ook'` (using `mz_modulator`): lower/upper values of the electric signal are 0 and 1.
2. `'bpsk'` (and thus `'dpsk'` and `'nf-dpsk'`, using `mz_modulator` or phase_modulator): lower/upper values of the electric signal are -1 and 1.
3. `'psbt'` (using `mz_modulator`): lower/upper values of the electric signal are -1 and 1. You can also specify two fields in the structure of parameters `PAR` to model the electrical filter:
    - `par.efilt` = electrical filter type (default: `'bessel5`')
    - `par.efiltbw` = electrical filter bandwidth (default: `0.3`)
4. `'qpsk'` (and thus `dqpsk` and `nf-dqpsk`): lower/upper values of the electric signal are -1 and 1. Only the driving signal for one of the two quadratures is created and thus `ELECTRICSOURCE` must be called twice to get the required inputs of `qi_modulator`.
5. `'userdef`': a custom electric signal is generated: the user must specify the following fields of the structure `PAR`:
    - `PAR.alphabet` = size of the alphabet of `PAT`
    - `PAR.limits` = a `[2x1]` vector containing lower and upper values of the generated signal. Symbols are assumed equally spaced
      OR
    - `PAR.ampls` = an `[PAR.alphabet,1]` vector containing amplitudes associated to every symbol of `PAT`

`SYMBRATE` is the signal's baudrate in [Gbaud], and it is associated to the global variable `GSTATE.SYMBOLRATE`. `PTYPE` is the pulse type used to create the electric signal. It can be one of the following strings:

1. <string used in myfilter>: filters an ideal signal with the correspondent filter in `MYFILTER`. In this case `ROLL` is the 3dB bandwidth and `ORD` is the order for special filters.
2. '`cosroll`': Pulses with a raised cosine behavior during the commutation states. In this case 0<`ROLL`≤1 indicates the roll-off. The elementary pulse assumes the form [4]:

$$p(t) = \begin{cases} 1 & 0 \le |t| \le \frac{(1-r)\cdot d}{2} \\ \frac{1}{2}\left\{1+\cos\left[\frac{\pi}{r\cdot d}\cdot\left(|t|-\frac{(1-r)\cdot d}{2}\right)\right]\right\} & \frac{(1-r)\cdot d}{2} \le |t| \le \frac{(1+r)\cdot d}{2} \\ 0 & |t| > \frac{(1+r)\cdot d}{2} \end{cases}$$

   where r is the roll-off and d the duty cycle

3. '`dirac`': Dirac's delta pulses. `'idpulse'`: Ideal pulses with only two levels. Do not confuse with the string `'ideal'` which calls for the ideal filter in `myfilter`.
4. '`sech`': Bright solitons (still to be implemented)
5. '`tanh`': Dark solitons (still to be implemented)

`ROLL` must be always declared. If you don't need `ROLL`, set it, for instance, equal to the empty variable, i.e. `ROLL`=[]. `DUTY` is the duty-cycle, and must be 0≤`DUTY`≤1.

`ELEC=ELECTRICSOURCE(...,INPOW, ...)` , `INPOW` is set to `'power'`, allows to do the pulse shaping on the signal's power (`abs(.)^2`); otherwise the shaping is done on the electric field.
`ELEC=ELECTRICSOURCE(..., ORD , ...)` is used to specify the order `ORD` of the special filter employed to shape the pulse.
`ELEC=ELECTRICSOURCE(...,PAR, ...)` is used in conjunction with `PSBT` modulation. It is a structure used to define the type of filter and its bandwidth used at the transmitter side.

2.12.3 Example

Fig. 2.1 shows the electric signal for an `OOK` modulation using the pattern `'010'` and duty-cycle 1, for some different types of pulse.

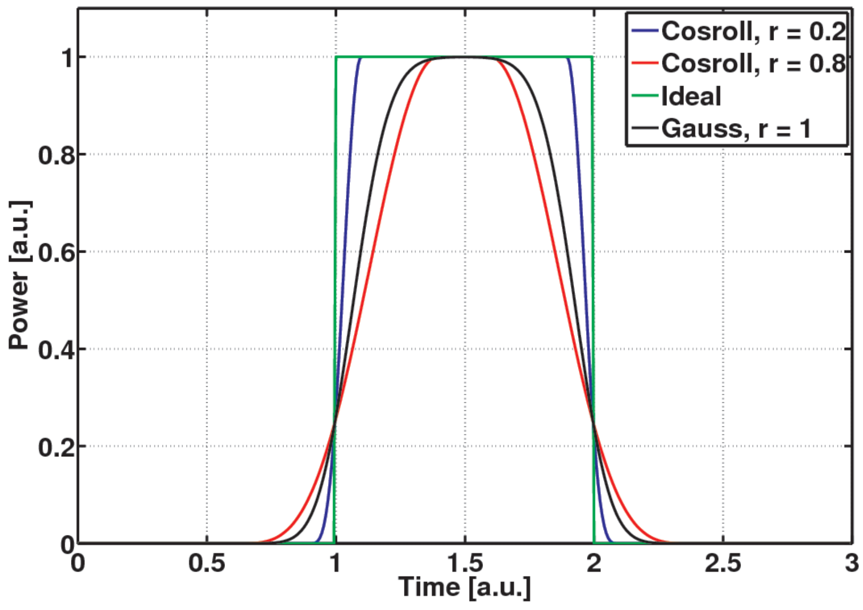

Figure 2.1: Electric signal obtained using ELECTRICSOURCE with four different type of pulse.

2.12.4 See Also

pattern, mz_modulator, lasersource

2.12.5 References

A detailed analysis of DPSK and QPSK modulation formats is available in [5]. PSBT and Enhanced PSBT (EPSBT) are described in [6, 7, 8]. NF-DPSK was presented in [9] and analyzed in [10, 11]. Finally NF-DQPSK was proposed in [12].

2.13 lasersource

Multichannel laser transmitter

2.13.1 Syntax

```
E=LASERSOURCE(PTX,LAM)
E=LASERSOURCE(PTX,LAM,SPAC)
E=LASERSOURCE(PTX,LAM,SPAC,OPTIONS)
```

2.13.2 Description

`E=LASERSOURCE(PTX, LAM)` creates the WDM optical field whose channels are saved into the columns of the matrix `E`.
`PTX` contains the channel's peak power. `PTX` can be vector `[1,Nch]`, being `Nch` the number of channels, or a scalar. In the last case, the same value is used for all channels. The signal's peak powers are all saved here into the global variable `GSTATE.POWER`.
`LAM` are the wavelengths [nm] of the channels. In this function they are associated with the global variable `GSTATE.LAMBDA`, here and for all. `LAM` can be a vector `[1,Nch]` or a scalar: if it is a scalar, the additional parameter `SPAC` is required, which indicates the spacing between channels [nm], while `LAM` is assumed as the central wavelength.

`E=LASERSOURCE(PTX, LAM, SPAC)` creates a WDM optical field whose channels are spaced by `SPAC` [nm]. In this case `LAM` is a scalar and assumed to be the central wavelength.

`E=LASERSOURCE(PTX, LAM, SPAC, OPTIONS)` is used to set optional parameters of the WDM:

1. `OPTIONS.single`: if exists and it is true, `LASERSOURCE` generates a single laser line and thus, `PTX`, `LAM` and `OPTIONS.linewidth`, `OPTIONS.n0`, if present, must be scalar. In this case `SPAC` must still be specified, but its value is neglected and thus could be safely set to 0.

2. `OPTIONS.linewidth`: represents the 3 dB width of the laser line, normalized to the symbolrate. The linewidth can be a scalar or a vector whose length equals the number of channels.

3. `OPTIONS.n0`: represents the one-sided spectral density of a Gaussian complex noise added to the laser, in dB. This way it's possible to set the desired OSNR of the laser.

4. `OPTIONS.anoise`: the matrix with amplitude complex noise samples. `OPTIONS.anoise` dimensions must be `[GSTATE.NSYMB*GSTATE.NT ; GSTATE.NCH]` unless `OPTIONS.single` is set to true. In this case the dimensions must be `[GSTATE.NSYMB*GSTATE.NT ; 1]`

5. `OPTIONS.pnoise`: the matrix with phase noise samples. `OPTIONS.pnoise` dimensions must be `[GSTATE.NSYMB*GSTATE.NT ; GSTATE.NCH]` unless `OPTIONS.single` is set to true. In this case the dimensions must be `[GSTATE.NSYMB*GSTATE.NT ; 1]`

The matrices representing amplitude and phase noise are ignored if `OPTIONS.linewidth` and `OPTIONS.n0` are specified.

Note: `GSTATE.POWER` may be modified by create_field if the user indicates that the power is the average power. See create_field.

2.13.3 Example 1

Fig. 2.2 shows an example of WDM comb with `SPAC`=0.4 nm (50 GHz) and baudrate 10 Gb/s.

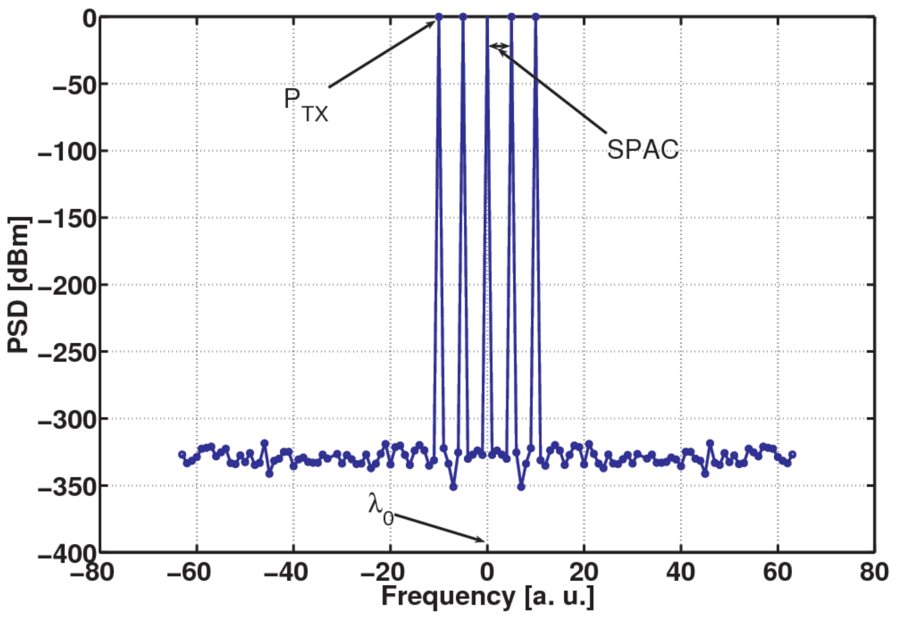

Figure 2.2: Equivalent lowpass of A 5-channel WDM created using lasersource. Since the frequency is normalized, SPAC is expressed in multiple of the baudrate.

2.13.4 Example 2

`E=LASERSOURCE(PTX, 1550, 0.8)`
creates the following vector `GSTATE.LAMBDA`:
GSTATE.LAMBDA = [1548.8 1549.6 1550.4 1551.2]

2.13.5 See Also

pattern, mz_modulator, electricsource, create_field

2.14 linear_modulator

Modulate the optical field with a linear modulator

2.14.1 Syntax

```
E=LINEAR_MODULATOR(E,MODSIG)
E=LINEAR_MODULATOR(E,MODSIG,EXRATIO)
```

2.14.2 Description

`E=LINEAR_MODULATOR(E,MODSIG)` modulates the optical field `E` using a linear modulator, i.e. the modulating electrical signal is impress exactly on the optical signal with an infinite extinction ratio. The parameter `MODSIG` is the electrical driving signal produced by `electricsource`. `LINEAR_MODULATOR` acts on a per channel basis, so if you want to create optical signals for a N-channel WDM system you have to call `LINEAR_MODULATOR` N times.

`E=LINEAR_MODULATOR(E,MODSIG,EXRATIO)` adds a finite extinction ratio `EXRATIO` [dB] to the signal.

2.14.3 See Also

electricsource, lasersource, mz_modulator

2.15 mz_modulator

Modulate the optical field with a Mach-Zehnder Interferometer

2.15.1 Syntax

```
E=MZ_MODULATOR(E,MODSIG)
E=MZ_MODULATOR(E,MODSIG,OPTIONS)
```

2.15.2 Description

`E=MZ_MODULATOR(E,MODSIG)` modulates the optical field `E` using a Mach-Zehnder interferometer [13]. The parameter `MODSIG` is the electrical driving signal produced by `electricsource`. The model of the Mach-Zehnder is the same as in [13]:

$$E_{out} = \frac{E_{in}}{2}\left[e^{j\frac{\pi(V_1(t)-V_{bias})}{V_\pi}} + \gamma e^{j\frac{\pi(V_2(t)+V_{bias})}{V_\pi}}\right]$$

where $V_1 = -V_2 = \frac{MODSIG}{2}$, $\gamma = \frac{\sqrt{\delta}-1}{\sqrt{\delta}+1}$ and δ is the extinction ratio in linear units.
`E=MZ_MODULATOR(E,MODSIG,OPTIONS)` can modify the parameters of the Mach-Zehnder modulator. `OPTIONS` is an optional structure whose fields can be:

- `exratio` : extinction ratio [dB] (default = `inf`)
- `bias` : V_{bias} of the modulator (default = `0`)
- `amplitude`: V_π of the modulator (default = `1`)
- `nochirp`: reduce effect of chirp due to finite extinction ratio [14] (default = `false`)

`MZ_MODULATOR` acts on a per channel basis, so if you want to create optical signals for a N-channel WDM system you have to call `MZ_MODULATOR` N times.

2.15.3 See Also

electricsource, lasersource, linear_modulator , phase_modulator

2.15.4 References

This function implements the model proposed in [13] and a technique used to minimize the effect of chirp in presence of finite extinction ratio and presented in [14].

2.16 phase_modulator

Modulate the optical field with a phase modulator

2.16.1 Syntax

```
E=PHASE_MODULATOR(E,MODSIG)
```

2.16.2 Description

E=PHASE_MODULATOR(E,MODSIG) modulates the optical field E using a phase modulator. The parameter MODSIG is the electrical driving signal produced by electricsource.

2.16.3 See Also

electricsource, lasersource, mz_modulator

2.17 qi_modulator

Modulate the optical field using a QI Mach-Zehnder modulator

2.17.1 Syntax

```
E=QI_MODULATOR(E,MODSIG_I,MODSIG_Q)
E=QI_MODULATOR(E,MODSIG_I,MODSIG_Q)
```

2.17.2 Description

E=QI_MODULATOR(E,MODSIG_I,MODSIG_Q) modulates the optical field using a QI Mach-Zehnder interferometer (QI-MZI) [15, 16]. The QI-MZI is basically a Mach-Zehnder super-structure, i.e. a MZI on whose arms are placed two standard MZI. Moreover on one of the two arms the MZI is followed by a $\frac{\pi}{2}$phase shift. The schematic representation of a QI-MZI is reported in Fig. 2.3.

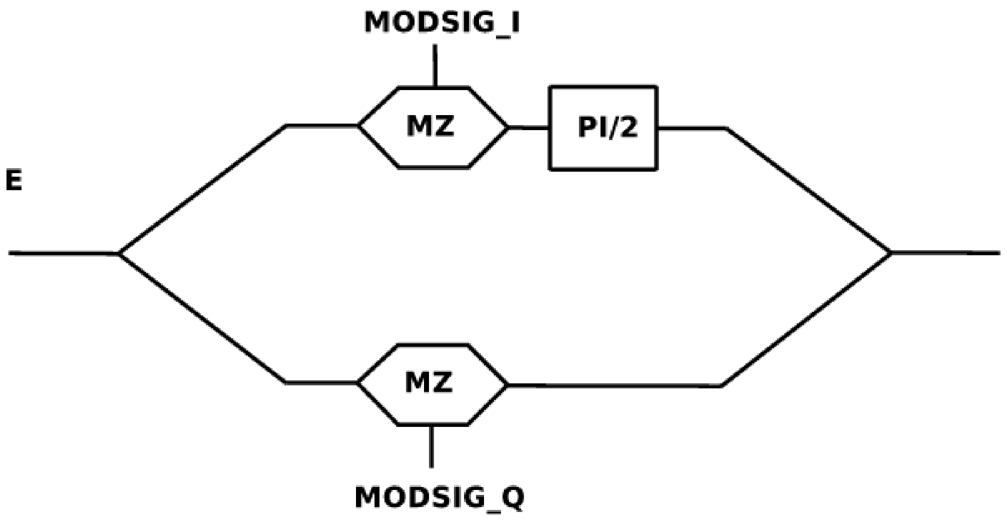

Figure 2.3: Schematic of the QI Mach-Zehnder modulator

The model of the QI-MZI is

$$
\begin{aligned}
E_{out} &= \frac{E_{in}}{2}\left\{iqr\left[e^{j\frac{\pi(V_{i1}(t)-V_{bias})}{V_\pi}} + \gamma e^{j\frac{\pi(V{i_2}(t)+V_{bias})}{V_\pi}}\right]\right. \\
&\left. + (1-iqr)\left[e^{j\frac{\pi(V{q_1}(t)-V_{bias})}{V_\pi}} + \gamma e^{j\frac{\pi(V_{q2}(t)+V_{bias})}{V_\pi}}\right] e^{j\left(\frac{\pi}{2}+V_{biasc}\right)}\right\}
\end{aligned}
$$

$V_{i1} = -V_{i2} = \frac{MODSIG\_I}{2}$ and $V_{q1} = -V_{q2} = \frac{MODSIG\_Q}{2}$ are the in-phase and in-quadrature electrical driving signals produced by electricsource.

`E=QI_MODULATOR(E,MODSIG_I,MODSIG_Q, OPTIONS)` can modify the parameters of the `QI-MZI`. `OPTIONS` is an optional structure whose fields can be:

- `iqratio` : change the power ratio between I and Q arms (iqr, default = 0)
- `biasc` : change the bias between I and Q arms (V_{biasc}, default = 0)
- `exratio` : extinction ratio of the two nested modulators [dB] (default = [inf inf])
- `bias` : V_{bias} of the two nested modulators (default = [0 0])
- `amplitude`: V_{π} of the two nested modulators (default = [1 1])
- `nochirp`: reduce effect of chirp due to finite exratio of the two nested modulators [14]1 (default = [false false])

`QI_MODULATOR` acts on a per channel basis, so if you want to create optical signals for a N-channel WDM system you have to call `QI_MODULATOR` N times.

2.17.3 See Also

electricsource, lasersource, mz_modulator

2.17.4 References

A description of the QI Mach Zehnder modulator (also called dual parallel Mach-Zehnder modulator) is given in [15, 16].

2.18 fiber

Optical fiber in the nonlinear regime.

2.18.1 Syntax

`FIBER(X,FLAG)`

2.18.2 Description

`FIBER(X,FLAG)` solves the nonlinear Schrödinger equation (NLSE) in absence of polarization effects, or the Coupled-NLSE (CNLSE) with polarization effects.

`X` is a structure of fields:

- `X.length`: fiber length [m]
- `X.alphadB`: fiber attenuation [dB/km]
- `X.aeff`: fiber effective area [μm^2]
- `X.n2`: fiber nonlinear index [m^2/W]
- `X.lambda`: wavelength [nm] at which `X.disp` is evaluated
- `X.disp`: fiber chromatic dispersion coefficient [ps/nm/km] @ `X.lambda`
- `X.slope`: fiber slope, i.e. derivative of `X.disp` [ps/nm^2/km] @ `X.lambda`
- `X.dzmax`: max. step for the split-step algorithm [m]
- `X.dphimax`: max. nonlinear phase rotation in each step [rad]

The attenuation is assumed independent from the wavelength.

For the solution of the CNLSE, i.e. with two polarizations, there are also the following additional parameters:

- `X.dgd`: fiber average differential group delay [symbols]
- `X.nplates`: number of waveplates or trunks for PMD emulation
- `X.manakov`: `'yes'`: Solve the Manakov equation. `'no'`: Solve the CNLSE. Default: `'no'`.

In the general case with two polarizations the fiber is the concatenation of randomly oriented polarization maintaining fibers (PMF) fibers. The user can force the use of a single PMF by adding the following optional parameters:

- `X.db0`: birefringence of the PMF fiber at `GSTATE.FN=0`
- `X.theta`: azimuth [rad] of the PMF fiber
- `X.epsilon`: ellipticity [rad] of the PMF fiber

The NLSE is solved by a split-step Fourier algorithm SSFM with a variable step so as to have a maximum nonlinear phase rotation into each step equal to `X.dphimax`. However, the step cannot be larger than `X.dzmax`. See Section 3.4.1.2 for more details. The CNLSE uses the same rules except that the step cannot be larger than `min(X.dzmax,X.length/X.nplates)`. For waveplates shorter than the nonlinear step, the waveplate length is rounded in order to apply the nonlinearity on multiples of the waveplates lengths. On the contrary, the birefringence is applied every nonlinear step.

Alternatively, the step can be chosen adaptively basing the choice on a target local truncation error (NLSE only). In such a case the following parameters should be added to `X`:

- `X.ltol`: local truncation error, i.e. max distance between the field obtained by moving once or twice in a step.
- `X.dphiadapt: true/false`. True: the local truncation error method is applied only in the first step and used to correct `X.dphimax`. After the first step the SSFM proceeds using the approach based on `X.dphimax`. Default: `false`. See Section 3.4.1.4 for more details.

`FLAG` is a string of four characters governing the type of propagation.
The first character is `'g'` if GVD (i.e. β_2, β_3) is on or `'-'` in absence of GVD. Note that with `'sepfields'` in create_field this function accounts for the walkoff effect even with the GVD flag set to `'-'`.
The second character is `'p'` for propagation of a polarized field in presence of birefringence and PMD or `'-'` in absence of such effects.
The third is `'s'` if SPM is on or `'-'` in absence of SPM. Likewise, the fourth character is `'x'` or `'-'` in presence/absence of XPM.
The most complete case is `FLAG='gpsx'` and corresponds to propagation in presence of fiber GVD + PMD + SPM + XPM.
The fourth character of FLAG is active only with channels separated (see option `'sepfields'` in create_field). In this case, the propagation neglects the effect of four-wave mixing, which can be taken in account only by combining all channels into a unique field and hence it is a special case of SPM.

`OUT=FIBER(X,FLAG)` returns in `OUT` a struct containing the birefringence parameters used by `FIBER`:

- `OUT.db0` = birefringence [rad] at `GSTATE.FN=0` (see reset_all).
- `OUT.theta` = azimuth [rad] of all the PMFs composing the fiber.
- `OUT.epsilon` = ellipticity [rad] of all the PMFs composing the fiber.
- `OUT.dgd` = DGD [symbols].

