Notes for **pydispersion** library

All the library can be found at

`https://github.com/louislafforgue/pydispersion`

Author : Louis Lafforgue

louislafforg@gmail.com

Chapter 1

Pulse implementation

All the class are stored in the file *pulse.py*

1.1 Analytical pulse

```
Analytics(self, shape="Gaussian", w0=2, E0=1, FWHM=10, poly_phase=[0] ,repre
    ="time",N_grid=2**15, N_signal=2**11, **kwargs)
```

Creation of a pulse object from an analytical expression either from temporal or frequencial representation, the different parameters and pulse representation are implemented thanks to a fourier transform

- *shape* : "Gaussian", "Hsech", "Lorentzian", default : "Gaussian"
- *repre*: "time" or "freq", parameters given for the time representation of the pulse id repre="time", for the spectral representation if repre="freq".
- *w0* :float, ω_0 of the pulse in rad/fem, default 2
- *E0* : float, Maximum amplitude in arbitrary unit, default 1
- *FWHM*: float, FWHM of the pulse depending of the representation.
- *poly_phase* : poly_phase= list(float) , list with the polynomial coffecients, ex: [1,2.3,5] = $1+2.3t+5t^2$
- *N_grid* : int, size for the fourier transform grif, more efficient as a power of two
- *N_grid* : int, size for the discretized signal, more efficient as a power of two

1.2 Data input pulse

```
Data_input(self,file_name,phase=[], axex="lambda", smoothing=True, **kwargs)
```

Creation of a pulse object from .dat or .csv file the file has to be given as following first column: frequency, second column : Intensity amplitude

- *file_name*: string (), file name with the extension, example: "pulse.dat"
- *phase* : np.array, phase array, default, null phase
- *axex* : unit for the X-axis in the file, "lambda" : wavelenght in nm, "freq": frequency in 10**15 Hz, "omega" angular frequency in rad/fs
- *smoothing*: bool , True to apply a smoothin to the experimental spectrum (recommanded for the calculations)

1.3 Common methods in class *Data_input* and *Analytics*

```
GetParameters(X,Y,precision=1.e-2)
```

Calculate the parameters X at Y-maximum , Y-maximum, and FWHM for a given profile. Return 3 floats.

- *X* : np.array, X axis
- *Y* : np.array, Y profile
- *precision*: optional, float, precision for the FWHM calculation

```
SetParameters(self, precision=1.e-2)
```

Calculate and update the frequency parameters of the pulse, ω_0 , Field maximum amplitude E_0 and frequency FWHM

- *precision*: optional, float, precision for the FWHM calculation

```
SetTimeParameters(precision=1.e-2)
```

Calculate and update the temporal parameters of the pulse, the delay , Field maximum amplitude E_0 and time FWHM (or half maximum duration) Store them in the dict parameters as 'w0': .. , 'E0_t': ... , 'HMD': ...

- *precision*: optional, float, precision for the FWHM calculation

```
FreqtoTime(update=True)
```

Compute the inverse fourier transform of the pulse

- *update*: optional, bool, True to have the result updated in the pulse object

```
TimetoFreq(update=True)
```

Compute the fourier transform of the pulse

- *update*: optional, bool, True to have the result updated in the pulse object

```
Save(file_name)
```

Save the pulse in a .dat file

- *file_name*: String, file name

```
Copy()
```

Copy the pulse and return a pulse object

Chapter 2

Components implementation

All the class are stored in the file *components.py*

2.1 Matter

```
Matter(name)
```

Implement an object which correspond to a transparent medium characterized by a refractive index n(λ)

- name : str() , matter name, all the matter already implemented can be printed with *Matter.media*

If you want to use a media not implemented, either implement or initialilize with a ramdom medium and change the configuration after the implementation using the representation of the Sellmeier equation:

$$n^2(\lambda) = A + \sum_i \frac{B_i \lambda^2}{\lambda^2 - C_i}$$

.

```
medium=Matter("BK7") #initialized the object
medium.name="new medium"
medium.indice=[A,[B1,C1],[B2,C2]...]
```

```
propagation(z,pulse_in, third_order=False)
```

Compute the propagation in the medium for a given distance z, return a pulse object corresponding to the pulse after the propagation

- *z* : float, distance of propagation in micrometer
- *pulse_in* : pulse object, initial optical pulse before the medium

- *third_order* : bool, consider True or not the third order porpagation, default=False

```
calculate_indice(l)
```

return the numerical value of the refractive index for a given wavelenght

- *l* : float, wavelenght in micrometer

```
derivatives(l)
```

return the numerical value of the different orders of the taylor developpment of k (k_0, k'_0, GVD, TOD), for a given wavelenght

