

# Hussar 1.1 software manual

Tomasz M. Kardaś

May 9, 2017

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	License . . . . .	2
1.2	About Hussar . . . . .	2
1.3	Requirements . . . . .	3
1.4	Installation . . . . .	4
1.5	Citing . . . . .	4
1.6	Function and class references manual . . . . .	4
<b>2</b>	<b>Changes and new features</b>	<b>5</b>
2.1	Version 1.1 – 2017.05.10 . . . . .	5
<b>3</b>	<b>List of examples</b>	<b>7</b>
3.1	Tutorials . . . . .	7
3.2	Examples . . . . .	8
3.3	Tests . . . . .	10
<b>4</b>	<b>Tutorials walk-through</b>	<b>12</b>
4.1	OPA example . . . . .	12
4.2	1D Propagation . . . . .	26

# Chapter 1

## Introduction

### 1.1 License

This software can be used for non-commercial, educational and scientific purposes only.

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.



THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

### 1.2 About Hussar

Hussar [1] is an Object Oriented MATLAB library designed for pulse and continuous wave beam nonlinear propagation.

Hussar uses an unidirectional pulse propagation equation (UPPE [2]):

$$\partial_z E = ik_z E + \frac{\tilde{\omega}}{2\epsilon_0 c^2 k_z} (i\tilde{\omega}P - j) \quad (1.1)$$

modified according to not-necessary slowly-varying envelope approach:

$$\tilde{E}(t, x, y, z) = \sum_j \tilde{A}_j(t, x, y, z) e^{i(\omega_j t - k_z^j z)}$$

or:

$$E(\tilde{\omega}, k_x, k_y, z) = \sum_j A_j(\omega = \tilde{\omega} - \omega_j, k_x, k_y, z) e^{-ik_z^j z} \quad (1.2)$$

This modification does not involve additional approximations. It is used to introduce distinction between physical beams e.g. pump, signal and idler in OPA experiment or fundamental and second harmonic in SHG experiment. Additionally this reduces memory requirements. The  $P(\tilde{\omega}, k_x, k_y, z)$  and  $j(\tilde{\omega}, k_x, k_y, z)$  are the nonlinear part of the medium polarization and the free current, respectively.

Upon the substitution of 1.2, equation 1.1 decouples into  $N$  equations, where  $N$  equals to number of envelopes. At the same time the  $P$  and  $j$  split into  $N$  expressions for polarization and current oscillating at reference frequencies  $\omega_j$ .

For example in the case of second harmonic generation:

$$\begin{aligned} \partial_z A_F &= ik_z^F A_F + i \frac{d_{\text{eff}} \tilde{\omega}_F^2}{c^2 k_z^F} F \{A_F^* A_{SH}\} e^{i\Delta k z} \\ \partial_z A_{SH} &= ik_z^{SH} A_{SH} + i \frac{d_{\text{eff}} \tilde{\omega}_{SH}^2}{2c^2 k_z^{SH}} F \{A_F^2\} e^{-i\Delta k z} \end{aligned}$$

and sum frequency generation:

$$\begin{aligned} \partial_z A_1 &= ik_z^1 A_1 + i \frac{d_{\text{eff}} \tilde{\omega}_1^2}{c^2 k_z^1} F \{A_2^* A_3\} e^{i\Delta k z} \\ \partial_z A_2 &= ik_z^2 A_2 + i \frac{d_{\text{eff}} \tilde{\omega}_2^2}{c^2 k_z^2} F \{A_1^* A_3\} e^{i\Delta k z} \\ \partial_z A_3 &= ik_z^3 A_3 + i \frac{d_{\text{eff}} \tilde{\omega}_3^2}{c^2 k_z^3} F \{A_1 A_2\} e^{-i\Delta k z} \end{aligned}$$

The information on dispersion diffraction and spatial walk-off is contained in the wavevector  $z$  component matrix (for each set of  $\tilde{\omega}$ ,  $k_x$ , and  $k_y$ ):

$$k_z(\tilde{\omega}, k_x, k_y) = \sqrt{\left(\frac{\tilde{\omega} n(\tilde{\omega}, k_x, k_y)}{c}\right)^2 - k_x^2 - k_y^2}$$

this recursive equation is solved by Hussar iteratively, the refractive index is calculated based on the Sellmeier equations.