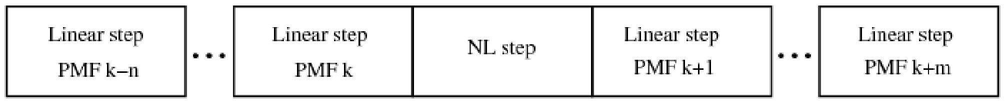

Figure 2.4: Concatenation of linear and nonlinear steps.

- `OUT.lcorr` = length [m] of each PMF trunk.
- `OUT.betat` = beta(omega), i.e. scalar phase shift [rad] including GVD, slope,etc, where omega/2/pi is the vector of FFT frequencies. `betat` is common to both polarizations.
- `OUT.db1` = differential phase shift [rad] induced by PMD.

`OUT` can be used to recover the PMD transfer matrix of the fiber (see inverse_pmd). The CNLSE is described as the concatenation of `X.nplates` PMF trunks, each with principal states of polarization randomly distributed over the Poincaré sphere. Each PMF has constant DGD and randomly distributed birefringence. The nonlinearity is inserted after a certain number of trunks, depending on `X.dzmax` and `X.dphimax`. The diagram is the following in Fig. 2.4:
where PMF k, k=1,2,... is a PMF fiber randomly chosen on the Poincaré sphere.

Note 1: The DGD is in [symbols]. The DGD expressed in [ps] is `x.dgd/GSTATE.SYMBOLRATE*1e3`.

Note 2: `FIBER` updates the global variables `GSTATE.DELAY` and `GSTATE.DISP`. At the output of the fiber `GSTATE.DISP` is increased of `X.disp*X.length*1e-3` [ps/nm] compared to the fiber input.

2.18.3 See also

fiber_gui, NLSE, SSFM

2.18.4 References

A nice reference about the NLSE can be found in [1]. See also the tutorial about the NLSE in Optilux. Concerning the choice of the step see Section 3.4.1.

2.19 fibergui

Optical fiber in the nonlinear regime (GUI tool).

2.19.1 Syntax

```
FIBERGUI(LF,ALPHAdB,AEFF,N2,LAMBDA,DC,SLOPE,DZMAX,DPHIMAX,FLAG,INFOAX)
```

2.19.2 Description

`FIBERGUI(LF,ALPHAdB,AEFF,N2,LAMBDA,DC,SLOPE,DZMAX,DPHIMAX,FLAG,INFOAX)` solves the nonlinear Schrödinger equation (NLSE) in absence of polarization effects using a graphical user interface (GUI).
The fiber parameters are:

- `LF`: length [m]
- `ALPHAdB`: attenuation [dB/km]
- `AEFF`: effective area [μm^2]

- `N2`: nonlinear index [m^2/W]
- `LAMBDA`: wavelength [nm] of `DC`
- `DC`: fiber chromatic dispersion coefficient [ps/nm/km] @ `LAMBDA`
- `SLOPE`: fiber slope, i.e. derivative of `DC` [$ps/nm^2/km$]
- `DZMAX`: max. step for the split-step algorithm [m]
- `DPHIMAX`: max. nonlinear phase rotation in each step [rad]

The NLSE is solved by a split-step Fourier algorithm with a variable step so as to have a maximum nonlinear phase rotation into each step equal to `DPHIMAX`. However, the step cannot be larger than `DZMAX`.

`FLAG` is a string of three characters governing the type of propagation.
The first character is `'g'` if GVD (i.e. β_2, β_3) is on or `'-'` in absence of GVD. Note that with `'sepfields'` in create_field this function accounts for the walkoff effect even with the GVD flag set to `'-'`. The third is `'s'` if SPM is on or `'-'` in absence of SPM. Likewise, the fourth character is `'x'` or `'-'` in presence/absence of XPM. The most complete case is `FLAG='gsx'` and corresponds to propagation in presence of fiber GVD+SPM+XPM.
E.g. Propagation with GVD+XPM, without SPM -> `FLAG='g-x'`.
The third character of `FLAG` is active only with channels separated (see option `'sepfields'` in create_field). In this case, the propagation neglects the effect of four-wave mixing, which can be taken in account only by combining all channels into a unique field and hence it is a special case of SPM.

`INFOAX` governs the GUI tool. The options of `INFOAX` are:

- `INFOAX.ch` = channel under investigation.
- `INFOAX.flag1d = [a b c]`. `a=1` -> plot the power. `b=1` -> plot the phase. `c=1` -> plot the chirp. `a` or `b` or `c=0` -> don't plot.
- `INFOAX.flag3d` = same form as `INFOAX.flag1d`, but for 3D plot.
- `INFOAX.ch` = channel to be plotted. E.g. `INFOAX.ch = [1 3]` indicates that only channels 1 and 3 will be plotted.
- `INFOAX.axprop` = cell array containing valid pairs of axes properties (see `AXES` for more details). E.g. `INFOAX.axprop = {'XLim',[0 10],'YLim',[0 2],'FontSize',24}`.

Note 1: `FIBERGUI` does not account for PMD yet. In future versions of Optilux it will be implemented like fiber.

Note 2: This function contains some known minor bugs.

Note 3: `FIBERGUI` does not work under Octave.

2.19.3 See Also

fiber

2.20 ampliflat

Ideal Optical amplifier with ASE noise.

2.20.1 Syntax

```
AMPLIFLAT(X,ATYPE)
AMPLIFLAT(X,ATYPE,OPTIONS)
```

2.20.2 Description

`AMPLIFLAT(X,ATYPE)` amplifies the optical field. `ATYPE` is a string equal to `'gain'` if the amplifier has a flat power gain equal to `X` [dB]. Otherwise, `ATYPE` can be `'fixpower'` if the amplifier takes the gain so as to have an output average power for channel `ceil(Nch/2)` equal to `X` [mW], `Nch` being the number of channels. This options works only with channels separated (see create_field).
`AMPLIFLAT(X,ATYPE,OPTIONS)` has the additional variable `OPTIONS` to insert the amplified spontaneous emission (ASE) noise.

`OPTIONS` is a sctructure whose fields can be:

- `OPTIONS.f`: [dB] is the optical ASE noise figure, which corresponds to a one-sided ASE power spectral density, on two polarizations, N0 = F*(Gain-1)*h*nu, with Gain the amplifier gain, h the Planck's constant and nu the channel central frequency. Hence, ASE power on a frequency band B is Pase = N0*B.
- `OPTIONS.asepol`: If `'asex'` allows to force to zero the ASE noise added to `GSTATE.FIELDY`, while for `'asey'` allows to force to zero the noise added to `GSTATE.FIELDX`.
- `OPTIONS.noise`: A matrix containing user's defined complex, unit variance, ASE noise samples. `OPTIONS.noise` must have the same size of `[GSTATE.FIELDX, GSTATE.FIELDY]` and must be read in that way.

If the amplifier does not generate ASE noise, don't set `OPTIONS`.

Note: `AMPLIFLAT` assumes the same gain for both polarizations.

2.20.3 Example

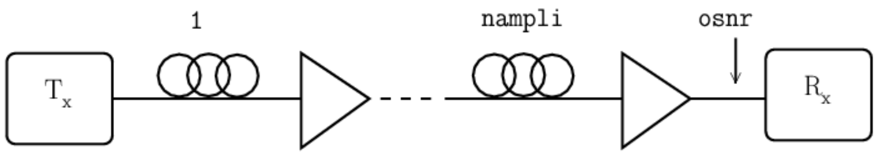

Figure 2.5: Periodic optical link of `nampli` spans.

Suppose an optical transparent system composed of `nampli` identical spans, each amplified at the end; a signal of average power `Pavg`, central wavelength `lam`, propagating in such a system. Given the link optical signal to noise ratio `osnr` in [dB] over a bandwidth of `osnbw` (e.g. 0.1 nm), the noise figure `F` [dB] of each amplifier of gain `Gflat` can be evaluated as follows:

```
CLIGHT= 299792458;       % speed of light in vacuum [m/s]
HPLANCK= 6.62606896e-34; % Planck's constant [J*s]
lam = 1550;              % central wavelength [nm]
osnr = 12;               % OSNR [dB] over a bandwidth osnrbw
osnrbw = 0.1;            % OSNR reference bandwidth [nm]
Gflat = 10;              % amplifiers' flat gain [dB]
nampli = 20;             % number of identical amplifiers
hvdl = -30-10*log10(HPLANCK*CLIGHT/lam*...
    CLIGHT*osnrbw/lam^2*1e18); % conv.  factor
nsp = 10*log10(Pavg) + hvdl - 10*log10(10^(Gflat/10)-1) - 3 - ...
    10*log10(nampli) - osnr; % spontaneous emission factor [dB]
F = nsp + 3;                 % noise figure [dB]
OPTIONS.f = F;               % OPTIONS for ampliflat
```

2.21 inverse_pmd

Inverse PMD matrix.

2.21.1 Syntax

INVERSE_PMD(BRF)
`UINV=INVERSE_PMD(BRF)`
`[UINV,U]=INVERSE_PMD(BRF)`
`INVERSE_PMD(BRF,OPTIONS)`

2.21.2 Description

`INVERSE_PMD(BRF)` applies the inverse PMD matrix of the link to the electric field contained within `GSTATE`. The link parameter are within the cell `BRF`. The k-th field of `BRF` contains the parameters of fiber `k` of the current link that should be inverted. `BRF{k}` is a struct with the following fields:

- `BRF{k}.db0` = birefringence [rad] at `GSTATE.FN=0` (see reset_all).
- `BRF{k}.theta` = azimuth [rad] of all the PMFs composing the fiber.
- `BRF{k}.epsilon` = ellipticity [rad] of all the PMFs composing the fiber.
- `BRF{k}.dgdrms` = r.m.s. DGD [ns] per trunk.
- `BRF{k}.lcorr` = length [m] of each PMF trunk.
- `BRF{k}.betat` = $\beta(\omega)$, i.e. scalar phase shift [rad] including GVD, slope, etc, where $\omega/2/\pi$ is the vector of FFT frequencies. β is common to both polarizations.
- `BRF{k}.db1` = differential phase shift [rad] induced by PMD.

`BRF` is returned by fiber.

`UINV=INVERSE_PMD(BRF)` also returns in `UINV` a 3-D matrix such that `UINV(:,:,n)` contains the `(2,2)` inverse PMD matrix at frequency `GSTATE.FN(n)`. Such matrix includes also the scalar linear distortion like GVD, with exception (see later).

`[UINV,U]=INVERSE_PMD(BRF)` also returns the PMD matrix `U`.

`INVERSE_PMD(BRF,OPTIONS)` allows the following optional parameters:

- `OPTIONS.gvd = 'no'`: do not apply GVD in the `U, UINV` evaluation, i.e. force `BRF{k}.betat = 0`.
- `OPTIONS.apply = 'no'`: do not apply the inverse matrix to `GSTATE.FIELDX` and `GSTATE.FIELDY`.
- `OPTIONS.mat` = `(2,2)` unitary matrix that rotates the reference system before applying `U`. E.g. such matrix can be the one returned by set_sop.

Note 1:`UINV` is a unitary matrix, hence neglects the fiber attenuation. Hence, `Uinv(:,:,n) = U(:,:,n)'`.

Note 2: Given the link in Fig. 2.21, the n-th call to fiber must return `BRF{n}`.

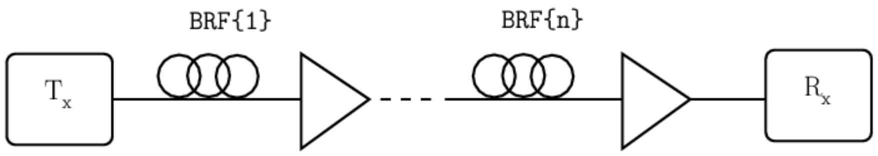

Figure 2.6: The n-th fiber of the line must return `BRF{n}`.

2.21.3 See Also

fiber, set_sop.

2.22 optfilter

Optical filter.

2.22.1 Syntax

```
OPTFILTER(ICH,FTYPE,BW)
OPTFILTER(ICH,FTYPE,BW,ORD)
```

2.22.2 Description

`OPTFILTER(ICH,FTYPE,BW)` filters the optical field `GSTATE.FIELDX` and `GSTATE.FIELDY` with the filter `FTYPE` (see `myfilter`) having 3-dB bandwidth `BW` (normalized to the symbolrate), with central frequency centered on the `ICH` channel. The resulting filtered field is again associated to `GSTATE.FIELDX` and `GSTATE.FIELDY`, respectively, hence the the field is overwritten.

In the case of separate fields (see create_field) only the `ICH` column of `GSTATE.FIELDX` and `GSTATE.FIELDY` is filtered, while the other columns remain unchanged.
The global variable `GSTATE.DELAY` is updated.

`OPTFILTER(ICH,FTYPE,BW,ORD)` use the additional parameter `ORD` for special filter. E.g. `ORD` is the supergauss order for the supergaussian filter (see myfilter)

2.22.3 Note

Remember that an optical filter of bandwidth `BW` ha an equivalent low pass description of bandwidth `BW/2`. Since Optilux works with the equivalent lowpass, applying an optical filter of bandwidth `BW` is equivalent to applying a lowpass filter of bandwidth `BW/2` to `GSTATE.FIELDX` and `GSTATE.FIELDY` [4, 17].

2.22.4 Examples

Assume that the symbolrate has been initialized to 20 Gsymb/s. Filtering only channel 3 with a supergaussian filter having bandwidth 30 GHz and order 4 can be done with:

```
OPTFILTER(3,'supergauss',30/20,4)
```

2.22.5 See also

myfilter

2.23 receiver_ook

Complete OOK receiver (POST fiber+OBPF+photodiode+LPF).

2.23.1 Syntax

```
IRIC=RECEIVER_OOK(ICH,X)
[IRIC,SN]=RECEIVER_OOK(ICH,X)
```

2.23.2 Description

`IRIC=RECEIVER_OOK(ICH,X)` returns the received current `IRIC` of channel `ICH` of an OOK transmission using the following receiver:

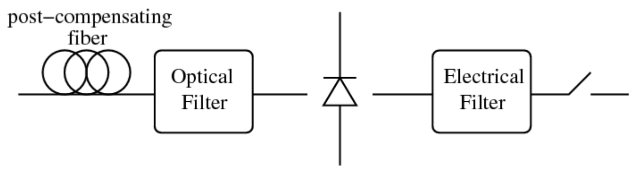

Figure 2.7: OOK receiver.

`[IRIC,SN]=RECEIVER_OOK(ICH,X)` also returns the vector `SN` containing the FFT coefficients of the electric field after the optical filter. `SN` is used by ber_kl.

`X` is a structure of fields:

- `X.oftype` = optical filter (OBPF) type (see myfilter)
- `X.obw` = OBPF 3 dB bandwidth normalized to the symbol rate.
- `X.oord` = optical filter order (for special filters, see myfilter)
- `X.eftype` = electrical filter (LPF) type (see myfilter)
- `X.ebw` = LPF 3-dB bandwidth normalized to the symbol rate.
- `X.eord` = electrical filter order (for special filters, see myfilter)

Optional parameters of `X`:

- `X.dpost` = post compensating fiber cumulated dispersion [ps/nm]
- `X.slopez` = post compensating fiber cumulated slope [ps/nm^2]
- `X.lambda` = wavelength [nm] at which the post compensating fiber has a cumulated dispersion equal to `X.dpost`.
- `X.b2b = 'b2b'` evaluates the current in back-to-back configuration, i.e. with the transmitter connected directly to the receiver. With this option the values of `x.dpost` and `x.slopez` are discarded.

The post-compensating fiber is assumed as a purely ideal-linear fiber, while the photodiode is ideal (`abs(.)^2`).

Note: This function works over a copy of the electric field. All fields of the global variable `GSTATE` are left unchanged.

2.23.3 See also

receiver_dpsk

2.24 receiver_dpsk

Complete DPSK receiver (POST fiber+OBPF+MZ+LPF).

2.24.1 Syntax

```
IRIC=RECEIVER_DPSK(ICH,X)
[IRIC,SN]=RECEIVER_DPSK(ICH,X)
```

2.24.2 Description

`IRIC=RECEIVER_DPSK(ICH,X)` returns the received current `IRIC` of channel `ICH` of an DPSK transmission using the following receiver:

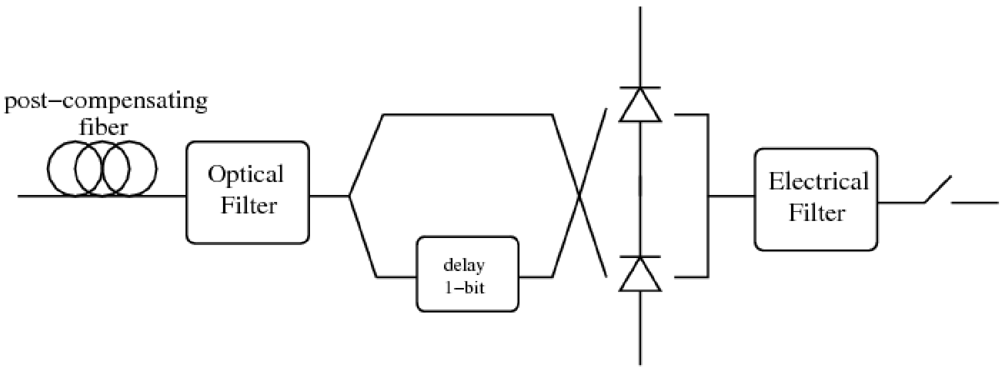

Figure 2.8: DPSK receiver.

`[IRIC,SN]=RECEIVER_OOK(ICH,X)` also returns the vector `SN` containing the FFT coefficients of the electric field after the optical filter. `SN` is used by ber_kl.

`X` is a structure of fields:

- `X.oftype` = optical filter (OBPF) type (see myfilter)
- `X.obw` = OBPF 3 dB bandwidth normalized to the symbol rate.
- `X.oord` = optical filter order (for special filters, see myfilter)
- `X.eftype` = electrical filter (LPF) type (see myfilter)
- `X.ebw` = LPF 3-dB bandwidth normalized to the symbol rate.
- `X.eord` = electrical filter order (for special filters, see myfilter)

Optional parameters of `X`:

- `X.dpost` = post compensating fiber cumulated dispersion [ps/nm]
- `X.slopez` = post compensating fiber cumulated slope [ps/nm^2]
- X.lambda = wavelength [nm] at which the post compensating fiber has a cumulated dispersion equal to X.dpost.

- `X.b2b = 'b2b'` evaluates the current in back-to-back configuration, i.e. with the transmitter connected directly to the receiver. With this option the values of `x.dpost` and `x.slopez` are discarded.
- X.mzdel = specify the delay of the upper brace of the MZDI interferometer for DPSK. The default delay is 1 and it must be comprised in the interval $0 \leq$ X.mzdel $\leq$1. Setting mzdel to a value smaller than 1 implements the Partial DPSK (P-DPSK) [18].

The post-compensating fiber is assumed as a purely ideal-linear fiber, while the photodiodes are ideal (`abs(.)^2`).

Note: This function works over a copy of the electric field. All fields of the global variable `GSTATE` are left unchanged.

2.24.3 Details

Calling $A(t)$ the electric field at the output of the optical filter, the current received by the top photodiode is $\frac{1}{4}\left|A(t)+A(t-T)\right|^2$, while the current received by the bottom photodiode is $\frac{1}{4}\left|A(t)-A(t-T)\right|^2$. T is the bit time. After the differential operation, the current at the input of the low pass filter is:

$$I(t) = \mathrm{Re}\left\{A(t)A^*(t-T)\right\}$$

In Optilux delaying the electric field by T seconds corresponds to a circular shift of `GSTATE.NT` points. See _description for more details.

2.24.4 See also

receiver_ook , receiver_dqpsk

2.25 receiver_dqpsk

Complete DQPSK receiver. (POST fiber+OBPF+MZs+LPF).

2.25.1 Syntax

```
IRIC=RECEIVER_DQPSK(ICH,X)
[IRIC,SN]=RECEIVER_DQPSK(ICH,X)
```

2.25.2 Description

`IRIC=RECEIVER_DQPSK(ICH,X)` returns the received current `IRIC` of channel `ICH` of an OOK transmission using the following receiver:

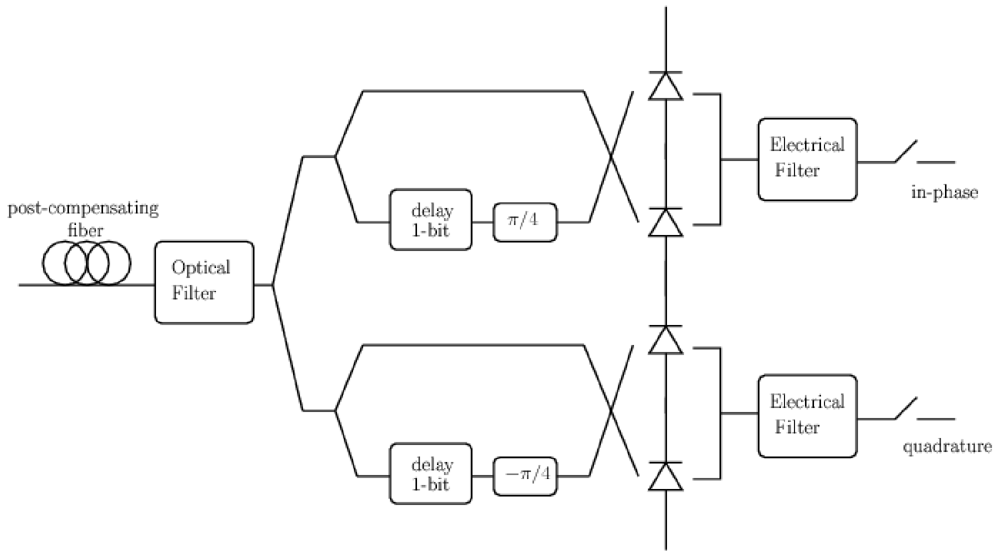

Figure 2.9: DQPSK receiver.

`[IRIC,SN]=RECEIVER_OOK(ICH,X)` also returns the vector `SN` containing the FFT coefficients of the electric field after the optical filter. `SN` is used by ber_kl.

`X` is a structure of fields:

- `X.oftype` = optical filter (OBPF) type (see myfilter)
- `X.obw` = OBPF 3 dB bandwidth normalized to the symbol rate.
- `X.oord` = optical filter order (for special filters, see myfilter)
- `X.eftype` = electrical filter (LPF) type (see myfilter)
- `X.ebw` = LPF 3-dB bandwidth normalized to the symbol rate.
- `X.eord` = electrical filter order (for special filters, see myfilter)
- `X.comp` = component on which evaluate eye and calculate BER. Can be `'phase'` or `'quadrature'`.