- *l* : float, wavelenght in micrometer

2.2 Double Grating Compressor

Class *GratingCompressor*

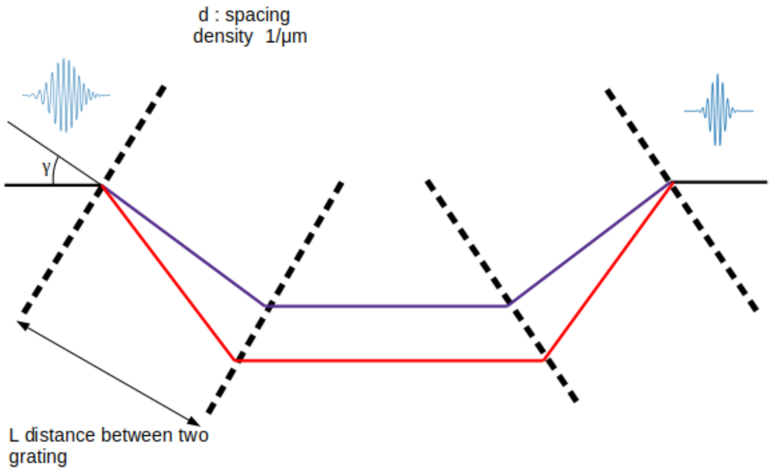

Figure 2.1: Double parallel gratting compressor

```
GratingCompressor(d)
```

Implement the object

- *d* : float, grating density in 1/micrometers

```
propagation(pulse_in, gamma, z)
```

Return a pulse object after the grating compressor

- *pulse_in* : pulse object, inital optical pulse
- *gamma* : float, incidence angle (see figure)
- *z* : float, distance in between the 2 gratings (see figure)

```
GDDcalculate(l0, gamma, z)
```

Return a numerical value of the grating compressor GDD

- *l0* : float, central wavelenght in micrometer
- *gamma* : float, incidence angle (see figure)
- *z* : float, distance in between the 2 gratings (see figure)

2.3 Double Prism Compressor

Class *DoublePrismCompressor*

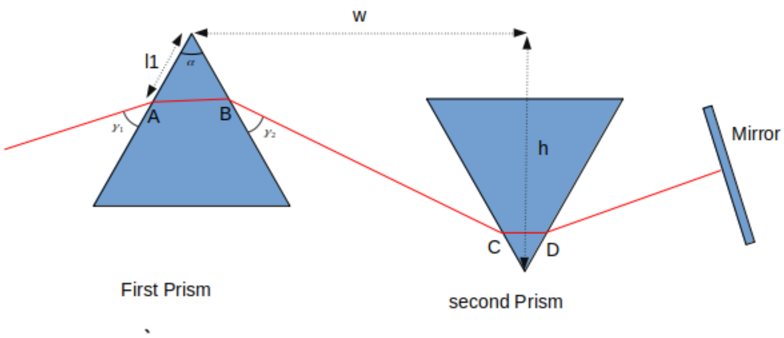

Figure 2.2: Double parallel prism compressor

```
DoublePrismCompressor(alpha=pi/3, matter_name="SiO2", Brewster=False)
```

Implement the object

- *alpha* : float, apex angle in radian
- *matter_name* : matter name of the prism , matter available with Matter.media, to change the media follow the steps presented in the class matter
- *Brewster* : bool, True to be in the brewster angle configuration

```
propagation(pulse_in, l1, w, h, gamma1=0)
```

Return a pulse object after the prism compressor

- *pulse_in* : pulse object, inital optical pulse
- *l1* : float, x distance entrance-apex in micrometer (see figure)
- *w* : float, x apex-apex distance in micrometer (see figure)
- *h* : float, y apex-apex distance in micrometer (see figure)
- *gamma1* : float, incidence angle in radian(see figure)

```
GDDcalculate(l0, l1, w, h, gamma1=0)
```

Return a numerical value of the prism compressor GDD

- *l0* : float, central wavelenght in micrometer
- *l1* : float, x distance entrance-apex in micrometer (see figure)
- *w* : float, x apex-apex distance in micrometer (see figure)
- *h* : float, y apex-apex distance in micrometer (see figure)
- *gamma1* : float, incidence angle in radian(see figure)

Chapter 3

Graphic

All the class and methods are stored in file *graphic.py*

```
BaseGraph(title):
```

Implement the graph object

- *title* : optional str, plot title

Methods

```
comparedouble(pulse1, pulse2)
```

Plot the frequency and the temporal representation of two pulses in the same window, the result is presented in the section example