Optional parameters of `X`:

- `X.dpost` = post compensating fiber cumulated dispersion [ps/nm]
- `X.slopez` = post compensating fiber cumulated slope [ps/nm^2]
- `X.lambda` = wavelength [nm] at which the post compensating fiber has a cumulated dispersion equal to `X.dpost`.
- `X.b2b = 'b2b'` evaluates the current in back-to-back configuration, i.e. with the transmitter connected directly to the receiver. With this option the values of `x.dpost` and `x.slopez` are discarded.
- X.mzdel = specify the delay of the upper brace of both the MZDI interferometers of the DQPSK receiver. The default delay is 1 and it must be % comprised in the interval 0 $\leq$ X.mzdel $\leq$1. Setting mzdel to a value smaller than 1 implements the Partial DQPSK (P-DQPSK) [19]

The post-compensating fiber is assumed as a purely ideal-linear fiber, while the photodiodes are ideal (`abs(.)^2`).

Note: This function works over a copy of the electric field. All fields of the global variable `GSTATE` are left unchanged.

2.25.3 Details

Calling $A(t)$ the electric field at the output of the optical filter, the current at the input of the low pass filter in the upper arm is:

$$I_I(t) = \mathrm{Re}\left\{A(t)A^*(t-T)e^{-j\frac{\pi}{4}}\right\}$$

being T the bit time. Similarly, in the lower arm we have:

$$I_Q(t) = \mathrm{Re}\left\{A(t)A^*(t-T)e^{+j\frac{\pi}{4}}\right\}$$

In Optilux delaying the electric field by T seconds corresponds to a circular shift of `GSTATE.NT` points. See _description for more details.

2.25.4 See also

receiver_dpsk

2.26 receiver_cohmix

Complete COHerent MIXer receiver. (POST fiber+OBPF+MIX+PD+LPF).

2.26.1 Syntax

```
IRIC=RECEIVER_COHMIX(ICH,X)
[IRIC,SN]=RECEIVER_COHMIX(ICH,X)
```

2.26.2 Description

`IRIC=RECEIVER_COHMIX(ICH,X)` returns the received current `IRIC` of channel `ICH` of an OOK transmission using a coherent receiver.
`[IRIC,SN]=RECEIVER_COHMIX(ICH,X)` also returns the vector `SN` containing the FFT coefficients of the electric field after the optical filter. `SN` is used by ber_kl.

`X` is a structure of fields:

- `X.oftype` = optical filter (OBPF) type (see myfilter)
- `X.obw` = OBPF 3 dB bandwidth normalized to the symbol rate.
- `X.oord` = optical filter order (for special filters, see myfilter)
- `X.eftype` = electrical filter (LPF) type (see myfilter)
- `X.ebw` = LPF 3-dB bandwidth normalized to the symbol rate.
- `X.eord` = electrical filter order (for special filters, see myfilter)

Optional parameters of `X`:

- `X.lodetuning` = Local Oscillator Detuning frequency [Hz]
- `X.lophasenoise` = Local Oscillator Phase Noise Vector [rad]
- `X.lopower` = Local Oscillator Power [dBm]
- `X.dpost` = post compensating fiber cumulated dispersion [ps/nm]
- `X.slopez` = post compensating fiber cumulated slope [ps/nm^2]

- `X.lambda` = wavelength [nm] at which the post compensating fiber has a cumulated dispersion equal to `X.dpost`.
- `X.b2b = 'b2b'` evaluates the current in back-to-back configuration, i.e. with the transmitter connected directly to the receiver. With this option the values of `x.dpost` and `x.slopez` are discarded.

The post-compensating fiber is assumed as a purely ideal-linear fiber, while the photodiode is ideal (`abs(.)^2`).

Note: This function works over a copy of the electric field. All fields of the global variable `GSTATE` are left unchanged.

2.26.3 See also

receiver_dpsk

2.27 eval_eye

Evaluate the eye opening for a non-coherent transmission.

2.27.1 Syntax

```
EO=EVAL_EYE(ICH,X,PAT)
[EO, TS]=EVAL_EYE(ICH,X,PAT)
[EO, TS, IS]=EVAL_EYE(ICH,X,PAT)
[EO, TS, IS, SN]=EVAL_EYE(ICH,X,PAT)
[EO, TS, IS, SN, DELAY]=EVAL_EYE(ICH,X,PAT)
```

2.27.2 Description

`EO=EVAL_EYE(ICH,X,PAT)` evaluates the eye opening `EO` of channel `ICH` of a non-coherent transmission. The eye opening [mA] is defined as: max(worst1-worst0) being worst1 and worst0 the worst mark/space samples.

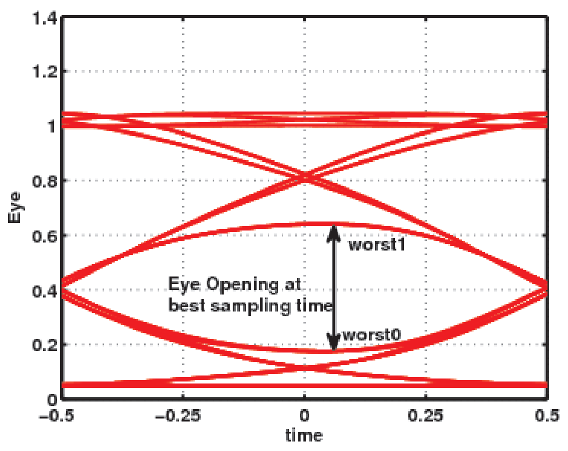

Figure 2.10: Example of eye for OOK transmission.

`[EO,TS] =EVAL_EYE(ICH,X,PAT)` returns also the best sampling time `TS` normalized to the bit time.
`EO` and `TS` are evaluated by parabolic interpolation of the available data. `PAT` is the symbol pattern, after decoding (see pat_decoder). `PAT` can be a vector or a matrix, depending on the modulation format (see next).

`X` is a structure whose fields are:

- `X.rec` = receiver type. Valid arguments are: `'ook','psbt','nf-dpsk'` to use receiver_ook, `'dpsk'` to use receiver_dpsk, `'dqpsk'` to use receiver_dqpsk.
- `X.oftype` = optical filter type (see myfilter)
- `X.obw` = optical filter 3 dB bandwidth normalized to the bit rate
- `X.oord` = optical filter order (if `X.oftype` is `'supergaussian'`)
- `X.eftype` = electrical filter type (see myfilter)
- `X.ebw` = electrical filter 3-dB bandwidth normalized to the bit rate
- `X.eord` = electrical filter order (if `X.eftype` is `'supergaussian'`)

`X` can also have the optional parameters:

- `X.ts` = Fixed sampling time (`-0.5 <= X.ts <= 0.5`).
- `X.plot = 'ploteye'`: plots the eye in the active figure; `'plotcur'` plots the received current.
- `X.color` = color string for the plot (see PLOT). E.g. `'b-'`.
- `X.dpost` = post compensating fiber cumulated dispersion [ps/nm]
- `X.slopez` = post compensating fiber cumulated slope [ps/nm^2]
- `X.lambda` = wavelength [nm] at which the post compensating fiber has a cumulated dispersion equal to `X.dpost`.
- `X.comp` = component on which evaluate eye and calculate BER (dqpsk modulation only). Can be `'phase'` or `'quadrature'` or `'both'`. In the last case the function gets two measurements over the in-phase and quadrature components, sampled with the same clock time. `X.comp='both'` requires `PAT` to be a two-column matrix with the phase/quadrature binary patterns on column 1,2, respectively.
- `X.print` = structure for print. E.g. `X.print = {'nomefile','eye'}` or `X.print ={'nomefile', 'current'}`, prints to file `nomefile` the eye or the current, respectively. nomefile will be place into `GSTATE.DIR` within a directory ending with `'.MOD'`.
- `X.delay = 'theory'` means that the delay uses the theoretical delay saved within `GSTATE.DELAY` (see create_field). By default the delay is measured by a cross-correlation measurement between the received current and an artificial pulse amplitude modulation (PAM) signal with ideal non-return to zero bits with symbols equal to `PAT`. The correlation method is useful in presence of polarization mode dispersion (PMD). See corrdelay.

The receiver is composed of an ideal, purely linear, post compensating fiber + optical filter + optical to electrical converter + electrical lowpass filter. For example see receiver_ook or receiver_dpsk.

`[EO, TS, IS, SN, DELAY]=EVAL_EYE(ICH,X,PAT)` also returns in `IS` a column vector containing the sampled bits. `SN` is a vector containing the FFT coefficients of the signal after the optical filter (`SN` is used by ber_kl). `DELAY` is the overall system delay in bits.
With `X.comp='both'` all variables on output are doubled in size for accounting for the in-phase and quadrature component.

It is `-0.5 <= TS <= 0.5`, with usually `TS ~= 0`.

Note 1: This function works over a copy of the electric field. All fields of the global variable `GSTATE` are left unchanged.

Note 2: All fields of `X` must be lowercase.

Note 3: Please, beware that `EVAL_EYE` needs the decoded pattern, not the transmitted one. See the examples.

2.27.3 See also

pattern, myfilter, best_eye, receiver_ook, receiver_dpsk, receiver_dqpsk, ber_kl, pat_decoder, corrdelay

2.28 ber_kl

Evaluate the ber for noncoherent transmission by Karhunen-Loève method.

2.28.1 Syntax

```
PB=BER_KL(ICH,X,PAT)
[PB,OSNR]==BER_KL(ICH,X,PAT)
[PB,OSNR,EO]==BER_KL(ICH,X,PAT)
```

2.28.2 Description

`PB=BER_KL(ICH,X,PAT)` evaluates the bit error rate `PB` of channel `ICH` of a non-coherent transmission by means of the Karhunen-Loeve (kl) method. Available modulation formats are OOK, DPSK, PSBT, DQPSK (see electricsource).
`PAT` is the pattern for the `ICH` channel, after decoding (see pat_decoder).

`X` is a structure whose fields are:

- `X.rec = 'ook'` to use receiver_ook, `'dpsk'` to use receiver_dpsk, `'dqpsk'` to use receiver_dqpsk
- `X.oftype` = optical filter type (see myfilter)
- `X.obw` = optical filter 3 dB bandwidth normalized to the bit rate
- `X.oord` = optical filter order (for special filter, see myfilter)
- `X.eftype` = electrical filter type (see myfilter)
- `X.ebw` = electrical filter 3-dB bandwidth normalized to the bit rate
- `X.eord` = electrical filter order (for special filter, see myfilter)
- `X.osnr` = Optical signal to noise ratios (osnr), [dB], over which the ber is evaluated. The osnr is over a conventional bandwidth of 0.1 nm and is measured immediately before the receiver. Can be a vector, and its size is also the size of `PB`. `X.osnr` refers to `X.poln` noise polarizations.
- `X.poln` = Noise polarizations, 1 or 2. Note: `X.poln` is independent from the signal polarizations, e.g. the algorithm can work with two noise polarizations and just one signal polarization. However, with two signal polarizations `X.poln` must be 2.

`X` has also the following parameters required by the kl-method:

- `X.eta` = bandwidth expansion factor. The kl method samples the signal and the noise up to a frequency equal to `X.eta` times the bandwidth of the optical filter. Usually it is `1 < X.eta < 3`.

- `X.mu` = Time expansion factor. The memory of the receiver is `X.mu` times the time duration of the memory devices inside the receiver, i.e. the optical/electrical filter. For DPSK there is an additional memory due to the Mach-Zehnder delay interferometer. The memory of such devices is approximated by the inverse of their bandwidths, as suggested in [20]. Usually it is `1 < X.mu < 10`.

- `X.saddle = 'yes'`: the ber is evaluated through the saddle point approximation (faster). `'no'`: the ber is evaluated by numerical integration of the moment generating function (slower, but more accurate).

For more details about `X.eta`, `X.mu` and `X.saddle` see [20]. `X` can also have the optional parameters:

- `X.ber` = reference ber at which the algorithm returns the corresponding osnr, searched within the range `X.osnr` by numerical interpolation.

- `X.interp` = interpolation method for finding `X.ber`, see INTERP1. Default is `'spline'`.

- `X.extrap = 'yes'`: `X.ber` can be extrapolated outside `X.osnr`, see INTERP1. `'no'`: If `X.ber` is outside the range `X.osnr` the function returns `OSNR = NaN`, which is also the default strategy.

- `X.plot = 'ploteye'`: plots the eye in the active figure; `'plotcur'` plots the received current.

- `X.color` = color string for the plot (see PLOT). E.g. `'b-'`.

- `X.dpost` = post compensating fiber cumulated dispersion [ps/nm], i.e. the product chromatic dispersion times fiber length.

- `X.slopez` = post compensating fiber cumulated slope [ps/nm^2], i.e. the product dispersion slope times fiber length.

- `X.lambda` = wavelength [nm] at which the post compensating fiber has a cumulated dispersion equal to `X.dpost`.

- `X.comp` = component on which evaluate eye and calculate BER (DQPSK modulation only). Can be `'phase'` or `'quadrature'`.

- `X.b2b = 'b2b'`: The function works in back-to-back transmission.

- `X.print` = structure for print. E.g. `X.print = {'nomefile','eye'}` or `X.print = {'nomefile', 'current'}`, prints to file `nomefile` the eye or the current, respectively. nomefile will be place into `GSTATE.DIR` within a directory ending with `'.MOD'`.

- `X.delay = 'theory'` means that the delay uses the theoretical delay saved within `GSTATE.DELAY` (see create_field). Per default the delay is measured by a cross-correlation measurement between the received current and an artifical pulse amplitude modulation (PAM) signal with ideal non-return to zero bits with symbols equal to `PAT`. The correlation method is useful in presence of polarization mode dispersion (PMD). See corrdelay.

- X.mzdel = specify the delay of the upper brace of the MZDI interferometer for DPSK/DQPSK. The default delay is 1 and it must be comprised in the interval `0 < mzdel <=1`. Setting `mzdel` to a value smaller than 1 implements the Partial DPSK/DQPSK (P-DPSK/DQPSK) [18].

- `X.threshold` = Fixed threshold for the threshold detector. By default the threshold is optimized for OOK and set to zero for DPSK/DQPSK. The ook receiver is composed of an ideal, purely linear, post compensating fiber + optical filter + photodiode + electrical lowpass filter (see receiver_ook or receiver_dpsk). For DPSK there is also a Mach-Zehnder interferometer before the photodiodes.

`[PB,OSNR,EO]=BER_KL(X,ICH,PAT)` returns in `OSNR` the signal-to-noise ratio [dB] that yields `X.ber`, and in `EO` the normalized eye opening (see eval_eye).

This function implements the algorithm proposed by E. Forestieri in [20]. The DPSK version of the algorithm can be found in [21]. Note that in this function noise parametric gain is neglected.
Many thanks to E. Forestieri for the fortran code of his algorithm, which has been the main inspiration for this function.

Note 1: This function works over a copy of the electric field. All fields of the global variable `GSTATE` are left unchanged.

Note 2: All fields of `X` must be lowercase.

Note 3: The noise is assumed white over the frequency. The possible presence of parametric gain [21] is not accounted by this function in this version of Optilux.

2.28.3 See also

pattern, myfilter, receiver_ook, receiver_dpsk, best_eye, eval_eye, best_sp, pat_decoder

2.29 best_eye

Search algorithm for the best eye opening.

2.29.1 Syntax

```
BEYE=BEST_EYE(ICH,X,PAT)
[BEYE,BPOST]=BEST_EYE(ICH,X,PAT)
```

2.29.2 Description

`BEYE=BEST_EYE(ICH,X,PAT)` searches the minimum eye closure penalty `BEYE` of a non-coherent transmission by varying the post compensating fiber dispersion in front of the receiver (E.g. see receiver_ook or receiver_dpsk).

`[BEYE,BPOST]=BEST_EYE(ICH,X,PAT)` also returns in `BPOST` the best post compensating fiber cumulated dispersion in [ps/nm].
The eye closure penalty `BEYE` [dB] is defined as:

```
BEYE = 10*log10(eob2b/eo1)
```

where `eo1=max(min1-max0)` is the eye opening after propagation, being `min1` and `max0` the worst mark/space samples. `eob2b` is the eye opening in back to back configuration.
`PAT` contains the symbols pattern after decoding (see pat_decoder).

`X` is a structure whose fields are:

- `X.rec` = receiver type. Valid arguments are: `'ook','psbt','nf-dpsk'` to use receiver_ook, `'dpsk'` to use receiver_dpsk, `'dqpsk'` to use with receiver_dqpsk.
- `X.oftype` = optical filter type (see myfilter)
- `X.obw` = optical filter 3 dB bandwidth normalized to the bit rate
- `X.oord` = optical filter order (if `X.oftype` is `'supergaussian'`)

- `X.eftype` = electrical filter type (see myfilter)
- `X.ebw` = electrical filter 3-dB bandwidth normalized to the bit rate
- `X.eord` = electrical filter order (if `X.eftype` is `'supergaussian'`)
- `X.dpost` = post compensating fiber cumulated dispersion [ps/nm]: Vector of two elements `[d1 d2]` that is the range within it the best post compensation is searched. `X.dpost` can also be a scalar: in this case, the function returns the eye closure penalty at this value of `X.dpost`.
- `X.slopez` = post compensating fiber cumulated slope [ps/nm^2]
- `X.lambda` = wavelength [nm] at which the post compensating fiber has a cumulated dispersion equal to `X.dpost`.

Optional parameters for `X`:

- `X.tol` = tolerance of the golden search algorithm (fractional precision: +/-tol).
- `X.print`= structure for print. E.g. `X.print = {'nomefile','eye'}` or `X.print = {'nomefile', 'current'}`, prints to file `nomefile` the eye or the current, respectively. `nomefile` will be placed into `GSTATE.DIR` within a directory ending with `'.MOD'`.
- `X.plot = 'ploteye'`: plots the eye in the active figure; `'plotcur'` plots the received current.
- `X.color`= color string for the plot (see plot.m). E.g. `'b-'`.
- `X.ts` = Fixed sampling time (`-0.5 <= X.ts <= 0.5`).
- `X.comp` = component on which evaluate eye and calculate BER (dqpsk modulation only). Can be `'phase'` or `'quadrature'` or `'both'`. In the last case the function gets two measurements over the in-phase and quadrature components, sampled with the same clock time. `X.comp='both'` requires `PAT` to be a two-column matrix with the phase/quadrature binary patterns on column 1,2, respectively.
- `X.delay = 'theory'` means that the delay uses the theoretical delay saved within `GSTATE.DELAY` (see create_field). By default the delay is measured by a cross-correlation measurement between the received current and an artifical pulse amplitude modulation (PAM) signal with ideal non-return to zero bits with symbols equal to `PAT`. The correlation method is useful in presence of polarization mode dispersion (PMD). See corrdelay.

The receiver is composed of an ideal, purely linear, post compensating fiber + optical filter + optical to electrical converter + electrical lowpass filter. For example see receiver_ook or receiver_dpsk.

The best eye opening is searched through a golden search algorithm (see [22].)

Note 1: The golden search algorithm works when only one minimum is present within the range `[d1 d2]`. Otherwise the returned `BEYE` is just one of the local min, and may not be the lowest.

Note 2: This function works over a copy of the electric field. All fields of the global variable `GSTATE` are left unchanged.

2.29.3 See also

eval_eye, best_sp, pattern, myfilter, receiver_ook, receiver_dpsk, receiver_dqpsk , corrdelay

2.30 best_sp

Search algorithm for the best OSNR penalty vs. back-to-back.

2.30.1 Syntax

```
BSP=BEST_SP(ICH,X,PAT)
[BSP,BPOST]=BEST_SP(ICH,X,PAT)
[BSP,BPOST,WARN]=BEST_SP(ICH,X,PAT)
[BSP,BPOST,WARN,ECP]=BEST_SP(ICH,X,PAT)
```

2.30.2 Description

`BSP=BEST_SP(ICH,X,PAT)` searches the minimum osnr penalty `BSP` [dB], also called sensitivity penalty, of channel `ICH` vs. back to back transmission, by varying the post compensating fiber dispersion.
The function works for non-coherent transmissions. The target bit error rate (ber) of `BSP` is measured through the Karhunen-Loeve (kl) method (see ber_kl for more details).

`[BSP,BPOST,WARN,ECP]=BEST_SP(ICH,X,PAT)` also returns the best post-dispersion `BPOST` [ps/nm] that yields `BSP`. `WARN` is a flag equal to `1` when the function found a sensitivity penalty equal to `NaN` during the search of the optimal post, that corresponds to a possible wrong result. `ECP` is the eye closure penalty [dB] using `BPOST`. `PAT` is the symbols pattern.

`X` is a structure whose fields are:

- `X.rec` = receiver type. Valid arguments are: `'ook'`, `,'nf-dpsk'` to use receiver_ook, `'dpsk'` to use receiver_dpsk, `'dqpsk'` to use with receiver_dqpsk.
- `X.ber` = target ber at which the algorithm measures the OSNR penalty.
- `X.oftype` = optical filter type (see myfilter)
- `X.obw` = optical filter 3 dB bandwidth normalized to the bit rate
- `X.oord` = optical filter order (if `X.oftype` is `'supergaussian'`)
- `X.eftype` = electrical filter type (see myfilter)
- `X.ebw` = electrical filter 3-dB bandwidth normalized to the bit rate
- `X.eord` = electrical filter order (if `X.eftype` is `'supergaussian'`)
- `X.dpost` = post compensating fiber cumulated dispersion [ps/nm]: Vector of two elements `[d1 d2]` that is the range within it the best post compensation is searched. `X.dpost` can also be a scalar: in this case, the function returns the eye closure penalty at this value of `X.dpost`.
- `X.slopez` = post compensating fiber cumulated slope [ps/nm^2]
- `X.lambda` = wavelength [nm] at which the post compensating fiber has a cumulated dispersion equal to `X.dpost`.
- `X.osnr` = Optical signal to noise ratios (osnr), [dB], over which the ber is evaluated. The osnr is over a conventional bandwidth of 0.1 nm and is measured immediately before the receiver. `X.osnr` refers to `X.poln` noise polarizations. `X.osnr` is a vector.
- `X.poln` = Noise polarizations, `1` or `2`. Note: `X.poln` is independent from the signal polarizations, e.g. the algorithm can work with two noise polarizations and just one signal polarization.

`X` have also the following parameters required by the kl-method:

- `X.eta` = Bandwidth expansion factor. The kl method samples the signal and the noise up to a frequency equal to `X.eta` times the bandwidth of the optical filter. Usually is (faster) `1 < X.eta < 3` (slower).

- `X.mu` = Time expansion factor. The memory of the receiver is `X.mu` times the time duration of the memory devices inside the receiver, i.e. the optical/electrical filter. For DPSK there is an additional memory due to the Mach-Zehnder delay interferometer. The memory of such devices is approximated by the inverse of their bandwidth, as suggested in [20]. Usually is (faster) `1 < X.mu < 10` (slower).
- `X.saddle = 'yes'`: the ber is evaluated through the saddle point approximation (faster). `'no'`: the ber is evaluated by numerical integration of the moment generating function (slower, but more accurate).

For more details about `X.eta`, `X.mu` and `X.saddle` see ber_kl.

`X` can also have the optional parameters:

- `X.interp` = interpolation method for finding `X.ber`, see INTERP1. Default is `'spline'`.
- `X.extrap = 'yes'`: `X.ber` can be extrapolated outside `X.osnr`, see INTERP1. `'no'`: If `X.ber` is outside the range `X.osnr` the function returns `BPOST = NaN`, which is also the default strategy.
- `X.tol` = tolerance of the golden search algorithm (fractional precision: +/-tol).
- `X.plot = 'ploteye'`: plots the eye in the active figure; `'plotcur'` plots the received current.
- `X.color` = color string for the plot (see PLOT). E.g. `'b-'`.
- `X.print` = structure for print. E.g. `X.print = {'nomefile','eye'}` or `X.print = {'nomefile','current'}`, prints to file nomefile the eye or the current, respectively. `nomefile` will be place into `GSTATE.DIR` within a directory ending with `'.MOD'`.
- `X.delay = 'theory'` means that the delay uses the theoretical delay saved within `GSTATE.DELAY` (see create_field). By default the delay is measured by a cross-correlation measurement between the received current and an artificial pulse amplitude modulation (PAM) signal with ideal non-return to zero bits with symbols equal to `PAT`. The correlation method is useful in presence of polarization mode dispersion (PMD). See corrdelay.
- `X.ts` = Fixed sampling time (`-0.5 <= X.ts <= 0.5`).

The receiver is composed of an ideal, purely linear, post compensating fiber + optical filter + optical to electrical converter + electrical lowpass filter. For example see receiver_ook or receiver_dpsk.

The best ber is searched through a golden search algorithm (see [22]).

Note 1: The golden search algorithm works when only one minimum is present within the range `[d1 d2]`. Otherwise the returned `BSP` is just one of the local minima, and may not be the lowest.

Note 2: This function works over a copy of the electric field. All fields of the global variable `GSTATE` are left unchanged.

2.30.3 See also

ber_kl, best_eye, pattern, myfilter, receiver_ook, receiver_dpsk, receiver_dqpsk, eval_eye, corrdelay

2.31 ber_estimate

Bit-error rate estimate by Monte Carlo simulation

2.31.1 Syntax

```
[COND,AVGBER,NRUNS,STDBER]=BER_ESTIMATE(PAT_HAT,PAT,X)
[COND,AVGBER,NRUNS,STDBER]=BER_ESTIMATE(PAT_HAT,PAT,X,NIND)
```

2.31.2 Description

`[COND,AVGBER,NRUNS,STDBER]=BER_ESTIMATE(PAT_HAT,PAT,X)` estimates the average bit error rate (`AVGBER`) by standard Monte Carlo (MC) simulation.
`NRUNS` is the number of samples used in the estimation. `STDBER` is the standard deviation of the measured BER. `COND` is true during the MC cycle, and false when the convergence stop criterion has been reached. `PAT` is the bit pattern, decoded with pat_decoder if necessary, while `PAT_HAT` is the received estimated pattern given by samp2pat.

`X` is a struct with one or both of the following fields:

- `X.stop` = vector `[t1 t2]`. `t1` is the relative BER accuracy, while `t2` the Gaussian confidence of the accuracy. In such a case the MC simulation tests a sufficient number of runs as soon as the accuracy is reached. For instance, with `t1=0.01` and `t2=95`, the simulation ends as soon as the ratio

  `error/AVGBER<0.01`

  being error the estimated error on `AVGBER` with a Gaussian confidence of 95%. Beware that the concept of confidence works for Gaussian distributed random variables, while in the general case with few samples it is just a rule of thumb. A Gaussian confidence of 68% means that a Gaussian distributed BER is within +/- `STDBER` with probability 68%.

- `X.nmin` = minimum number of errors counted. In absence of `X.stop` this is also the overall number of errors counted. If not specified, the default value is `1`.

`[COND,AVGBER,NRUNS,STDBER]=BER_ESTIMATE(PAT_HAT,PAT,X,NIND)` runs a vectorial MC. In this case `X` must have the additional field:

- `X.dim` = dimension of `AVGBER` (external cycle dimension).

`COND,AVGBER,NRUNS,STDBER` are vectors of size `X.dim`, and they are currently evaluated in position (or index) `NIND`. Such options is useful for MC estimation inside a loop and allows to evaluate all the entries of `AVGBER` with the same accuracy given by `X.stop` and `X.nmin`. `COND` is a vector of logical. Example: MC estimation for a system by testing a vector of post-compensating fibers placed at the end of the link. See the examples for more information.

Note: A Monte Carlo estimation of the bit error rate requires that the sampling time is fixed, hence call eval_eye using the option `X.ts (see` eval_eye). Otherwise eval_eye may uses different sampling time from run to run due to different sampling time optimizations.

2.31.3 See also

eval_eye, ber_kl, pat_decoder

2.31.4 References

An exhaustive description of the Monte Carlo method can be found in [23]. Other significant contributions are in [24, 22]. A tutorial about the evaluation of the bit error rate in digital systems is in [25].

2.32 mc_estimate

Monte Carlo estimation of a random variable mean and variance

2.32.1 Syntax

```
[COND,OUT]=MC_ESTIMATE(S,X)
[COND,OUT]=MC_ESTIMATE(S,X,NIND)
```

2.32.2 Description

`[COND,OUT]=MC_ESTIMATE(S,X)` estimates the average value and variance of a random variable by standard Monte Carlo (MC) simulation.

`S` is the vector of random samples. `X` is a struct with one or both of the following fields:

- `X.stop` = vector `[t1 t2]`. `t1` is the relative accuracy of the estimator, while `t2` the Gaussian confidence of the accuracy. In such a case the MC simulation tests a sufficient number of runs as soon as the accuracy is reached. For instance, with `t1=0.01` and `t2=95`, the simulation ends as soon as the ratio `OUT.stdmean/OUT.mean < 0.01`, if `X.method = 'mean'`. Otherwise the simulation ends with `(O2-O1)/OUT.var < 0.01` if `X.method = 'var'` (see `OUT.varlim` next). In the previous example the Gaussian confidence was 95%. Beware that the concept of confidence works for Gaussian distributed random variables, while in the general case with few samples it is just a rule of thumb. A Gaussian confidence of 68% means that the exact mean of a Gaussian distributed `S` is within +/- `OUT.stdmean` of `OUT.mean` with probability 68%.
- `X.nmin` = minimum number of samples for applying `X.stop`. In absence of `X.stop` this is also the overall number of samples tested.
- `X.method` = `'mean'` means that `X.stop` is applied on the estimated average value. `'var'` means that `X.stop` is applied to the variance. Default is `'mean'`.

On output the function returns `COND` which is true during the MC cycle, and false when the convergence stop criterion has been reached.

`OUT` is a struct containing the results, i.e.:

- `OUT.mean` = estimated average value of `X`;
- `OUT.var` = estimated variance of `X`;
- `OUT.nruns` = overall number of samples tested;
- `OUT.stdmean` = standard deviation of `OUT.mean`. Note: this is not `sqrt(OUT.var)`, but the standard deviation of the average value. Hence, for increasing `OUT.nruns`, `OUT.stdmean` decreases making `OUT.mean` a good measure of the exact average value;
- `OUT.varlim = [O1 O2]`. The unknown exact variance satisfies: `O1` < exact variance < `O2` with confidence `X.stop(2)`. Note: `O1` and `O2` are functions of chi square percentiles. Since for `OUT.nruns > 50` they can be safely approximated with Gaussian percentiles we adopted such approximation (see [24]).

`[COND,OUT]=MC_ESTIMATE(S,X,NIND)` runs a vectorial MC. In this case `X` must have the additional field:

- `X.dim` = dimension of `OUT.mean` (external cycle dimension).

`COND,OUT.mean,OUT.nruns,OUT.var,OUT.stdmean` are vectors of size `X.dim`, and they are currently evaluated in position (or index) `NIND`. `S` can be a matrix or a vector. In the matrix case, all samples of column `NIND` are used to update the estimation of `OUT` at index `NIND`, while in the vector case the entire vector (row or column) is used to update index `NIND` of `OUT`. Such options is useful for MC estimation inside a loop and allows to evaluate all the entries of `OUT.mean` with the same accuracy given by `X.stop` and `X.nmin`. `COND` is a vector of logical.

`MC_ESTIMATE` is a generalization of ber_estimate.

2.32.3 See also

ber_estimate, eval_eye, ber_kl , pat_decoder

2.32.4 References

An exhaustive description of the Monte Carlo method can be found in [23]. Other significant contributions are in [24, 22].

2.33 cmaadaptivefilter

Polarization demultiplexing filter using CMA algorithm

2.33.1 Syntax

```
[Y H1 H2] = CMAADAPTIVEFILTER(XX, H1, H2, TAPS, MU, R, SPS)
```

2.33.2 Description

`[Y H1 H2] = CMAADAPTIVEFILTER(XX, H1, H2, TAPS, MU, R, SPS)` applies a matrix of adaptive filters whose initial stati are written in `H1` and `H2` matrices. The filters coefficients are updated with the constant modulus algorithm (CMA) proposed by [26].
The parameters of this algorithm are the number of taps `TAPS`, the radius `R`, the convergence parameter `MU` and the number of samples per symbol `SPS`. Normally there are four filters, let's call them `H11`, `H12`, `H21`, `H22` and the relation between the inputs and the outputs of this function is the following:

$$\begin{aligned} Y_1 &= X_1 \otimes H11 + X_2 \otimes H12 \\ Y_2 &= X_1 \otimes H21 + X_2 \otimes H22 \end{aligned}$$

where $\otimes$ is the convolution, X_1 and X_2 are the two inputs signals and Y_1 and Y_2 are the two output signals. Usually the filters Hxy have several coefficients (taps). In order to store all the filter a 3-dimensional matrix is needed. I propose the following order for the dimensions: `H(tap,y,x)` where `H` has the following size: 5x2x2 if the filters have 5 taps each.
Since Matlab is very slow with operations with such a kind of matrix I preferred to split `H` in two 2-dimensional matrices: `H1` and `H2` are the matrix you could virtually obtain by doing:

```
H1 = squeeze(H(:,1,:))

H2 = squeeze(H(:,2,:))
```

So that `H11 = H1(:,1)`, `H12 = H1(:,2)`, `H21 = H2(:,1)`, `H22 = H2(:,2)`

It is recommended to use the equivalent mex file of this function when possible. Try to compile `cmaadaptivefilter.c` by `mex cmaadaptivefilter.c`.

2.33.3 See Also

dsp4cohdec, easiadaptivefilter

2.33.4 References

The algorithm implemented by this function was proposed for the first time in [26].

2.34 dsp4cohdec

Digital signal processing for a coherent receiver

2.34.1 Syntax

`[PHASES AMPLITUDES]=DSP4COHDEC(ICH,PAT,X,P)`

2.34.2 Description

`[PHASES AMPLITUDES]=DSP4COHDEC(ICH,PAT,X,P)` returns the received phase (`PHASES`) and amplitude (`AMPLITUDES`) of a quadrature phase shift keying (QPSK) signal after digital signal processing (DSP). `ICH` is the channel number, `PAT` the pattern. The processing consists in several steps:

1. Analog to Digital Conversion (quantization)
2. Dispersion Compensation through FIR
3. Rebuilding of Complex Signal(s)
4. Digital Clock Recovery
5. Polarization Demultiplexing
6. Frequency Estimation using the Bell Labs algorithm [27]
7. Phase Estimation using the Viterbi&Viterbi algorithm [28]
8. Hard decision

`X` a struct whose fields are:

- `X.oftype` = optical filter type (see myfilter)
- `X.obw` = optical filter 3 dB bandwidth normalized to the symbol rate
- `X.oord` = optical filter order (if `X.oftype` is `'supergaussian'`)
- `X.eftype` = electrical filter type (see myfilter)
- `X.ebw` = electrical filter 3-dB bandwidth normalized to the symbol rate
- `X.eord` = electrical filter order (if `X.eftype` is `'supergaussian'`)

`X` can also have the optional parameters:

- `X.lodetuning` = Local Oscillator Detuning frequency [Hz].
- `X.lophasenoise` = Local Oscillator Phase Noise Vector [rad].
- `X.lolinewidth` = Local Oscillator Linewidth (Hz/symbolrate)
- `X.lopower` = Local Oscillator Power [dBm]

- `X.ts` = Fixed sampling time (`-0.5 <= X.ts <= 0.5`)
- `X.plot` = `'ploteye'`: plots the eye in the active figure; `'plotcur'` plots the received current
- `X.pol` = `'x','y','xy'`. Polarization of `X.plot`
- `X.color` = color string for the plot (see `PLOT`). E.g. `'b-'`
- `X.dpost` = post compensating fiber cumulated dispersion [ps/nm]
- `X.slopez` = post compensating fiber cumulated slope [ps/nm^2]
- `X.lambda` = wavelength [nm] at which the post compensating fiber has a cumulated dispersion equal to `X.dpost`
- `X.delay` = `'theory'` means that the delay uses the theoretical delay saved within `GSTATE.DELAY` (see creat_field). By default the delay is measured by a cross-correlation measurement between the received current and an artificial pulse amplitude modulation (PAM) signal with ideal non-return to zero bits with symbols equal to `PAT`. The correlation method is useful in presence of polarization mode dispersion (PMD)

`P` is a structure whose fields are:

- `P.baudrate` : Baud-Rate or symbol rate [GBaud]
- `P.workatbaudrate`: work with one (true) or two (false) samples/symb
- `P.sps` : samples per symbol in the `IRX` matrix
- `P.applyadc` : enable quantization (true/false)
- `P.adcbits` : number of bits of the ADC
- `P.applydcf` : enable dispersion compensation filter (true/false)
- `P.dispersion` : dispersion [ps/nm]
- `P.lambda` : wavelength of the channel [nm]
- `P.ndispsym` : dispersion compensating filter length [symbols]
- `P.applynlr` : enable nonlinear rotation (true/false)
- `P.nlralpha` : constant for non linear rotation algorithm
- `P.applypol` : enable polarization demux/tracking (true/false)
- `P.polmethod` : used method:
  - `'singlepol': Kikuchi` [29]
  - `'cma': Constant Modulus Algorithm (CMA)` [26]
  - `'easi': EASI` [30]
  - `'combo': COMBO: EASI+CMA`
- `P.cmaparams.R` : radius of CMA
- `P.cmaparams.mu` : convergence parameter of CMA
- `P.cmaparams.taps`: CMA filter taps
- `P.cmaparams.txpolars`: transmitted polarizations

- `P.cmaparams.phizero`: rotation angle between tx and rx field
- `P.easiparams.mu`: convergence parameter of EASI
- `P.easiparams.txpolars`: transmitted polarizations
- `P.easiparams.phizero`: rotation angle between tx and rx field
- `P.modorder` : modulation order: `BPSK`=1, `QPSK`=2
- `P.freqavg` : frequency estimation smoothing parameter [27]
- `P.phasavg` : phase estimation smoothing parameter [28]
- `P.poworder` : modulus power in phase estimation

Given `PHASES` ans `AMPLITUDES` the symbol constellation can be plotted % using the function `POLAR`:

```
POLAR(PHASES,AMPLITUDES)
```

2.34.3 See Also

cmaadaptivefilter, easiadaptivefilter, receiver_cohmix

2.34.4 References

The frequency and phase estimation algorithms employed in this function are described in [27] and [28]. Two of the possible polarization demultiplexing algorithms are described in [26, 30].

2.35 easiadaptivefilter

Source separation filter using EASI algorithm

2.35.1 Syntax

```
[Y H1 H2] = EASIADAPTIVEFILTER(XX, H1, H2, TAPS, MU, SPS )
```

2.35.2 Description

`[Y H1 H2] = EASIADAPTIVEFILTER(XX, H1, H2, TAPS, MU, SPS )` applies an adaptive matrix whose initial stati are written in `H1` and `H2`. The matrix elements are updated with the equivariant adaptive source separation via Independence (EASI) proposed in [30]. The parameters of this algorithm are the convergence parameter `MU` and the number of samples per symbol `SPS`. `TAPS` should be always set to 1. Normally there are four elements, let's call them `H11`, `H12`, `H21`, `H22` and the relation between the inputs and the outputs of this function is the following:

$$\begin{aligned} Y_1 &= X_1 \otimes H11 + X_2 \otimes H12 \\ Y_2 &= X_1 \otimes H21 + X_2 \otimes H22 \end{aligned}$$

where $\otimes$ is the convolution, X_1 and X_2 are the two inputs signals and Y_1 and Y_2 are the two output signals.

It is recommended to use the equivalent mex file of this function when possible. Try to compile `easiadaptivefilter.c` by `mex easiadaptivefilter.c`.

2.35.3 See Also

dsp4cohdec, cmaadaptivefilter

2.35.4 References

The algorithm implemented by this function was proposed for the first time in [30].

2.36 nmod

N-modulus of an integer.

2.36.1 Syntax

```
Y=NMOD(A,N)
```

2.36.2 Description

`Y=NMOD(A,N)` reduces the integer `A` into `1->N, mod N`.

2.36.3 Example

```
N=8;
A =[ -2 -1 0 1 2 3 4 5 6 7 8 9 10];
Y=nmod(A,N);
yields:  Y=[6 7 8 1 2 3 4 5 6 7 8 1 2]
```

2.37 fastshift

Fast but simplified circular shift.

2.37.1 Syntax

```
Y=FASTSHIFT(X,N)
```

2.37.2 Description

`Y=FASTSHIFT(X,N)` shifts the vector or matrix `X` circularly. If `X` is a vector then `Y=FASTSHIFT(X,N)` is equivalent to `Y=CIRCSHIFT(X,N)`.

2.37.3 Examples

```
X=[0 1 2 3 4 5 6 7 8 9 10];
Y=fastshift(X,2);
yields:  Y=[9 10 0 1 2 3 4 5 6 7 8]
```

If `X` is a matrix then FASTSHIFT acts on the first dimension (rows) and `Y=FASTSHIFT(X,N)` is equivalent to `Y=CIRCSHIFT(X,[N 0])`.

```
X = [1 2 3; 4 5 6;7 8 9 ]
Y=fastshift(X,2);
yields:  Y = [4 5 6 ;7 8 9;1 2 3 ]
```

2.37.4 See also

nmod

2.38 pow2phi

Convert power into nonlinear phase.

2.38.1 Syntax

`PHI=POW2PHI(PWR,L,ALPHA,GAM,G,NSPAN)`

2.38.2 Description

`PHI=POW2PHI(PWR,L,ALPHA,GAM,G,NSPAN)` yields the cumulated nonlinear phase `PHI` [rad] along the overall optical link by a signal of power `PWR` [mW]. `PHI` is the self-phase modulation (SPM) rotation induced by the link over a constant signal of power `PWR`.
`L` is a vector of size equal to the number of nonlinear fibers in the link and contains the fiber lengths in [m]. `ALPHA` are the fibers attenuation [dB/km], `GAM` are the fibers nonlinear coefficients, usually called gamma [1/W/m]. `G` is the net gain [dB] from fiber to fiber (see the example).

If `L,ALPHA,GAM,G` are of length `1`, the same value is used for a total of `NSPAN` spans.

2.38.3 Example

Two-fiber periodic link of `N` spans. One of the `NSPAN` periods is:

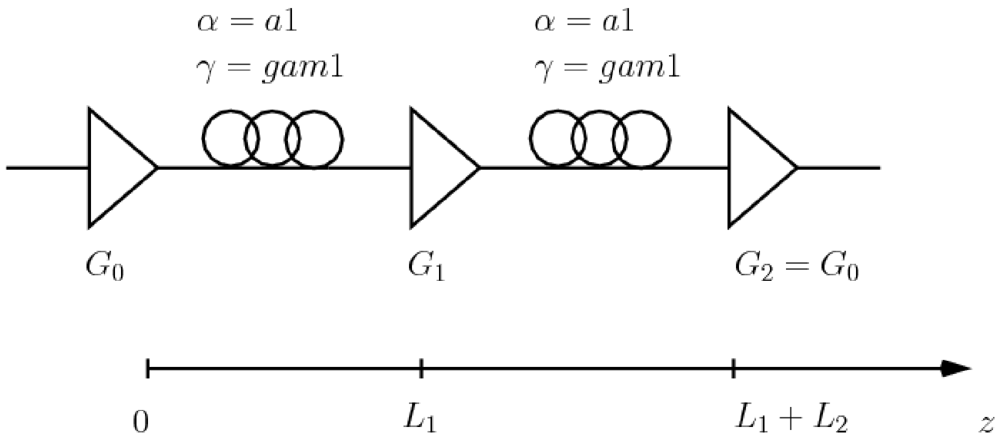

Figure 2.11: Single period of a periodical optical link.

Call the function with:
```
L = [L1,L2];
ALPHA = [a1,a2];
GAM = [gam1,gam2];
G=[G0,G1];
```

Note 1: If the laser is directly connected to the link, `G0=0` dB.

Note 2: Transparent link of `N` equal spans -> `L = L1, ALPHA=a1, G=0` and `NSPAN = N`.

The function returns the cumulated nonlinear phase `PHI` given by the power `PWR` [mW] entering the amplifier ampli0.

2.38.4 See also

phi2pow

2.39 phi2pow

Convert nonlinear phase into power.

2.39.1 Syntax

`PWR=PHI2POW(PHI,L,ALPHA,GAM,G,NSPAN)` yields the transmitted power `PWR` [mW] corresponding to a cumulated nonlinear phase into the overall optical link equal to `PHI` [rad]. `PHI` is the self-phase modulation (SPM) rotation induced by the link over a constant signal of power `PWR`.
`L` is a vector of size equal to the number of nonlinear fibers in the link and contains the fiber lengths in [m]. `ALPHA` are the fibers attenuation [dB/km], `GAM` are the fibers nonlinear coefficients, usually called gamma [1/W/m]. `G` is the net gain [dB] from fiber to fiber (see the example).

If `L,ALPHA,GAM,G` are of length `1`, the same value is used for a total of `NSPAN` spans.