- *pulse1* : pulse object, pulse to plot
- *pulse2* : pulse object, pulse to plot

```
compare(pulse1, pulse2)
```

Plot the frequency and the temporal representation of two pulses in different windows, the result is presented in the section example

- *pulse1* : pulse object, pulse to plot
- *pulse2* : pulse object, pulse to plot

```
plotattribut(pulse, X="X", Y="Y", legend_X="frequency 10^15 rad/s", legend_Y
    ="Field amplitude a.u",label="", absolute=False, **kwargs)
```

Plot an attribute of a pulse. List of the possible attributes to plot are

X	frequency absice in 10^{15}rad/s
Y	frequency amplitude in a.u
phase	spectral phase in rad
X_time	Time abscisse in fs
Y_time	Time amplitude in a.u
temporal_phase	temporal phase in rad

- *pulse* : pulse object, pulse to plot
- *X* : np.array, X-axis of the plot
- *Y* : np.array, Y-axis of the plot
- *legend_X* : string, legend for X-axis
- *legend_Y* : string, legend for Y-axis
- *absolute* : bool, True to take the absolute value of the signal

Chapter 4

Examples

```
pulse_in=pulse.Data_input("test.dat", axex="lambda")
pulse_in.SetParameters() #calculation of frequency parameters
pulse_in.SetTimeParameters() #calculation of temporal parameters

glass=components.Matter('SiO2') # creation of the material

pulse_out=glass.propagation(10000, pulse1) # propagation after 1 mm, z is given in micrometers
pulse_out.SetParameters() # calculation of the paramaters
pulse_out.SetTimeParameters()

graphique=graphic.BaseGraph()
graphique.comparedouble(pulse_in, pulse_out) #ploting the results
```