2.39.2 Example

Two-fiber periodic link of `N` spans. One of the `NSPAN` periods is:

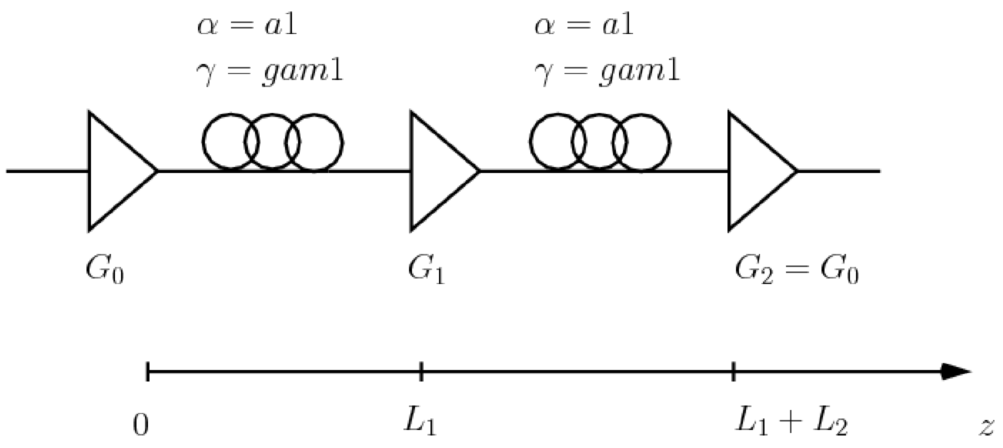

Figure 2.12: Single period of a periodical optical link.

Call the function with:
```
L = [L1,L2];
ALPHA = [a1,a2];
GAM = [gam1,gam2];
G=[G0,G1];
```

Note 1: If the laser is directly connected to the link, `G0=0` dB.

Note 2: Transparent link of `N` equal spans -> `L = L1, ALPHA=a1, G=0` and `NSPAN = N`.

The function returns `PWR` [mW] that entering the amplifier ampli0 gives the cumulated nonlinear phase `PHI`.

2.39.3 See also

pow2phi

2.40 avg_power

Evaluate the average power per symbol.

2.40.1 Syntax

```
E=AVG_POWER(ICH)
E=AVG_POWER(ICH,FLAG)
E=AVG_POWER(ICH,FLAG,FIL,BW,ORD)
```

2.40.2 Description

`E=AVG_POWER(ICH)` returns in `E` the average power [mW] per symbol of channel `ICH`.

`E=AVG_POWER(ICH,FLAG)` has the optional parameter `FLAG` that can be `'abs'` or `'norm'`. `FLAG='norm'` allows to return the power normalized to the transmitted peak power, otherwise with `FLAG='abs'` (default) the power is in [mW].

`E=AVG_POWER(ICH,FLAG,FIL,OBW,ORD)`temporary extracts the channel using the filter `FIL` with bandwidth `OBW` and order `ORD` (see myfilter).
The average power is evaluated in the frequency domain.
In the case of a unique field (see create_field) the function evaluates the power of channel `ICH` (e.g. `ICH` may be the one centered on f1 or f2 or f3 in the bottom figure) over the window D1, D2 or D3, respectively.

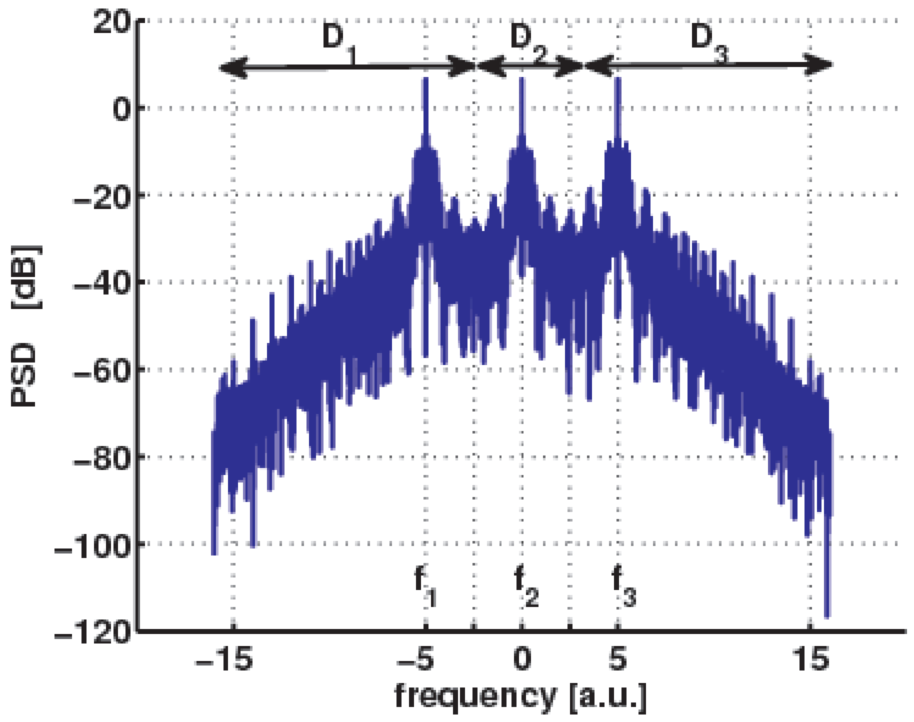

Figure 2.13: D_1,D_2, D_3 are the regions over which the average power is evaluated for channel 1,2,3, respectively.

`E` contains also the contribute of `GSTATE.FIELDY`, if it exists.

2.40.3 Example

An ideal On-off keying signal with non-return to zero pulses has `E=0.5` with `FLAG='norm'`.

2.41 corrdelay

System delay by cross-correlation measurement

2.41.1 Syntax

```
DELAY=CORRDELAY(IRIC,PAT,NT,NSYMB)
DELAY=CORRDELAY(IRIC,PAT,NT,NSYMB,OPT)
[DELAY,WRN,RHO]=CORRDELAY(IRIC,PAT,NT,NSYMB,OPT)
[DELAY,WRN,RHO,IRICA]=CORRDELAY(IRIC,PAT,NT,NSYMB,OPT)
```

2.41.2 Description

`DELAY=CORRDELAY(IRIC,PAT,NT,NSYMB)` evaluates the system delay `DELAY` (in symbols) between the electric signal that drives the transmitter laser and the received signal `IRIC`. For binary alphabets `IRIC` is a vector of real, while in multi-level formats `IRIC` is a complex vector. The transmitted electric signal is artificially created by repeating the pattern `PAT` in order to have `NT` points x symbol (see reset_all).
`NSYMB` is the number of symbols.
The cross-correlation is the convolution between the vector `conj(x)` (see example below) and `IRIC`.

This function measures the delay as the coordinate of the maximum of the cross-correlation. If the difference between the largest and the second maximum is lower than a threshold (initialized at the beginning of this function), the function prints a warning of low accuracy. Low accuracy is an indicator that the delay may be wrong.

`DELAY=CORRDELAY(IRIC,PAT,NT,NSYMB,OPT)` has the optional flag `OPT`.
If `OPT='phase'` `CORRDELAY` treats `IRIC` as a complex quaternary signal (e.g. QPSK signal).

`[DELAY,WRN,RHO]=CORRDELAY(IRIC,PAT,NT,NSYMB,OPT)` also returns the correlation coefficient `RHO` of the signal and the reference one.

`WRN` is a flag equal to true if the the delay is measured with low accuracy. Low accuracy may happen in presence of amplified spontaneous emission noise or with big distortions. In such cases, if possible, it is better to use the theoretical delay saved into global variable `GSTATE.DELAY`.

`[DELAY,WRN,RHO,IRICA]=CORRDELAY(IRIC,PAT,NT,NSYMB,OPT)` returns on `IRICA` the angle of the input `IRIC` after removing phase shift ambiguity (useful in multi-level formats).

2.41.3 Example

if `PAT='1101'` and `Nt=4` the reference transmitted signal turns out to be non-return to zero equal to:
`x=[1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1];`
The cross-correlation is the convolution between the vector `x` and `IRIC`. For multi-level formats `x` is complex like `IRIC`, e.g. `x=[exp(i*pi/4) exp(i*3/4*pi) ...]` for QPSK.

2.42 evaldelay

Evaluate the group-delay of the filter.

2.42.1 Syntax

`Y=EVALDELAY(FTYPE,BW)`

2.42.2 Description

`Y=EVALDELAY(FTYPE,BW)` evaluates the group delay of the filter `FTYPE` (see myfilter for available filters) having 3dB bandwidth `BW`.
Thanks to E. Forestieri for the filter expressions.

2.42.3 See Also

myfilter

2.43 myfilter

Filter device in the frequency domain.

2.43.1 Syntax

```
HF=MYFILTER(FTYPE,F,BW)
HF=MYFILTER(FTYPE,F,BW,ORD)
```

2.43.2 Description

`HF=MYFILTER(FTYPE,F,BW,ORD)` returns the filter `FTYPE` evaluated at frequencies `F` in the column vector `HF`. `BW` is the the 3-dB bandwidth of the filter (with exceptions, Note 2), with respect to the filtered signal power, i.e., `|HF|^2=0.5` at `F=+-BW`.

`FTYPE` can be a string of the following:

- `'movavg'` : Short term integrator (moving average) [see Note 2]
- `'gauss'` : Gaussian filter [see Note 3]
- `'butt2'` : Butterworth 2nd order
- `'butt4'` : Butterworth 4th order
- `'butt6'` : Butterworth 6th order
- `'ideal'` : Ideal filter
- `'bessel5'` : Bessel 5th order
- `'rc1'` : RC1 filter
- `'rc2'` : RC2 filter
- `'supergauss'`: Super-Gaussian filter of order ORD
- `'gauss_off'` : Gaussian filter with offset ORD from the center of the channel

Note 1: The bandwidth `BW` is a lowpass bandwidth -> For optical bandpass filters having 3dB bandwidth `Bo`, it is `BW=Bo/2`.

Note 2: For the moving average `BW` is not the 3dB bandwidth, but the first zero of the sinc, i.e. `1/BW` is the duration of the moving average. The 3dB bandwidth is `0.443*BW`.

Note 3: For the Gaussian filter the bandwidth 1/e (`Be`) is related to the 3dB bandwidth by `Be= 1.6986*BW`

Note 4: If the frequency is normalized, the 3dB bandwidth must be normalized as well to the same value.

Note 5: For future new implementations of special filters, use the variable `ORD` for the new filter parameters.

Thanks to E. Forestieri for the filter expressions.

2.44 lpfilter

Filtering with a lowpass filter.

2.44.1 Syntax

```
EOUT=LPFILTER(EIN,FTYPE,BW)
EOUT=LPFILTER(EIN,FTYPE,BW,ORD)
```

2.44.2 Description

`EOUT=LPFILTER(EIN,FTYPE,BW)` filters the input field `EIN` with the filter `FTYPE` (see myfilter) having 3-dB bandwidth `BW` (normalized to the symbolrate). The resulting filtered field is associated to `EOUT`

`EOUT=LPFILTER(EIN,FTYPE,BW,ORD)` use the additional parameter `ORD` for special filter. E.g. `ORD` is the supergauss order for the supergaussian filter (see myfilter)

2.44.3 See Also

myfilter

2.45 pol_scrambler

Rotates the SOP of signal samples on the Poincaré sphere.

2.45.1 Syntax

```
POL_SCRAMBLER(TYPE, COH_TIMER, THETA, EPSILON, DELPHI)
```

2.45.2 Description

`POL_SCRAMBLER(TYPE, COH_TIMER, THETA, EPSILON, DELPHI)` rotates the signal State Of Polarization (SOP) on the Poincaré sphere.

- If `TYPE = 'rand'` then a random rotation is used, so that the signal SOP is uniformly scattered on the Poincaré sphere, as is typical of "long" SMF fibers. The rotation parameters are changed randomly every `COH_TIMER` symbol periods. For the method used to randomize rotations, see eqs.(13-15) and following observation in [31]. `COH_TIMER` is the "coherence time" of the scrambler, normalized to the symbol interval (i.e., multiplied by the symbol rate "R"). Use `COH_TIMER= 1/NT` to scramble each sample independently and produce a DOP that approaches zero; `COH_TIMER= NSYMB` to scramble all samples equally and preserve the original DOP (`NSYMB`=# of transmitted symbols; `NT`=# of samples per symbol).
- If `TYPE = 'fixed'` then `THETA`, `EPSILON` and `DELPHI` should be provided: the signal SOP is then rotated by a fixed amount `DELPHI` [rad.] (the Jones matrix "retardation") about a fixed rotation axis (the Jones matrix "eigenmode") with azimuth `THETA` and ellipticity `EPSILON`. `COH_TIMER` is ignored in this case.

2.45.3 See Also

set_sop, dop_meter

2.46 dop_meter

Computes the Degree Of Polarization of the Optical field.

2.46.1 Syntax

```
DOP=DOP_METER(ICH)
DOP=DOP_METER(ICH,NFIG_FLAG)
DOP=DOP_METER(ICH,NFIG_FLAG,FIL,BW,ORD)
[DOP,PHI]=DOP_METER(ICH,NFIG_FLAG,FIL,BW,ORD)
```

2.46.2 Description

`DOP=DOP_METER(ICH)` returns the degree of polarization (DOP) of partially polarized light. `ICH` is the channel number.

`DOP=DOP_METER(ICH,NFIG_FLAG)` with NFIG_FLAG=1 indicates that the states of polarization (SOP) of channel `ICH` will be plotted in the current figure.

`DOP=DOP_METER(ICH,NFIG_FLAG)` with `NFIG_FLAG~=0` or nonempty indicates that the SOP of channel `ICH` will be plotted in the current figure. `NFIG_FLAG` can be a char, e.g. `NFIG_FLAG='b'`, which means that the SOP plot will use that color for each sample. Otherwise, `NFIG_FLAG` can be a struct with fields:

- `NFIG_FLAG.col` = color of plot (e.g. `'b'`)
- `NFIG_FLAG.avgcol` = color of the average DOP (e.g. `'r'`)

The SOP shows the state of partially polarized light as a cloud of magenta points (one for each time sample) surrounding the "polarized component" of the field, shown with a black vector. The Poincaré sphere is plotted as a reference. As extreme case, the black circle is in the origin for unpolarized light (natural, light, ASE noise...), since the polarized component is null, while the magenta dots all collapse on the black circle for totally polarized light, since there is no unpolarized component.

`DOP=DOP_METER(ICH,NFIG_FLAG,FIL,BW,ORD)` works for a unique field (see create_field) and temporary extracts channel `ICH` with an optical filter `FIL` of bandwidth `BW` and optional order `ORD` (see myfilter).

`[DOP,PHI]=DOP_METER(ICH,NFIG_FLAG,FIL,BW)` also returns in `PHI.azi` and `PHI.ell` the azimuth and ellipticity of the average SOP.

2.46.3 See Also

pol_scrambler, set_sop

2.47 set_sop

Sets the average State Of Polarization of the transmitted signal.

2.47.1 Syntax

```
[EX,EY]=SET_SOP(EX,EY,ANG1,ANG2)
[EX,EY]=SET_SOP(EX,EY,ANG1,ANG2,ANGTYPE)
[EX,EY,MAT]=SET_SOP(EX,EY,ANG1,ANG2)
[EX,EY]=SET_SOP(EX,EY,ANG1,ANG2,RTYPE)
```

2.47.2 Description

`[EX,EY]=SET_SOP(EX,EY,ANG1,ANG2)` sets the State Of Polarization (SOP) of the electric field having x component `EX` and y component `EY`. `EX` and `EY` are column vectors (see mz_modulator). `ANG1`, `ANG2` are the azimuth and ellipticity [rad] of the output SOP, respectively.

`[EX,EY]=SET_SOP(EX,EY,ANG1,ANG2,ANGTYPE)` specifies `ANG1` and `ANG2` according to `ANGTYPE`. Available options are:

- `ANGTYPE = 'aziell'`; user specifies azimuth and ellipticity angles. Latitude and longitude on the Poincaré sphere are equal to `2*ANG2` and `2*ANG1`, respectively. Note: azimuth, normally in $[-\frac{\pi}{2};\frac{\pi}{2}]$, is interpreted modulo π; ellipticity, normally in $[-\frac{\pi}{4};\frac{\pi}{4}]$, is interpreted modulo $\frac{\pi}{2}$;
- `ANGTYPE = 'aarphd'`; user specifies the "absolute amplitude ratio" angle ($=\arctan\left(\left|\frac{E_y}{E_x}\right|\right)$) and the phase difference between field components ($=\arg\left(E_y \cdot E_x^*\right)$). Note: absolute amplitude ratio, normally in $[0;\frac{\pi}{2}]$, is interpreted modulo $\frac{\pi}{2}$; phase difference, normally in $[-\pi;\pi]$, is interpreted modulo 2π.

`[EX,EY,MAT]=SET_SOP(EX,EY,ANG1,ANG2)` also returns in `MAT` the unitary matrix that rotates the SOP. `MAT` can be used by inverse_pmd.
`[EX,EY]=SET_SOP(EX,EY,ANG1,ANG2,RTYPE)` with `RTYPE='mean'` change the default behavior by setting the output SOP with `ANG1` and `ANG2` rotations relatively to a reference system aligned with the input average SOP. The degree of polarization (DOP) is preserved.

2.47.3 Examples

An ideal polarization division multiplexed signal (PDM) signal before `SET_SOP` lies in the plane (s_2, s_3), because the X polarization carries the same power as the Y polarization. Calling:

```
[EX,EY]=SET_SOP(EX,EY,pi/2,pi/4)
```

let it lie in the plane (s_1, s_2) . See Section 3.2.1 for more information about the signals Poincarè description.

2.47.4 See Also

dop_meter, mz_modulator

2.48 polarizer

Linear optical polarizer

2.48.1 Syntax

```
POLARIZER(ANG1,ANG2)
POLARIZER(ANG1,ANG2,ANGTYPE)
POLARIZER(ANG1,ANG2,ANGTYPE,T)
```

2.48.2 Description

`POLARIZER(ANG1,ANG2)` polarizes the electric field contained in `GSTATE` (see reset_all) into the direction having azimuth `ANG1` and ellipticity `ANG2`.

`POLARIZER(ANG1,ANG2,ANGTYPE)` has the additional `ANGTYPE` selecting the type of angular coordinates used to specify the position of the state of polarization (SOP) on the Poincaré sphere. Available options are:

- `ANGTYPE = 'aziell'`; user specifies azimuth and ellipticity angles. Latitude and longitude on the Poincaré sphere equal `2*ANG2` and `2*ANG1`, respectively. NOTE: azimuth, normally in `[-pi/2;pi/2]`, is interpreted modulo pi; ellipticity, normally in `[-pi/4;pi/4]`, is interpreted modulo pi/2;

- `ANGTYPE = 'aarphd'`; user specifies the "absolute amplitude ratio" `atan(abs(EY/EX))` and the phase difference between field components `arg(EY*conj(EX))`. In this function it is `EX=GSTATE.FIELDX` and `EY=GSTATE.FIELDY`. NOTE: absolute amplitude ratio, normally in `[0;pi/2]`, is interpreted modulo pi/2; phase difference, normally in `[-pi;pi]`, is interpreted modulo 2*pi.

`POLARIZER(ANG1,ANG2,ANGTYPE,T)` indicates in `T = [TM, Tm]` the power transmittance [32] of the polarizer, being `TM` the maximum power flow and `Tm` the minimum power flow along the polarizer's axes. `TM` and `Tm` must be within 0 and 1.

2.48.3 See also:

pol_scrambler, set_sop, dop_meter , reset_all

2.49 plotfield

Plot the optical field.

2.49.1 Syntax

```
PLOTFIELD(POL,ICH,FLAG)
PLOTFIELD(POL,ICH,FLAG,COL,FIL,BW,ORD)
```

2.49.2 Description

`PLOTFIELD(POL,ICH,FLAG)` plots the power and/or phase of channel `ICH` contained into the global variable `GSTATE.FIELDX` if `POL='x'`, `GSTATE.FIELDY` if `POL='y'`, or both if `POL='xy'`. `POL` can also be `'tot'` and indicates to plot the overall power in the time domain, or the overall power of the spectrum. Such options works only with `FLAG='p---'`, `'n---'`, `'--p-'` or `'--n-'` see later.

`FLAG` is a 4-char string indicating the type of plot:

- `FLAG(1)= 'p'`: power in the time domain (normalized to the symbol time). Use `'n'` if you want the power normalized to the transmitted peak power of channel `ICH`. See Time domain representation for more details about the description of time in Optilux.
- `FLAG(2)= 'a'`: angle (phase) in the same time domain
- `FLAG(3)= 'p'`: `10*log10(abs(.)^2)` of the spectrum (FFT) in the frequency domain (normalized to the symbol rate). Use `'n'` for a spectrum normalized to the transmitted peak power of channel `ICH`. See Frequency domain representation for more details about the description of frequency in Optilux.
- `FLAG(4)= 'a'`: angle (phase) of the spectrum in the same frequency domain.
- A specific char of `FLAG` should be set to `'-'` to avoid its plot.

`PLOTFIELD(POL,ICH,FLAG,COL,FIL,BW,ORD)` With this call the specific channel `ICH` can be temporary extracted using the filter `FIL` having 3-dB bandwidth `BW` and order `ORD` (see myfilter). Such a requirement is necessary when the fields are combined into a unique field (see create_field). Otherwise, if `ICH=1` and `FIL` and `BW` are not indicated, all the complete unique field is plotted. The option `POL='tot'` is unactive with `FIL` present .

`COL` is the color of the lines (see PLOT. E.g. `COL='b-','r-x'`,etc)

2.49.3 Examples

Plot only the power in time and the phase of the spectrum: `FLAG = 'p--a'`
Plot both power and phase in time and frequency: `FLAG = 'papa'`

2.49.4 See also

printfield

2.50 plotfile

Plot file from disk.

2.50.1 Syntax

`PLOTFILE(FILE)`

2.50.2 Description

`PLOTFILE(FILE)` plot the double-column data stored in the file named `FILE` from disk. `FILE` can be a compressed file (with extension `.gz,.bz2,.zip`).

2.50.3 See also

printfield

2.51 printfield

Print the optical field to file.

2.51.1 Syntax

```
PRINTFIELD(POL,ICH,NAME,FLAG)
PRINTFIELD(POL,ICH,NAME,FLAG,NCYCLE,FIL,BW)
```

2.51.2 Description

`PRINTFIELD(POL,ICH,NAME,FLAG)` prints the electric field of channel `ICH` into the file `NAME` in the global directory `GSTATE.DIR`. `POL` can be either `'x'` for x-polarization, `'y'` for y-polarization or `'xy'` for printing both polarizations.
`FLAG` is a string that indicates the type of files to be printed. It can be `'papa'` in the most complete case, indicating that it will be printed both the power and angle (phase) in the time domain (first two chars of `FLAG`) and the power (`abs(.)^2`) and phase of the spectrum (last two chars of `FLAG`). If you don't want to print a specific component set it to `'-'`.

`PRINTFIELD(POL,ICH,NAME,FLAG,NCYCLE,FIL,BW)` `NCYCLE` is a counter that will be added to the output file name, and can be useful in presence of an external cycle to the function.
If all channels are combined into a unique field, you can temporary extract the `ICH` channel with the filter `FIL` having 3-dB bandwidth `BW` (see myfilter). In this case and in absence of an external cycle, leave `NCYCLE=[]`. However, if only the spectrum is requested in `FLAG, ICH` is ignored and all the complete spectrum is saved into one file.
`PRINTFIELD(POL,ICH,NAME,FLAG,NCYCLE,FIL,BW,ORD)` use the special parameter `ORD` for special filters. E.g. `ORD` is the supergauss order for the supergaussian filter (see myfilter).

The power will be printed in a directory (see reset_all) with the suffix ".MOD", while the phase with the suffix ".ANG".

2.51.3 Example 1

Print only the power in time and the phase of the spectrum: `FLAG = 'p--a'`

2.51.4 Example 2

```
POL='x';
ICH=1;
NCYCLE=81;
FLAG='p---';
NAME='Tx';
printfield(POL,ICH,NAME,FLAG,NCYCLE) % creates tempx_ch01_Tx_Cycle_0081
```

Note: `PRINTFIELD` presumes that the user initialized `GSTATE.DIR` in reset_all. If not, a temporary directory is used and a warning is displayed.

2.51.5 See Also

plotfield, reset_all

2.52 ber2q

Convert the bit-error rate in Q-factor

2.52.1 Syntax

```
Q=BER2Q(BER)
```

2.52.2 Description

`Q=BER2Q(BER)` converts the bit error rate `BER` in the `Q` factor [dB] using the formula:

```
Q=20log10(sqrt(2)*erfcinv(2*BER));
```

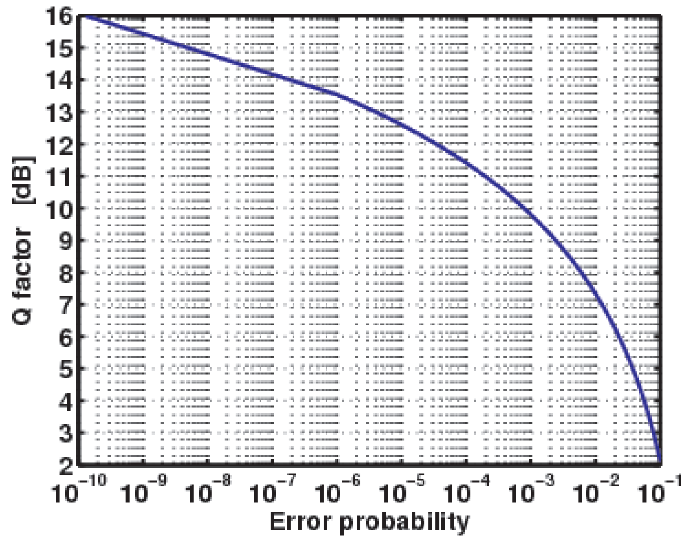

Figure 2.14: Relation between the error probability and the Q factor [dB].

2.53 mdoc

Display Optilux HTML documentation in the browser

2.53.1 Syntax

```
MDOC FUNC
MDOC('FUNC')
```

2.53.2 Description

`MDOC FUNC` displays the HTML documentation for the function `FUNC`. Under Octave the default browser is konqueror, under Matlab is the help browser. Such a default behavior can be simply changed inside the function.

Tip: the TAB completion under Matlab is not active for user defined functions. A simple trick is to call the function as for `DOC`, then, when the syntax is ready for a carriage return, press the `Home-key` (or `CTRL+A`) so that `DOC` can be changed in `MDOC` after digiting `M`.

Note: This function assumes that the Optilux documentation is in the default (original) doc directory relatively to the m-files.

2.54 fprintmsg

Write a message into the file `simul_out`.

2.54.1 Syntax

`FPRINTMSG(STR)`

2.54.2 Description

`FPRINTFMSG` Write a message into the file `simul_out`. `FPRINTMSG(STR)` print the message `STR` into the file `simul_out` using the rules of `FPRINTF (\n,\t,...)`.

2.55 checkfields

Check for valid input fields

2.55.1 Syntax

`CHECKFIELDS(X,ALLFIELDS)`

2.55.2 Description

`CHECKFIELDS(X,ALLFIELDS)` checks if the struct variable `X` has valid fields. `ALLFIELDS` is a cell variable containing all possible valid fields of `X`. Each element of the cell is a string. The check is case insensitive.

Note: Use `CHECKFIELDS` only for low level functions. If a function calls a subfunction, `CHECKFIELDS` should be run only in the nested subfunction. Do not abuse the function use. Beware.

2.56 offmat

Run Matlab/Octave simulation offline

2.56.1 Syntax

`offmat [OPTIONS] FILE.m`

2.56.2 Description

offmat is a bash script for running Matlab/Octave simulations offline using `nohup` under Linux-based operating systems. This way the user can logout from the host leaving the simulation still running. `OPTIONS` can be any of the following:

- `-s` save simulation results in `FILE.mat`
- `-p PATH` add `PATH` to the matlab/octave search path
- `-c LICENSEFILE` Set location of the license file that MATLAB should use. See `matlab -h`
- `-o OFILE` save simulation results in `OFILE.mat`
- `-O` force to use Octave
- `-M` force to use Matlab
- `-w` write information about the simulation in `/tmp/$USER_offmat`
- `-m USER@addr` Send a notification mail to `USER@addr` after the simulation
- `-n VALN` Use nice value `VALN`

- `-h` display help message and exit

With the option `-m` the program requires `mailx` and a `mail` sender installed and properly configured. The script by default run first matlab, then octave if matlab failed.

This function uses nohup to run the process in background. All screen messages will be written into a file called `FILE.nohup`. If such file already exists, the output file will be `FILE_n.nohup`, being `n` an incremental integer until `FILE_n.nohup` is a new file.

Note: This script assumes that your binary matlab calls `'matlab'` and your binary octave calls `'octave'`. If not, please manually change the first lines in this script under the voice "Matlab/Octave binary names".

2.57 fastexp

Calculates exp(i * x) quickly

2.57.1 Syntax

`Y=FASTEXP(X)`

2.57.2 Description

`Y=FASTEXP(X)` It's a fast computation of the expression: `Y = EXP( i * X )`. If you have compiled the file `fastexp.c` with mex then Matlab/Octave will run the still quicker (or quickest) `fastexp.mexglx`.

2.58 saddle

Evaluate the MGF saddle point

2.58.1 Syntax

`[U0,NSW] = SADDLE(SGN,XI,DX,DSG2,QSG2,QSG4,VARS,LD,LDEX,LD2,B2,POL2,NSYMB)`

2.58.2 Description

`[U0,NSW] = SADDLE(SGN,XI,DX,DSG2,QSG2,QSG4,VARS,LD,LDEX,LD2,B2,POL2,NSYMB)` returns the real saddle point `U0` for all bits `NSYMB` of the moment generating function (MGF) of the sampled signal.

`NSW` is a control parameter equal to:

- `NSW=0` if the saddle point has been evaluated with the desired accuracy
- `NSW=1` if the function did not found the saddle point
- `NSW=2` if the function found a saddle point with low accuracy

`SGN` is a flag equal to `1` for polar bit-formats, while for unipolar formats it takes `1` for spaces and `-1` for marks.
`XI` is (22) of [20], `DX` is `XI-eta`, being `eta` eq.(19) of [20].
`DSG2,QSG2,QSG4` are respectively `2*sigma^2,4*sigma^2,4*sigma^4`, being `sigma^2` the noise variance of the sampled signal.
`VARS` is (20) of [20]. `LD` and `LD2` are the eigenvalues, eq.(A.22) of [20], and the square of the eigenvalues, respectively.

`LDEX=[min(LD),max(LD)]`.
`B2` is (A.26) of [20], and here is of size (number of eigenvalues, number of symbols).
`POL2` is the number of noise polarizations.
`NSYMB` is the number of symbols.

The function evaluates the saddle point by the Newton-Rhapson method [22] along the lines described in [20]. Note that some input parameters are actually not necessary, but they were inserted for speed reasons. The accuracy of the routine is taken under control through some parameters at the beginning of the function. It is recommended to use the equivalent mex file of this function when possible. Try to compile `saddle.c` by `mex saddle.c.`
If the mex is created successfully, matlab/octave work with it and discards such function.

2.58.3 References

This function implements the algorithm proposed by E. Forestieri in [20]. Further notes about the saddle point can be found in [33].

2.59 comp_mex

compile all .c files into the directory

2.59.1 Syntax

`COMP_MEX`

2.59.2 Description

`COMP_MEX` is a simple routine to create `.mex` files starting from the `.c` files. Please, read carefully the matlab/octave support guide for the options of the mex function.

Chapter 3

Background

This chapter presents the basic properties of an optical propagation and the mathematics behind them.

3.1 NLSE

The nonlinear Schrödinger equation (NLSE) in absence of polarization effects for an electric field $A(z,t)$ [$\sqrt{\mathrm{W}}$], z being the distance [m] and t the time [s], in engineering notation is the following partial differential equation (PDE) [1]:

$$\frac{\partial A(z,t)}{\partial z} = -\frac{\alpha}{2}A - \beta_1\frac{\partial A}{\partial t} + j\frac{\beta_2}{2}\frac{\partial^2 A}{\partial t^2} + \frac{\beta_3}{6}\frac{\partial^3 A}{\partial t^3} - j\gamma\left|A\right|^2 A \tag{3.1}$$

where j is the imaginary unit; α is the fiber attenuation; $\beta_k = \left.\frac{\mathrm{d}^k\beta}{\mathrm{d}\omega^k}\right|_{\omega=\omega_0}$, $k = 1, 2, \ldots$, being $\beta(\omega)$ the wave propagation constant and $\omega_0 = 2\pi f_0 = 2\pi c/\lambda_0$ with f_0, λ_0 the central frequency/wavelength of $A(z,t)$, respectively, c being the speed of light; γ is the nonlinear coefficient. Such parameters satisfy the following relations.

3.1.1 Attenuation

The attenuation α [m^{-1}] is a measure of the power loss along the distance. Assuming all parameters zero except α, the NLSE (3.1) becomes:

$$\frac{\partial A}{\partial z} = -\frac{\alpha}{2}A$$

whose solution is $A(z,t) = A(0,t)e^{-\frac{\alpha}{2}z}$. In Optilux α is assumed constant over the signals spectrum even if it actually depends on the wavelength.
The attenuation is usually expressed in [dB/km]. In a dB scale the power loss can be measured as:

$$10\log_{10}\left(\left|A(z,t)\right|^2\right) - 10\log_{10}\left(\left|A(0,t)\right|^2\right) = \alpha z \cdot 10\log_{10} e$$

Hence, the relation between the attenuation in [m^{-1}] and the attenuation in [dB/km] is the following:

$$\alpha \quad [\mathrm{m}^{-1}] = \frac{\alpha \quad [\mathrm{dB/km}]}{\log_{10} e} \cdot 10^{-4} = 10^{-4} \cdot \ln(10)\,\alpha \quad [\mathrm{dB/km}]$$

To the attenuation can be associated the attenuation length $L_A = 1/\alpha$ as a measure of the distance over which the loss effect is significant. For a typical system having $\alpha = 0.2$ dB/km it is $L_A = 21715$ [m].

3.1.2 Group velocity

$\beta_1 = \frac{d\beta}{d\omega}\Big|_{\omega=\omega_0} = 1/v_g\left(\omega_0\right)$ [s/m] accounts for the group velocity v_g of the signal along the fiber, and hence is a delay per unit length. In presence of only β_1 the NLSE writes as:

$$\frac{\partial A}{\partial z} = -\beta_1 \frac{\partial A}{\partial t}$$

whose solution is:

$$A(z,t) = A\left(0, t - \frac{z}{v_g}\right)$$

Two signals centered at wavelengths λ_1 and λ_2 generally have different group velocities v_{g1} and v_{g2}, respectively, and hence travel at different speed. The delay per unit length between the two signals is the walk-off parameter d_{12} equal to:

$$d_{12} = \frac{1}{v_{g1}} - \frac{1}{v_{g2}} \tag{3.2}$$

$d_{12} > 0$ means that the channel having group velocity v_{g1} travels slower than the other. The walk-off weights the impact of the XPM and FWM effects.
To the walk-off can be associated the walk-off length $L_W = T_0/d_{12}$, being T_0 a reference time, generally the symbol time.

3.1.3 GVD parameters

The GVD parameters are the derivatives of order greater than 1 of $\beta(\omega)$ at $\omega = \omega_0$. Usually only β_2 [s^2/m] and β_3 [s^3/m] are included in the NLSE. Assuming all parameters zero except $\beta_{1,2,3}$ the NLSE becomes a linear PDE and writes as:

$$\frac{\partial A}{\partial z} = -\beta_1 \frac{\partial A}{\partial t} + j\frac{\beta_2}{2}\frac{\partial^2 A}{\partial t^2} + \frac{\beta_3}{6}\frac{\partial^3 A}{\partial t^3}$$

which writes in a simple form in the frequency domain $\widetilde{A}(z,\omega) = \mathcal{F}\left\{A(z,t)\right\}$:

$$\frac{\partial \widetilde{A}}{\partial z} = -j\left(\beta_1\omega + \frac{\beta_2}{2}\omega^2 + \frac{\beta_3}{6}\omega^3\right)\widetilde{A} \tag{3.3}$$

whose solution is:

$$\widetilde{A}(z,\omega) = \widetilde{A}(0,\omega)e^{-j\left(\beta_1\omega + \frac{\beta_2}{2}\omega^2 + \frac{\beta_3}{6}\omega^3\right)z} \tag{3.4}$$

Note from (3.4) that, being the system linear, the behavior of each frequency along the fiber depends only by itself. Since in (3.3) the loss is absent, for the energy conservation principle the energy carried by frequency ω must remain unaltered, i.e. $\left|\widetilde{A}(z,\omega)\right|^2 = \left|\widetilde{A}(0,\omega)\right|^2$, so that the GVD parameters induce a pure phase rotation in the frequency domain.
Most of the times β_2 and β_3 are expressed as functions of the wavelength through the fiber dispersion, $D = \frac{d\beta_1}{d\lambda}\Big|_{\lambda=\lambda_0}$, and through the fiber dispersion slope, $D' = \frac{d^2\beta_1}{d\lambda^2}\Big|_{\lambda=\lambda_0}$, where $\lambda_0 = c/f_0$. The following relations hold:

$$\begin{aligned}\beta_2 &= -\frac{\lambda_0^2}{2\pi c}D \\ \beta_3 &= \left(\frac{\lambda_0}{2\pi c}\right)^2\left(2\lambda_0 D + \lambda_0^2 D'\right)\end{aligned}$$

Note that the third order dispersion β_3 exists even with zero slope.
The dispersion D is usually expressed in [ps/(nm·km)] while the dispersion slope D' in [ps^2/(nm·km)]. As a reference, at $\lambda_0 = 1550$ nm it is $\beta_2 \quad [\text{s}^2/\text{m}] =\!-\!-1.2754 \cdot 10^{-27} \cdot D \quad [\text{ps}/(\text{nm}\cdot\text{km})]$.

$D > 0$ ($\beta_2 < 0$) corresponds to anomalous dispersion, $D < 0$ ($\beta_2 > 0$) corresponds to normal dispersion.
To the GVD parameters can be associated the dispersion length, $L_D = \beta_2/T_0^2$, and the dispersion slope length, $L_D^{'} = \beta_3/T_0^3$, being T_0 a reference time, generally the symbol time. The previous lengths have signs, hence sometimes the absolute value is used. For instance, the power of a Gaussian pulse doubles its $1/e$ width after propagating over a length of L_D [1].

3.1.4 Nonlinear coefficient

The nonlinear coefficient γ [1/(W·km)] is due to the Kerr effect of the fiber. The relation between the nonlinear coefficient and the fiber nonlinear index n_2 [m^2/W] is the following:

$$\gamma = \frac{2\pi n_2}{\lambda_0 A_{\text{eff}}} \tag{3.5}$$

being A_{eff} [μm^2] the fiber effective area. As a reference, a single mode fiber (SMF) has almost $A_{\text{eff}} = 80\mu\text{m}^2$ and $n_2 = 2.7 \cdot 10^{-20}$ [m^2/W] thus giving $\gamma = 1.37$ [1/(W · km)] @ $\lambda_0 = 1550$ nm, while for a dispersion compensating fiber (DCF) the value is usually four times larger due to a reduced effective area. To the nonlinear coefficient can be associated the nonlinear length $L_{\text{NL}} = 1/(\gamma \cdot P)$, being P a reference power, usually the transmitted signal peak power. A direct comparison between the nonlinear length and the dispersion length allows to deduce the propagation regime inside the optical fiber. $L_{\text{NL}} \gg L_D$ implies propagation in the dispersion, or purely linear, regime; on the opposite with $L_{\text{NL}} \ll L_D$ the propagation is in the nonlinear regime.
In the nonlinear regime the NLSE writes as:

$$\frac{\partial A}{\partial z} = -\frac{\alpha}{2}A - j\gamma\left|A\right|^2 A$$

whose solution is:

$$A(z,t) = A(0,t)e^{-\frac{\alpha}{2}z - j\gamma|A(0,t)|^2 L_{\text{eff}}(z)} = A(0,t)e^{-\frac{\alpha}{2}z}e^{-j\Phi(z,t)} \tag{3.6}$$

where $\Phi(z,t)$ is the SPM nonlinear phase rotation, while $L_{\text{eff}}(z)$ is the effective length up to coordinate z and is equal to:

$$L_{\text{eff}}(z) = \frac{1 - \exp(-\alpha z)}{\alpha}$$

For $z \ll 1/\alpha$ $L_{\text{eff}}(z) \simeq z$, while for $z \gg 1/\alpha$ $L_{\text{eff}}(z) \simeq L_A$. It turns out that L_{eff} is a measure of the distance over which the nonlinear effect is significant.
Note from (3.6) that the solution is memoryless so that what happens at time t depends only from the input at the same time. Assuming zero loss, for the energy conservation principle the energy carried by time t, being the system memoryless, must remain unaltered, i.e. $|A(z,t)|^2 = |A(0,t)|^2$, so that SPM is a pure phase rotation in the time domain.

3.1.5 Alternative expressions of the NLSE

In the frequency domain (3.1) the Fourier transform of the electric field, i.e. $\widetilde{A}(z,\omega) = \mathcal{F}\{A(z,t)\}$, satisfies the following NLSE:

$$\begin{aligned}\frac{\partial \widetilde{A}}{\partial z} &= -\frac{\alpha}{2}\widetilde{A} - j\left(\beta_1\omega + \frac{\beta_2}{2}\omega^2 + \frac{\beta_3}{6}\omega^3\right)\widetilde{A} \\ &\quad -j\gamma \iint \widetilde{A}\left(z,\omega+\omega_1\right)\widetilde{A}\left(z,\omega+\omega_2\right)\widetilde{A}^*\left(z,\omega+\omega_1+\omega_2\right)\frac{d\omega_1}{2\pi}\frac{d\omega_2}{2\pi}\end{aligned} \tag{3.7}$$

where the integrals cover the entire space of real numbers. With the definition $A(z,t) = U(z,\tau)\exp\left(-\frac{\alpha}{2}z\right)$ with $\tau = t - z/\beta_1$ the NLSE becomes:

$$\frac{\partial U}{\partial z} = +j\frac{\beta_2}{2}\frac{\partial^2 U}{\partial \tau^2} + \frac{\beta_3}{6}\frac{\partial^3 U}{\partial \tau^3} - j\gamma\left|U\right|^2 U e^{-\alpha z}$$

The time τ is usually called the retarded time frame. Note that the linear impairment α still remains in the equation since the NLSE is a nonlinear differential equation.
Alternatively, by writing $A(z,\tau_n) = \sqrt{P}V(z,\tau_n)\exp\left(-\frac{z}{2L_A}\right)$ with $\tau_n = (t - z/\beta_1)/T_0$, being T_0 and P a reference time and power, respectively, and exploiting the characteristics lengths we have the following:

$$\frac{\partial V}{\partial z} = +j\frac{1}{2L_D}\frac{\partial^2 V}{\partial \tau_n^2} + \frac{1}{6L_D'}\frac{\partial^3 V}{\partial \tau_n^3} - \frac{j}{L_{\mathrm{NL}}}|V|^2 V e^{-z/L_A} \tag{3.8}$$

where τ is the time in a retarded frame normalized to a reference time T_0.