Listing 4.1: pulse from data input propagation

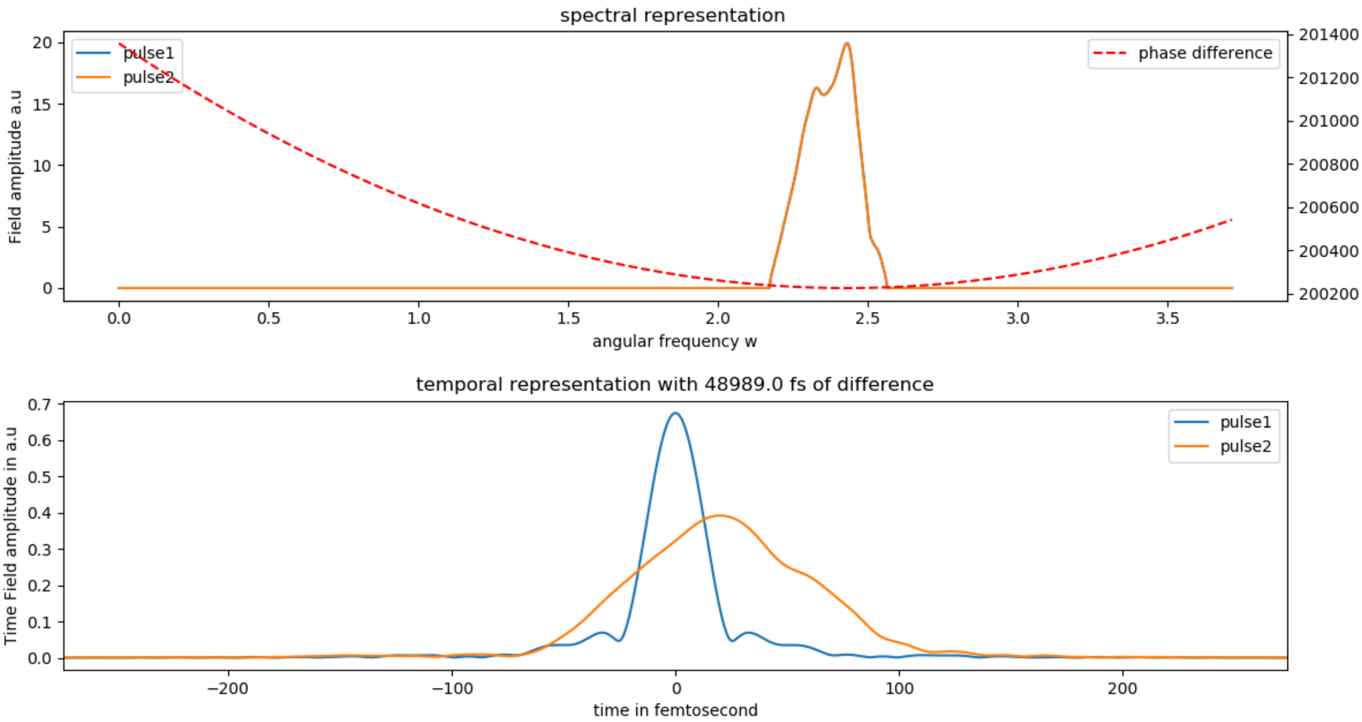

Figure 4.1: result window for the method *comparedouble*, propagation of an experimental pulse after 1000 of SiO2

```
pulse_in = pulse.Analytics(repre="time",shape="Gaussian", w0=2.5, E0=1, FWHM
    =10) #creation of a gaussian pulse with amplitude of 1  and half maximum
     duration of 10 fs
pulse_in.SetParameters() #calculation of frequency parameters

glass=components.Matter('BK7') # creation of the material

pulse_out=glass.propagation(1000, pulse_in) # propagation after 1000
    micrometers
pulse_out.SetParameters() # calculation of the paramaters
pulse_out.SetTimeParameters()

graphique=graphic.BaseGraph()
graphique.compare(pulse_in, pulse_out)
```

Listing 4.2: Analytical pulse propagation

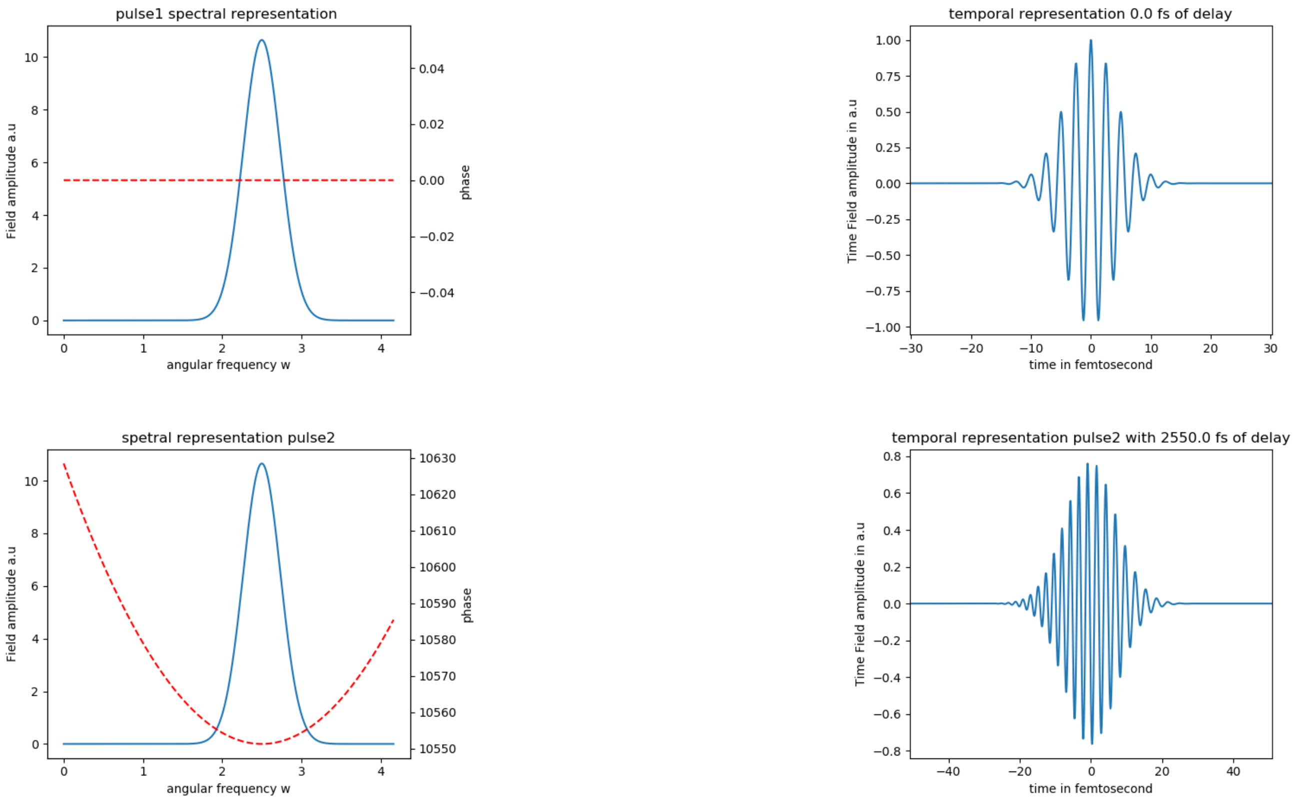

Figure 4.2: result window for the method *compare*, propagation of a gaussian pulse in 0.5mm of BK7 glass