3.2 Coupled-NLSE (CNLSE)

The CNLSE is the vectorial version of the NLSE. It is used whenever polarization effects are to be considered, since the optical (transverse) propagating field envelope is represented by $\mathbf{A}(z,t)$, where the bold face stands for a complex vector with 2 elements (*Jones vector*). The CNLSE, in a general form, has the following expression [34, 35, 36]:

$$\frac{\partial \mathbf{A}(z,\tau)}{\partial z} = -\frac{\alpha}{2}\mathbf{A} - i\frac{\Delta\beta_0}{2}(\hat{l}(z)\cdot\vec{\sigma})\mathbf{A} - \frac{\Delta\beta_1}{2}(\hat{l}(z)\cdot\vec{\sigma})\frac{\partial \mathbf{A}}{\partial \tau} + j\frac{\beta_2}{2}\frac{\partial^2 \mathbf{A}}{\partial \tau^2} - j\gamma\left[|\mathbf{A}|^2\mathbf{A} - \frac{1}{3}\left(\mathbf{A}^\dagger\sigma_3\mathbf{A}\right)\sigma_3\mathbf{A}\right] \tag{3.9}$$

where $\tau = t - \frac{z}{v_g}$ is the retarded time frame, discussed in Sec. 3.1.2, while † means transpose-conjugate. Note that for the sake of simplicity we are neglecting the fiber slope. Comparing (3.9) with the *scalar* NLSE, described in Sec. 3.1, two new terms (those with $\Delta\beta_{0,1}$) appear in the linear part, while the nonlinear part of the equation is modified by an extra term (that with σ_3).
The linear terms are due to the fiber *birefringence*: when a polarized optical field propagates through a *birefringent* fiber, the propagation constant depends on the field polarization. This behavior can be due to imperfections in the fiber core as well as geometrical asymmetries, stresses, bends, etc. Birefringence is stochastic and fluctuates both in time (slowly, compared to the symbol period; hence it is assumed constant in τ in the CNLSE) and along fiber length, depending on the characteristics of the fiber and on local condition, such as temperature. At each position z, the fiber is characterized by an *eigenmode* $\hat{l}(z)$, corresponding to the field polarization with slowest propagation constant $\beta_s(\omega)$; at the same, time, the orthogonal eigenmode $\hat{l}_o(z)$, is the field polarization with fastest propagation constant $\beta_f(\omega)$. The difference $\Delta\beta(\omega) = \beta_s(\omega) - \beta_f(\omega)$ between these propagation constants is the strength of the birefringence, while $\hat{l}(z)$ represents the birefringence orientation. $\hat{l}(z)$ is a 3D unit-magnitude real vector (a *Stokes vector*) [37, 38]. As a simple example, $\hat{l}(z)$ can represent the "horizontal" linear polarization, where the field oscillates in the x direction (hence, the second element of $\mathbf{A}$ is null); in this case, $\hat{l}_o(z)$ is the the "vertical" linear polarization ($\mathbf{A}$ oscillates along the y direction and its first element is null). It is generally believed that silica fibers are characterized by linear birefringence [39]: this implies that the third component of $\hat{l}(z)$ is null.
In (3.9), birefringence is modeled by a Stokes vector $\vec{W}(z,\omega) = (\Delta\beta_0 + \Delta\beta_1\omega)\hat{l}(z)$ with a linear frequency dependence, although other models are possible; the variations in z of its orientation cause "random mode coupling", i.e., the exchange of energy between the field components parallel or perpendicular to $\hat{l}(z)$, eventually leading to Polarization Mode Dispersion (PMD), a distortion of the pulse shape causing signal degradation and intersymbol interference. If the frequency dependent term $\Delta\beta_1$ is set to zero, there is no pulse distortion, and the overall result of birefringence is just a rotation of the signal State Of Polarization (SOP) on the *Poincaré sphere*. If $\hat{l}$ is constant along z, the fiber is called Polarization Maintaining Fiber (PMF): its input-output behavior amounts to splitting each input pulse into two "shadow pulses" arriving at the fiber output with a mutual delay equal to $\Delta\beta_1 z$: this effect, known as *first-order PMD*, causes intersymbol interference. In the more general case, mode coupling produces PMD at all orders, hence a pulse is not only split in two (first-order PMD), but each of the shadow pulses suffers a different amount of linear distortion, including GVD, that differently affects the polarized components of the signal (*Polarization-dependent Chromatic Dispersion*, PCD).
The symbol $\vec{\sigma}$, appearing in (3.9), is the so-called *spin-vector*, whose elements are the three Pauli matrices (hence, $\vec{\sigma}$ is actually a tensor!); the scalar product $(\hat{l}(z)\cdot\vec{\sigma})$ yields a *unitary Jones matrix*, i.e. a 2×2 complex matrix with unit determinant, that acts on the elements of the field vector $\mathbf{A}$ and produces mode-coupling.

Of the three Pauli matrices, only the third, σ_3, appears in the nonlinear term in (3.9): despite many different (but equivalent) expressions are possible for the nonlinear term in (3.9), the concept is that the circular component of the signal polarization (associated to the third Stokes component, whose mathematical expression is $\left(\mathbf{A}^\dagger\sigma_3\mathbf{A}\right)$) plays a special role in the CNLSE. This peculiarity is not always remarked in the literature, since other alternative and simplified forms of the CNLSE are implemented for its numerical solution.

3.2.1 Poincaré sphere notation

The Stokes representation of the complex electric field $\mathbf{A} = [A_x,\, A_y]^T$ is by definition [38, 37]:

$$\vec{A} = \mathbf{A}^\dagger\vec{\sigma}\mathbf{A}$$

and hence it is a 4-dimensional vector equal to:

$$\vec{A} = \begin{bmatrix} |A_x|^2 + |A_y|^2 \\ |A_x|^2 - |A_y|^2 \\ 2\Re\{A_x^* A_y\} \\ 2\Im\{A_x^* A_y\} \end{bmatrix} = \left(|A_x| + |A_y|^2\right)\begin{bmatrix} 1 \\ \widehat{a} \end{bmatrix}$$

where $\widehat{a} = [a_1,\, a_2,\, a_3]^T$ is the Poincaré unit vector or Stokes SOP vector. The Poincaré sphere represents this vector as a point on the unit sphere, whose main axes are generally called $(s_1,\, s_2,\, s_3)$. Being $\widehat{a}$ a unit vector it can be written as:

$$\widehat{a} = \begin{bmatrix} \cos 2\theta \cos 2\varepsilon \\ \sin 2\theta \cos 2\varepsilon \\ \sin 2\varepsilon \end{bmatrix}$$

where θ and ε are the azimuth and ellipticity of the field $\mathbf{A}$. Hence, linear polarized light lies on the equator of the sphere, while circular polarized light lies on one of the poles of the sphere. Any other position over the sphere identifies an elliptical polarized field. Moreover, orthogonal fields have Stokes vectors in opposite directions on the Poincaré sphere.

3.2.2 Numerical solution of the CNLSE

The numerical integration of (3.9) requires choosing a sufficiently small step (in z): in fact, the birefringence term with $\Delta\beta_0$ is purely imaginary and causes a differential phase rotation in the signal components (hence a change of its state of polarization). Such a phase rotation is frequency-independent and does not cause signal distortion, but indeed affects the nonlinear term in the CNLSE. An important parameter of transmission fibers is the *beat length* $L_B = \frac{2\pi}{\Delta\beta_0}$: one mist then choose an integration step Δz such that the phase rotation $\Delta\beta_0\Delta z$ is small compared to 2π. Since L_B is typically of the orders of meters (or tens of meters), for standard fibers, the integration of (3.9) is extremely time-consuming.
An alternative approach, requiring much smaller computation times, is that of averaging the impact of signal polarization over the nonlinear term in (3.9): if L_B is small enough, the rapid variations in the state of polarization of $\mathbf{A}$ are such that the term $\frac{1}{3}\left(\mathbf{A}^\dagger\sigma_3\mathbf{A}\right)\sigma_3\mathbf{A}$ undergoes a *complete mixing* and reduces to $\frac{1}{9}\mathbf{A}|\mathbf{A}|^2$, on average. The CNLSE is then simplified to

$$\frac{\partial\mathbf{A}(z,\tau)}{\partial z} = -\frac{\alpha}{2}\mathbf{A} - i\frac{\Delta\beta_0}{2}(\hat{l}(z)\cdot\vec{\sigma})\mathbf{A} - \frac{\Delta\beta_1}{2}(\hat{l}(z)\cdot\vec{\sigma})\frac{\partial\mathbf{A}}{\partial\tau} + j\frac{\beta_2}{2}\frac{\partial^2\mathbf{A}}{\partial\tau^2} - j\gamma\frac{8}{9}|\mathbf{A}|^2\mathbf{A} \qquad (3.10)$$

which is known as the Manakov-PMD equation [35, 40]. Here, the nonlinear term introduces *common-mode* phase rotations that do not alter the signal SOP; moreover, nonlinear distortions only depend on the signal intensity and not on its polarization, although there is an interplay between the nonlinear distortions and the polarization distortions due to the linear term. Note that although the birefringence term (that with $\Delta\beta_0$) is still there, there is no need for integration steps smaller than L_B: the effect of birefringence can be analytically accounted for in the linear step of the SSFM algorithm (see Sec.3.4). The Manakov-PMD equation is generally regarded as a simple and reliable way to model optical fibers affected both by Kerr effects and PMD.

3.3 NLSE in the WDM case

Assume that the input field to an optical fiber is a wavelength division multiplexing (WDM) comb of M channels. Such global, or unique, field has lowpass envelope $A(z,t)$ at coordinate z and time t. Calling ω_0 the central bandpass frequency of this electric field, it is possible to express the unique field as:

$$A(z,t)=\sum_{k=1}^{M}A_k(z,t)e^{j\Delta\omega_k t} \tag{3.11}$$

where $A_k(z,t)$ is the lowpass envelope of channel k while $\Delta\omega_k=\omega_k-\omega_0$ is the difference between the central frequency of channel k and the central frequency of the WDM comb. We want to derive a propagation equation for $A_k(z,t)$ for $k=1,\dots,M$. To this aim we start from the NLSE (3.7) written in the frequency domain:

$$\begin{aligned}\frac{\partial\widetilde{A}(z,\omega)}{\partial z} &= -\frac{\alpha}{2}\widetilde{A}-j\beta(\omega)\widetilde{A}\\ &\quad -j\gamma\iint\widetilde{A}\left(z,\omega+\omega_1\right)\widetilde{A}\left(z,\omega+\omega_2\right)\widetilde{A}^*\left(z,\omega+\omega_1+\omega_2\right)\frac{d\omega_1}{2\pi}\frac{d\omega_2}{2\pi}\end{aligned} \tag{3.12}$$

where $\widetilde{A}(z,\omega)=\sum_k\widetilde{A}_k\left(z,\omega-\Delta\omega_k\right)$ is the Fourier transform of $A(z,t)$, $\beta(\omega)=\beta_1\omega+\frac{\beta_2}{2}\omega^2+\frac{\beta_3}{6}\omega^3+\dots$ and all parameters are evaluated at the bandpass frequency ω_0. Let us first investigate the linear part of (3.12). From the superposition principle we have:

$$\frac{\partial\widetilde{A}_k\left(z,\omega\right)}{\partial z}=-\frac{\alpha}{2}\widetilde{A}_k(z,\omega)-j\beta\left(\omega+\Delta\omega_k\right)\widetilde{A}_k\left(z,\omega\right) \tag{3.13}$$

where $\beta\left(\omega+\Delta\omega_k\right)=\beta_{0k}+\beta_{1k}\omega+\frac{\beta_{2k}}{2}\omega^2+\frac{\beta_{3k}}{6}\omega^3+\dots$ being $\beta_{nk}=\left.\frac{\mathrm{d}^n\beta(\omega)}{\mathrm{d}\omega^n}\right|_{\omega=\Delta\omega_k}$. For the assumptions it is $\beta_{0k}=0$ if channel k has $\Delta\omega_k=0$. For instance, if $\beta(\omega)$ is cubic in ω, we have that $\beta_{2k}=\beta_2+\beta_3\Delta\omega_k$. The nonlinear part of (3.12) is more easy to manage in the time domain yielding:

$$\sum_k\frac{\partial A_k}{\partial z}e^{j\Delta\omega_k t}=-j\gamma\sum_{n,l,m}A_nA_lA_m^*e^{j(\Delta\omega_k+\Delta\omega_l-\Delta\omega_m)t} \tag{3.14}$$

We are tempted to split (3.14) into the following system of differential equations:

$$\frac{\partial A_k}{\partial z}=-j\gamma_k\sum_{\substack{n,l,m\\ \omega_n+\omega_l-\omega_m=\omega_k}}^{M}A_nA_lA_m^*,\qquad k=1,\dots,M \tag{3.15}$$

where n, l, m can range from 1 to M but must satisfy $\omega_n+\omega_l-\omega_m=\omega_k$. The substitution of $\gamma_k=\gamma\left(\omega_k\right)$ for γ in (3.15) follows the same steps as done for β_{nk} in the linear equation (3.13), but since γ is slowly varying with ω (see 3.5) we set γ_k constant over the bandwidth of $A_k(z,t)$.
The solution of (3.15) does not coincide with the solution of (3.14) since in (3.15) we discarded all terms of the sum falling outside the bandwidth of $A(z,t)$, i.e. all n, l, m such that $\omega_n+\omega_l-\omega_m$ cannot be associated

to a frequency ω_k, $k = 1, \dots, M$ of the comb. However, such terms are usually of small energy for weakly nonlinear systems, hence the solution of (3.15) is of great interest. It turns out that we are investigating a system of differential equations (3.15) behaving as a closed system.
We are now in position to merge the linear (3.13) and the nonlinear (3.15) differential equations together. After inserting β_{0k} into a constant phase shift as $A_k(z,t) = E_k(z,\omega)\exp(-j\beta_{0k}z)$, the NLSE, which we call the NLSE for separate fields, in the time domain writes as:

$$\begin{aligned}\frac{\partial E_k}{\partial z} &= -\frac{\alpha}{2}E_k - \left(\beta_{1k}\frac{\partial}{\partial t} - j\frac{\beta_{2k}}{2}\frac{\partial^2}{\partial t^2} - \frac{\beta_{3k}}{6}\frac{\partial^3}{\partial t^3}\right)E_k \\ &\quad -j\gamma_k \sum_{\substack{n,l,m \\ \omega_n+\omega_l-\omega_m=\omega_k}} E_n(z,t)\,E_l(z,t)\,E_m^*(z,t)\,e^{-j\Delta\beta_{nlm}z}\end{aligned} \tag{3.16}$$

where $\Delta\beta_{nlm} = \beta_{0n} + \beta_{0l} - \beta_{0m} - \beta_{0k}$ is called the phase matching coefficient. By varying the indexes n, l, m we can identify the following terms:

- self phase modulation (SPM): $\omega_n = \omega_l = \omega_m = \omega_k \Rightarrow |E_k|^2 E_k$
- cross phase modulation (XPM): $(\omega_n = \omega_m) \neq (\omega_l = \omega_k)$ or $(\omega_n = \omega_k) \neq (\omega_l = \omega_m) \Rightarrow \sum_{m\neq k} |E_m|^2 E_k$
- four wave mixing (FWM): set $\mathcal{H}$ of all terms that are not SPM or XPM $\Rightarrow$

$$\sum_{n,l,m\in\mathcal{H}} E_n(z,t)\,E_l(z,t)\,E_m^*(z,t)\,e^{j\Delta\beta_{nlm}z}$$

Exploiting these terms in (3.16) yields:

$$\begin{aligned}\frac{\partial E_k}{\partial z} &= -\frac{\alpha}{2}E_k - \left(\beta_{1k}\frac{\partial}{\partial t} - j\frac{\beta_{2k}}{2}\frac{\partial^2}{\partial t^2} - \frac{\beta_{3k}}{6}\frac{\partial^3}{\partial t^3}\right)E_k \\ &\quad -j\gamma_k\left(|E_k|^2 E_k + 2\sum_{m\neq k}|E_m|^2 E_k + \sum_{n,l,m\in\mathcal{H}} E_n(z,t)\,E_l(z,t)\,E_m^*(z,t)\,e^{j\Delta\beta_{nlm}z}\right)\end{aligned} \tag{3.17}$$

Note that $E_k(z,t)$ may differ from zero even if $E_k(0,t) = 0$ because of the FWM process due to the other terms. If we further assume that such terms remains undepleted in z, the resulting FWM over E_k is called first order FWM. See [1] for an introduction to FWM and [41, 42] for advanced studies of FWM.
Also note that in absence of FWM the solution of (3.15) coincides with the solution of (3.14) since XPM and SPM alone cannot generate E_k if $E_k(0,t) = 0$.

3.3.1 Unique and separate fields

Solving the NLSE for $E(z,t)$ corresponds to solve the NLSE for a *unique field*. Solving the NLSE for $E_k(z,t)$, $k = 1, 2, \dots$ corresponds to solve the NLSE for *separate fields*. Such options must be indicated in create_field. The unique field solution is the most complete one, since it correctly includes SPM, XPM and FWM. The solution for separate fields is possible in a simple and numerically efficient form only in absence of FWM, for which a closed form expression of the solution in presence of the nonlinear operator only exists. In such a case the NLSE (3.16) for channel k is:

$$\frac{\partial E_k}{\partial z} = -\frac{\alpha}{2}E_k - \beta_{1k}\frac{\partial E_k}{\partial t} - j\gamma|E_k|^2 E_k - j2\gamma\sum_{m\neq k}|E_m|^2 E_k$$

where for the sake of simplicity we assumed $\alpha_k = \alpha$ and $\gamma_k = \gamma$, $\forall k$. The solution in the retarded frame time $\tau = (t - z/\beta_{1k})/T_0$ is:

$$E_k(z,\tau) = E_k(0,\tau)e^{-j\gamma\left(|E_k(0,\tau)|^2 + 2\sum_{m\neq k}|E_m(0,\tau+zd_{km})|^2\right)L_{\text{eff}}(z)}\,. \tag{3.18}$$

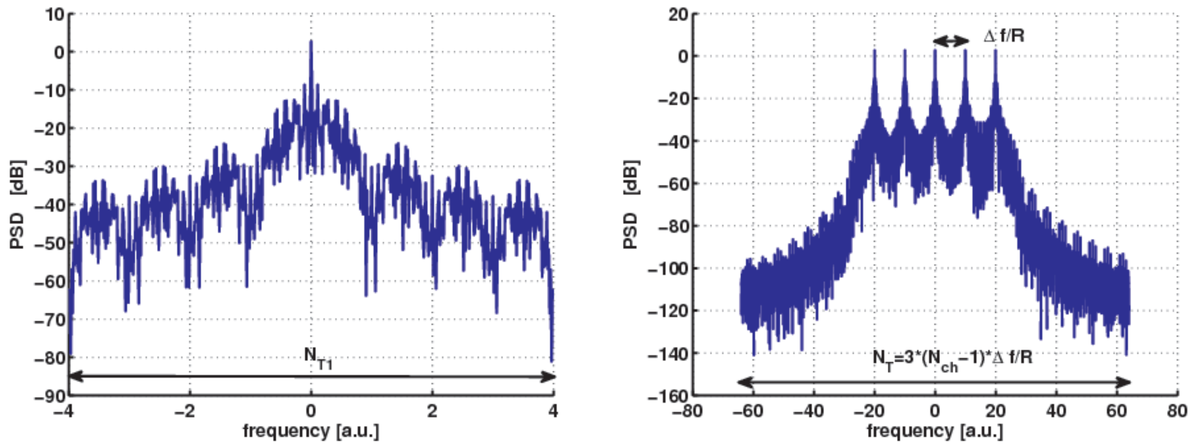

Figure 3.1: Left: Spectrum of a generic channel in a separate field scenario of $N_{ch} = 5$ channels. Right: Spectrum in a unique field scenario.

being d_{km} the walkoff parameter (3.2). A parallel SSFM algorithm applied to each channel is therefore possible. Note that the nonlinear phase rotation due to XPM is in principle two times as larger than the one of SPM. However, XPM is reduced by the filtering effect of the walk-off effect [1].
The main motivations for using a separate field approach are the following:

1. With separate fields it is possible to allocate a lower Nyquist frequency, i.e. a lower number of points per symbol. Infact the electric field (e.g. `GSTATE.FIELDX`) is now a matrix where each column represent a channel. hence the number of points per symbol must be sufficient to describe just a single channel. The figure below, showing the power spectrum vs. the frequency normalized to the symbol rate, sketches the idea. The unique field is a WDM comb of $N_{ch} = 5$ OOK channels, spaced $\Delta f = 100$ [GHz] @ $R = 10$ Gbs. Suppose that $N_{\text{symb}} = 64$ symbols are sufficient to describe such a system. It remains the problem of finding a good value for N_T, i.e. the number of points per symbol. For the right figure we note that from the first to the fifth channel the spacing is $B = (N_{ch} - 1)\,\Delta f$ [GHz] which in normalized units is B/R. In a unique field scenario, since FWM enlarges the spectrum during the propagation, in order to capture at least 1st order FWM the bandwidth allocation must be at least three times larger, yielding $N_T = 3\lceil B/R\rceil = 3\,(N_{ch}-1)\lceil \Delta f/R\rceil$ points per symbol. We obtain $N_T = 120$ and hence use $N_T = 128$ for working the FFT efficiently. In a separate field scenario the allocation is different. Now FWM is absent, but SPM still enlarges the spectrum. A good choice is to describe the spectrum with some lobes to capture this enlargement. In the left figure we used $N_{T1} = 8$ points per symbol which allows to capture the first four lobes. N_{T1} is used for all channels, hence each step a basic SSFM runs N_{ch} FFTs and IFFTs of size $N = N_{T1}N_{\text{symb}}$ each, while the unique field runs one FFT/IFFT of $N = N_T N_{\text{symb}}$ points. Assuming the cost of the FFT equal to $N \log_2 N$, the FFT algorithm in the separate field scenario costs 23040 while using the unique field costs 106496.

2. In the separate field case the step of the SSFM algorithm can be significantly larger. For instance, by adopting a step with fixed maximum nonlinear phase rotation as in Section 3.4.1.2, the step size is inversely proportional to the maximum power. In a separate field scenario the maximum power at coordinate z is $\max\limits_k\left(\max\limits_t\left(|E_k(z,t)|^2\right)\right)$, while in a unique field scenario it is:

$$\max_t\left(|E(z,t)|^2\right) = \max_t\left(\left|\sum_k E_k(z,t)e^{j\Delta\omega_k t}\right|^2\right) < \max_t\left(\sum_k |E_k(z,t)|^2\right)$$

The last bound is greater than the maximum power of the separate field case, hence with a unique field the propagating power may be greater than with separate fields, thus giving smaller steps.

The separate field approach is discouraged for the following motivations:

1. The approach, as implemented in Optilux, neglects FWM, hence it works only for optical systems with non negligible dispersion management.

2. The separate fields method basically neglects the channels spectrum overlap, hence it is inappropriate for dense WDM systems.

3.4 Numerical solution of the NLSE: SSFM

The split-step Fourier Method (SSFM) is an efficient algorithm for the numerical solution of the NLSE. It is a special application of the splitting method for solving a PDE. Generally speaking, the method is useful to face out the problem [43]:

$$\frac{\partial A(z,t)}{\partial z} = \mathcal{D}A(z,t) \tag{3.19}$$

where $\mathcal{D}$ is a differential operator that can be written in the form $\mathcal{D} = \mathcal{L} + \mathcal{N}$, being $\mathcal{L}$ and $\mathcal{N}$ differential operators as well, such that:

$$\begin{aligned} \frac{\partial A}{\partial z} &= \mathcal{L}A \\ \frac{\partial A}{\partial z} &= \mathcal{N}A \end{aligned}$$

are two differential equations easy to solve. This is the case of the NLSE where closed form solutions exist with only dispersion or with only nonlinearity. If $\mathcal{D}$ does not depends on z and $\mathcal{D}A$ is continuous with continuous derivatives, from the Peano-Picard-Liouville theorem the following succession in n uniformly converges to the exact solution of (3.19):

$$\frac{\partial A_{n+1}(z,t)}{\partial z} = \mathcal{D}A_n(z,t)$$

yielding a solution that can formally be written as:

$$A(z,t) = \exp\left(\mathcal{D}z\right) A(0,t)$$

where the exponential makes sense only after writing it into a series form:

$$\exp\left(\mathcal{D}z\right) = \sum_{n=0}^{\infty} z^n \frac{\mathcal{D}^n}{n!} \tag{3.20}$$

and $\mathcal{D}^n$ means the application of $\mathcal{D}$ n times. If $\mathcal{D}$ is non-constant the solution is more complicated. However, as we will see next, such operator is usually evaluated over small steps where it can be assumed almost constant. A possible generalization of the problem is with $\mathcal{D} = g(z)\mathcal{D}_1$ with $g(z)$ a continuous function with continuous derivative and $\mathcal{D}_1$ a constant differential operator, yielding $\exp\left(\mathcal{D}_1 \int_0^z g(x)dx\right)$ instead of $\exp\left(\mathcal{D}z\right)$.
Usually $\exp\left(\mathcal{D}z\right)$ is unknown or very tedious to evaluate, while it is easy to evaluate $\exp\left(\mathcal{L}z\right)$ and $\exp\left(\mathcal{N}z\right)$. Using (3.20) it is simple to show that:

$$\begin{aligned} \exp\left(\mathcal{D}z\right) &= \exp\left(\left(\mathcal{L}+\mathcal{N}\right)z\right) = \exp\left(\mathcal{L}z\right)\cdot\exp\left(\mathcal{N}z\right) + O\left(z^2\right) \\ &= \exp\left(\mathcal{N}z\right)\cdot\exp\left(\mathcal{L}z\right) + O\left(z^2\right) \end{aligned} \tag{3.21}$$

where $O\left(z^2\right)$ indicates a term that is bounded by a constant times z^2. Such term is zero only when the operators $\mathcal{L}$ and $\mathcal{N}$ commute, i.e. $\mathcal{L}\mathcal{N} = \mathcal{N}\mathcal{L}$. Unfortunately this is not the case of the NLSE, hence applying separately the two operators leaves an error that decreases quadratically by decreasing z. The $O\left(z^2\right)$ is usually called local truncation error. The SSFM consists therefore in subdividing the fiber in small steps and by applying in each step the two operators separately (splitting). Summarizing, if $z_{k+1} = h_k + z_k$ with $z_0 = 0$ and $z_M = L$ being L the fiber length, the algorithm is the following:

$$A(z_{k+1}, t) = \exp(\mathcal{L}h_k) \cdot \exp(\mathcal{N}h_k) A(z_k, t), \quad k = 0, 1, \ldots, M-1$$

If the steps are sufficiently small, the local truncation error is "small" as well and hopefully $A(L, t)$ is very close to the exact solution.
The local truncation error can be reduced to $O(z^3)$ by using the two alternative splitting:

$$\begin{aligned}\exp(\mathcal{D}z) &= \exp\left(\frac{\mathcal{L}}{2}z\right) \cdot \exp(\mathcal{N}z) \cdot \exp\left(\frac{\mathcal{L}}{2}z\right) + O(z^3) \\ \exp(\mathcal{D}z) &= \exp\left(\frac{\mathcal{N}}{2}z\right) \cdot \exp(\mathcal{L}z) \cdot \exp\left(\frac{\mathcal{N}}{2}z\right) + O(z^3) \qquad (3.22)\end{aligned}$$

The verify can be done by substituting the exponentials with their series expansion up to z^3 in (3.22).
Such idea, called the symmetric split-step, resemble the trapezoidal rule for numerically evaluate integrals [44, 22]. Note that in principle the symmetric split-step requires three exponential evaluations instead of two of the standard split-step. However, when many steps are applied successively and the step sizes are known a priori, adjacent operators can be combined into a single operator, yielding:

$$\begin{aligned}A(L,t) = &\left\{\exp\left(\frac{\mathcal{N}}{2}h_{M-1}\right) \cdot \exp(\mathcal{L}h_{M-1}) \cdot \exp(\mathcal{N}(h_{M-1} + h_{M-2})) \cdot \ldots\right. \\ &\left.\cdot \exp(\mathcal{N}(h_0 + h_1)) \cdot \exp(\mathcal{L}h_0) \exp\left(\frac{\mathcal{N}}{2}h_0\right)\right\} A(0,t)\end{aligned}$$

For the NLSE, the SSFM can be applied to the form (3.8), while the electric field $A(z_k, t)$ of (3.1) can be recovered by applying the step-attenuation at the end of each step. The operator $\mathcal{L}$ is therefore a constant linear operator:

$$\mathcal{L}z = z\left(-\beta_1\frac{\partial}{\partial t} + j\beta_2\frac{\partial^2}{\partial t^2} + \beta_3\frac{\partial^3}{\partial t^3}\right)$$

while $\exp(\mathcal{N}h_k)$ applied between coordinate $z = z_k$ and $z = z_k + h_k$ is generalized in $\exp\left(\int_0^{h_k} e^{-\alpha x}dx\mathcal{N}(z_k)\right) = \exp(-L_{\text{eff}}(h_k)\mathcal{N}(z_k))$ with $\mathcal{N}(z_k) = |A(z_k,t)|^2$ a z-independent nonlinear operator and

$$L_{\text{eff}}(h_k) = (1 - \exp(-\alpha h_k))/\alpha$$

the effective length due to the fiber attenuation α.
The linear operator is efficiently evaluated in the Fourier domain while the nonlinear operator in the time domain. Such approach calls for the FFT and IFFT algorithm for switching between the two domains efficiently.

3.4.1 Step choice

The choice of the SSFM step size is an hard task for which is difficult to give an universal answer, suitable for any optical system. The most accurate approach for choosing the SSFM step size is to run many simulations for decreasing step sizes until some convergence is observed. The convergence criterion is based on a distance measurement applied to some parameter, like the electric field, the sensitivity penalty, etc.
In Optilux the step can be chosen in different ways. These calls are active only in the nonlinear regime where the SSFM is an approximate algorithm. In cases where the exact solution is known, Optilux ignores the step method and evaluates the solution in a single step (e.g. with only GVD).
Assuming that fiber is called as `fiber(x,flag)` we have the following options for the step.

3.4.1.1 Constant step size

Simply call fiber with `x.dzmax=C` and `x.dphimax=Inf`. The step is fixed and equal to `C` [m]. This method uses a standard-SSFM.

3.4.1.2 Constant nonlinear phase rotation x step

The step is adaptively chosen so as to have a maximum nonlinear phase rotation (3.18) over the step equal to `x.dphimax` [rad]. If also `x.dzmax` exists and satisfies `x.dzmax<x.length`, the step is chosen in the same way, but cannot be larger than `x.dzmax`. This method allows short steps in regions of high power (usually at the beginning of the fiber) and large steps in regions of low power (usually at the end of the fiber). With a unique field the nonlinear phase rotation is the one of SPM; with separate fields it also accounts for XPM. See Section 3.3.1 for more details. The method of constant nonlinear phase rotation per step uses a standard SSFM. See [45, 46] for more details about the method.

3.4.1.3 Adaptive step based on the local error

1. This method implements an adaptive search of the step based on local information as described in [47]. The method is similar to the algorithm proposed in [45], except for the step updating rule. With the additional option `x.ltol`, the step is adaptively chosen so as to have a local truncation error equal to `x.ltol`. The local truncation error is defined as:

   $$e_l = \max\left(\|A_1 - A_2\|\right)$$

   being A_1 the electric field evaluated after the step and A_2 the electric field evaluated after propagating in the step twice, each with an halved step compared with A_1. This method is based on the symmetric-SSFM. Optilux uses the following norm for a matrix y:

   $$\|y\| = \texttt{max(max(abs(y)));}$$

   With such method x.dphimax is used to get a trial for the first step only. Please, note that the adaptive step is active for scalar propagations only (PMD flag is `'-'`).
   The basic idea behind the adaptive step method is the following. Denote with $A_e(z,t)$ the exact solution of the NLSE given the initial condition $A_e(z_n,t)$, and with $A_1(z,t)$ the approximate estimation of $A_e(z,t)$ using the symmetric-SSFM with a single step from z_n to $z = z_n + h$. From the theory we have that the following relation holds:

   $$A_1(z_n + h, t) = A_e(z_n + h, t) + Kh^3 + O\left(h^4\right) \tag{3.23}$$

   where Kh^3 is the local truncation error, while $O\left(h^4\right)$ is a term that we assume negligible compared to Kh^3. $K = K(t)$ is an unknown multiplying factor at this stage. Let us now advance from $z = z_n$ to $z = z_n + h$ with two SSFM steps, each of length $h/2$. Along the first step of length $h/2$ the local error is $K\left(h/2\right)^3$. Under the assumption that the $O\left(h^4\right)$ term is negligible, we can say that even along the second step we have an error Kh^3, with the same K since (3.23) K comes from a Taylor expansion evaluated in $z = z_n$. Summing the errors, we have that the two-step SSFM gives the following answer:

   $$A_2(z_n + h, t) = A_e(z_n + h, t) + 2K\left(\frac{h}{2}\right)^3 + O\left(h^4\right) \tag{3.24}$$

   which is more accurate than A_1. We can evaluate the local error between $A_1(z_n+h,t)$ and $A_2(z_n+h,t)$ which is our indicator of the step reliability:

   $$e_L = \max_t\left(|A_1 - A_2|\right) \simeq \frac{3}{4}K_m h^3$$

   being $K_m = \max_t\left(K(t)\right)$. From the local error we introduce the local error rate x unit step:

   $$r = \frac{e_L}{h} \simeq \frac{3}{4}K_m h^2 \tag{3.25}$$

   If $r > \varepsilon$, being ε a target local error in which we trust, the step must be rejected and repeated with a shorter one. Otherwise the step is accurate enough (on the basis of our target) and accepted. There remain two fundamental questions: how much we reduce h in case of failure and what h can we use

for the next step in case of success?
After multiplying both members of (3.25) by ε/r we have the following:

$$\varepsilon \simeq \frac{3}{4}K_m\left(\sqrt{\frac{\varepsilon}{r}}h\right)^2 = \frac{3}{4}K_m\left(h^{'}\right)^2 \tag{3.26}$$

It turns out that the new step $h^{'}$ should give the desired target error ε. To give ourself more confidence in this guess we introduce a safety factor by using $h^{'} = 0.9h\sqrt{\varepsilon/r}$. If $r > \varepsilon$ we have $h^{'} < h$, otherwise the opposite. Such a behavior agrees with the intuition besides the adaptive step size, e.g. use short steps when the accuracy is small. In Optilux ε is `x.ltol`. Having identified a reasonable method for adapting the step, we can further improve the solution. When the step is accepted, we could use A_2 as candidate to propagate for the next step since is more accurate than A_1. But maybe we can use use the information inside A_1 to increase the accuracy of A_2. From (3.23) and (3.24), we have the following relation in $z = z_n + h^{'}$:

$$A_R = \frac{4}{3}A_2 - \frac{1}{3}A_1 = A_e + O\left(h^4\right)$$

which is accurate as $O\left(h^4\right)$, and hence is a best guess than A_2. Such a linear combination is called Richardson extrapolation [47, 45].
The algorithm is therefore the following:

(a) Evaluate A_1 using a single step starting from z_n;

(b) Evaluate A_2 using two steps starting from z_n;

(c) Compute the local error-rate r;

(d) Compute $h^{'}$;

(e) If $r > \varepsilon$ reject the step and repeat using $h^{'}$. If $r < \varepsilon$ accept the step and use $h^{'}$ for the next step using A_R as initial condition.

Note 1: The adaptive step needs to keep memory of three electric fields: A_1, A_2 and the electric field at the beginning of the step in z_n. Hence the RAM consumption is at least three times larger than the method in Section 3.4.1.2.

Note 2: The adaptive step uses `x.phimax` and `x.dzmax` to evaluate the trial for the first step only.

Note 3: Remember that the local truncation error differs from the global truncation error, i.e. the output error. The relation between such errors is generally unknown, but an upper bound can be found [47].

3.4.1.4 Adaptive step just in the first step

This option is active with the additional flag `x.dphiadapt=true`. The first step is evaluated basing on the local truncation error as in Section 3.4.1.3. Once the first step is known, `fiber` corrects the value of `x.dphimax` to have that step, and then proceeds as for the method of the constant nonlinear phase rotation x step. It turns out that the first step is evaluated with a symmetric-SSFM, while all the others with a standard SSFM.

Summarizing we can set the step by activating the following flags (O: optional, *: required):

flag	3.4.1.1	3.4.1.2	3.4.1.3	3.4.1.4
`x.dzmax`	*	O	O	O
`x.dphimax`	*	*	*	*
`x.ltol`			*	*
`x.dphiadapt`				*

Note that the flag `x.dphimax` is always active since we want the user conscious of its choice. In cases where the solution of the NLSE is known exactly, the step is automatically set equal to the fiber length.

Bibliography

[1] G. Agrawal, *Nonlinear Fiber Optics.* Academic Press, January 2001.

[2] F. Annexstein, "Generating De Bruijn sequences: an efficient implementation," *IEEE Trans. Comput.*, vol. 46, no. 2, pp. 198–200, Feb 1997.

[3] D. van den Borne, G. Khoe, H. de Waardt, and E. Gottwald, "Bit pattern dependence in optical DQPSK modulation," *Electronics Letters*, vol. 43, no. 22, pp. –, 25 2007.

[4] J. Proakis, *Digital Communications.* McGraw-Hill, 2000.

[5] A. Gnauck and P. Winzer, "Optical phase-shift-keyed transmission," *J. Lightw. Technol.*, vol. 23, no. 1, pp. 115–130, Jan. 2005.

[6] D. Penninckx, M. Chbat, L. Pierre, and J.-P. Thiery, "The phase-shaped binary transmission (PSBT): a new technique to transmit far beyond the chromatic dispersion limit," *IEEE Photon. Technol. Lett.*, vol. 9, no. 2, pp. 259–261, Feb. 1997.

[7] D. Penninckx, "Enhanced-phase-shaped binary transmission," *Electronics Letters*, vol. 36, no. 5, pp. 478–480, Mar 2000.

[8] H. Kim and C. Yu, "Optical duobinary transmission system featuring improved receiver sensitivity and reduced optical bandwidth," *IEEE Photon. Technol. Lett.*, vol. 14, no. 8, pp. 1205–1207, Aug 2002.

[9] D. Penninckx, H. Bissessur, P. Brindel, E. Gohin, F. Bakhti, R. Alcatel, and F. Marcoussis, "Optical differential phase shift keying (DPSK) direct detection considered as a duobinary signal," in *Proc ECOC 2001*, 2001, paper We.P.40.

[10] E. Forestieri and G. Prati, "Narrow filtered DPSK implements order-1 CAPS optical line coding," *IEEE Photon. Technol. Lett.*, vol. 16, no. 2, pp. 662–664, Feb. 2004.

[11] I. Lyubomirsky and C.-C. Chien, "DPSK demodulator based on optical discriminator filter," *IEEE Photon. Technol. Lett.*, vol. 17, no. 2, pp. 492–494, Feb. 2005.

[12] I. Lyubomirsky, C.-C. Chien, and Y.-H. Wang, "Optical DQPSK receiver with enhanced dispersion tolerance," *IEEE Photon. Technol. Lett.*, vol. 20, no. 7, pp. 511–513, April1, 2008.

[13] S. Walklin and J. Conradi, "Effect of mach-zehnder modulator DC extinction ratio on residual chirp-induced dispersion in 10-Gb/s binary and AM-PSK duobinary lightwave systems," *IEEE Photon. Technol. Lett.*, vol. 9, no. 10, pp. 1400–1402, Oct. 1997.

[14] H. Kim and A. Gnauck, "Chirp characteristics of dual-drive. mach-zehnder modulator with a finite dc extinction ratio," *IEEE Photon. Technol. Lett.*, vol. 14, no. 3, pp. 298–300, Mar 2002.

[15] T. Kawanichi, K. Higuma, T. Fujita, S. Mori, S. Oikawa, J. Ichikawa, T. Sakamoto, M. Tsuchiya, and M. Izutsu, "40Gbit/s Versatile LiNbO3 External Lightwave Modulator," in *Proc ECOC 2005*, 2005, paper Th.2.2.6.

[16] T. Kawanishi, T. Sakamoto, A. Chiba, M. Izutsu, K. Higuma, J. Ichikawa, T. Lee, and V. Filsinger, "High-speed dual-parallel mach-zehnder modulator using thin lithium niobate substrate," in *Proc. OFC 2008*, 2008, paper JThA34.

[17] A. B. Carlson, P. Crilly, and J. Rutledge, *Communication Systems*, 4th ed. McGraw-Hill, New York, 2002.

[18] B. Mikkelsen, C. Rasmussen, P. Mamyshev, and F. Liu, "Partial dpsk with excellent filter tolerance and osnr sensitivity," *Electronics Letters*, vol. 42, no. 23, pp. 1363–1364, 9 2006.

[19] V. Mikhailov, R. I. Killey, and P. Bayvel, "Experimental investigation of partial demodulation of 85.3 gb/s dqpsk signals," in *Proc. ECOC 2008*, paper We.1.E.5,2008.

[20] E. Forestieri, "Evaluating the error probability in lightwave systems with chromatic dispersion, arbitrary pulse shape and pre- and postdetection filtering," *J. Lightw. Technol.*, vol. 18, no. 11, pp. 1493–1503, Nov 2000.

[21] P. Serena, A. Orlandini, and A. Bononi, "Parametric-gain approach to the analysis of single-channel DPSK/DQPSK systems with nonlinear phase noise," *J. Lightw. Technol.*, vol. 24, no. 5, pp. 2026–2037, May 2006.

[22] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in Fortran 77: The Art of Scientific Computing.* NY. Cambridge Univ. Press, 1992.

[23] G. Fishman, *Monte Carlo: Concepts, Algorithms, and Applications.* Springer, 1996.

[24] A. Papoulis, *Probability, random variables, and stochastic processes*, 3rd ed. McGraw-Hill, 1991.

[25] M. Jeruchim, P. Balaban, and K. Shanmugan, *Simulation of communication systems.* Plenum Press New York, NY, USA, 1992.

[26] D. Godard, "Self-recovering equalization and carrier tracking in two-dimensional data communication systems," *IEEE Trans. Commun.*, vol. 28, no. 11, pp. 1867–1875, Nov 1980.

[27] A. Leven, N. Kaneda, U.-V. Koc, and Y.-K. Chen, "Frequency estimation in intradyne reception," *IEEE Photon. Technol. Lett.*, vol. 19, no. 6, pp. 366–368, March15, 2007.

[28] A. Viterbi, "Nonlinear estimation of PSK-modulated carrier phase with application to burst digital transmission," *IEEE Trans. Inf. Theory*, vol. 29, no. 4, pp. 543–551, Jul 1983.

[29] K. Kikuchi, "Electronic Post-compensation for Nonlinear Phase Fluctuations in a 1000-km 20-Gbit/s Optical Quadrature Phase-shift Keying Transmission System Using the Digital Coherent Receiver," *Optics Express*, vol. 16, no. 2, pp. 889–896, 2008.

[30] J.-F. Cardoso and B. Laheld, "Equivariant adaptive source separation," *IEEE Trans. Signal Process.*, vol. 44, no. 12, pp. 3017–3030, Dec 1996.

[31] A. Vannucci and A. Bononi, "Statistical characterization of the Jones matrix of long fibers affected by polarization mode dispersion (PMD)," *J. Lightw. Technol.*, vol. 20, no. 5, pp. 811–821, May 2002.

[32] B. E. A. Saleh and M. C. Teich, *Fundamentals of Photonics*, 2nd ed. Wiley-Interscience, 2007.

[33] C. W. Helstrom, *Elements of signal detection and estimation.* NJ, USA: Prentice-Hall, Inc., 1995.

[34] C. Menyuk and B. Marks, "Interaction of polarization mode dispersion and nonlinearity in optical fiber transmission systems," *J. Lightw. Technol.*, vol. 24, no. 7, pp. 2806–2826, July 2006.

[35] D. Marcuse, C. Manyuk, and P. Wai, "Application of the Manakov-PMD equation to studies of signal propagation in optical fibers with randomly varying birefringence," *J. Lightw. Technol.*, vol. 15, no. 9, pp. 1735–1746, Sep 1997.

[36] C. Menyuk and B. Marks, "Interaction of polarization mode dispersion and nonlinearity in optical fiber transmission systems," *J. Lightw. Technol.*, vol. 24, no. 7, pp. 2806–2826, July 2006.

[37] J. P. Gordon and H. Kogelnik, "PMD fundamentals:polarization-mode dispersion in optical fibers," *Proc. Nat. Acad. Sci.*, vol. 97, no. 9, pp. 4541–4550, Apr. 2000.

[38] A. Bononi and A. Vannucci, "Is there life beyond the principal states of polarization?" *Optical Fiber Technology*, vol. 8/4, pp. 257–294, Oct. 2002.

[39] I. Kaminow, "Polarization in optical fibers," *IEEE J. Quantum Electron.*, vol. 17, no. 1, pp. 15–22, Jan 1981.

[40] S. J. Evangelides, L. Mollenauer, J. Gordon, and N. Bergano, "Polarization multiplexing with solitons," *J. Lightw. Technol.*, vol. 10, no. 1, pp. 28–35, Jan 1992.

[41] M. Eiselt, "Limits on WDM systems due to four-wave mixing: a statistical approach," *J. Lightw. Technol.*, vol. 17, no. 11, pp. 2261–2267, Nov 1999.

[42] S. Kumar, "Analysis of degenerate four-wave-mixing noise in return-to-zero optical transmission systems including walk-off," *J. Lightw. Technol.*, vol. 23, no. 1, pp. 310–320, Jan. 2005.

[43] R. J. LeVeque and J. Oliger, "Numerical methods based on additive splittings for hyperbolic partial differential equations," *Mathematics of Computation*, vol. 40, no. 162, pp. 469–497, 4 1983.

[44] J. H. Mathews and K. K. Fink, *Numerical Methods Using Matlab*, 4th ed. Prentice Hall, Inc., 2004.

[45] O. Sinkin, R. Holzlohner, J. Zweck, and C. Menyuk, "Optimization of the split-step Fourier method in modeling optical-fiber communications systems," *J. Lightw. Technol.*, vol. 21, no. 1, pp. 61–68, Jan 2003.

[46] Q. Zhang and M. Hayee, "An SSF scheme to achieve comparable global simulation accuracy in WDM systems," *IEEE Photon. Technol. Lett.*, vol. 17, no. 9, pp. 1869–1871, Sept. 2005.

[47] J. Feldman, "Variable step size methods," http://www.math.ubc.ca/ feldman/math/vble.pdf.