### 1.3 Requirements

Hussar was tested with MATLAB R2014a and 2014b it should, however, work with older versions which support `classdef` and “~” function argument notation.

Hussar has been created for PC computers. 4GB RAM is enough for running simulations with 3 envelopes (SFG/DFG/OPA etc.) and 4 million points in the grid of each envelope (e.g grid of 128x128x256 points in 3D and 1024x4096 points in 2D simulation).

## 1.4 Installation

Hussar pcode (.p) and Matlab (.m) files come as a compressed archive. In order to use Hussar it is enough to put it's location into MATLAB's path. Although this can be done permanently, the recommended way is to use the "includeAll.m" script which sets the MATLAB's path until MATLAB shutdown. This is useful especially when a new version of Hussar is obtained or multiple versions of Hussar are kept on a single computer.

To include Hussar into your MATLAB path run "includeAll.m" this can be done automatically by your script through the "run" command:

```
run('PathToHussarDirectory/includeAll'); %% include Hussar
```

## 1.5 Citing

Please cite Hussar and its authors in any publication for which you found it useful, thanks! The best publication to cite is:

T. M. Karda, M. Nejbauer, P. Wnuk, B. Resan, C. Radzewicz, and P. Wasylczyk, "Full 3D modelling of pulse propagation enables efficient nonlinear frequency conversion with low energy laser pulses in a single-element tripler," Scientific Reports 7, 42889 (2017). [1]

## 1.6 Function and class references manual

The detail description of functions and classes can be found at: [http://www.ufs.edu.pl/hussar\\_doc/index.html](http://www.ufs.edu.pl/hussar_doc/index.html). The manual is a work in progress – please don't hesitate to contact Tomasz Kardaś to get the details on the particular part of Hussar which was not yet documented (<mailto:tomasz.kardas@fuw.edu.pl>).

## Chapter 2

# Changes and new features

### 2.1 Version 1.1 – 2017.05.10

- *Reference Manual.* The function and class reference manual can now be found at [http://www.ufs.edu.pl/hussar\\_doc/index.html](http://www.ufs.edu.pl/hussar_doc/index.html).
- *Material naming.* Classes which represent materials no longer use the name preceding “C” convention. Therefore, instead of previously used CYVO4, CCalcite, CBBO, the YVO4, Calcite and BBO can be used. Old naming style is still available but it is deprecated and will be removed in future releases.
- *New Materials.* Many new materials including: AgGaGeS<sub>4</sub>, AgGaS<sub>2</sub>, BaF<sub>2</sub>, Cs<sub>2</sub>TeMo<sub>3</sub>O<sub>12</sub>, Cs<sub>2</sub>TeW<sub>3</sub>O<sub>12</sub>, GaAs, GaSe, Germanium, LiGaS<sub>2</sub>, LiIO<sub>3</sub>, LiNbO<sub>3</sub>, MgF<sub>2</sub>, N-BK7, N-SF10, N-SF11, SiC<sub>4</sub>H, Silicon, YAG, ZnS, ZnSe were implemented.
- *Four-Wave Mixing.* Two new examples concerning FWM have been added. First one shows the Nonlinear Polarization Evolution (NPE) in Polarization Maintaining (PM) fiber (see our publication [3] on this subject). This example features SPM/XPM and degenerate FWM. The second example presents a recently popular time lens implementation through FWM (3D).
- *Interface transfer.* In v. 1.0 interface transfer did not take the pulse polarization into account i.e. all pulses were treated as ordinary. This bug has now been fixed. For a few next releases the pulse polarization will have to be explicitly specified.
- *OPA.* An OPA process is now available it is in fact an alias for SFG.
- *Non-collinear propagation removed.* We have recently understood that, although used by others, the approach to non-collinear propagation leaned on spatial spectrum shift only is not exact. It is an approximation. We

will publish a paper on the subject of non-collinear propagation soon. Meanwhile, we have removed the possibility of using the old approach.

- *Material Files* The propagation constant ( $k_Z$ ) for a material is now not saved by default. Use additional bool parameter to CPropagationManager constructor to switch Material File saving on.

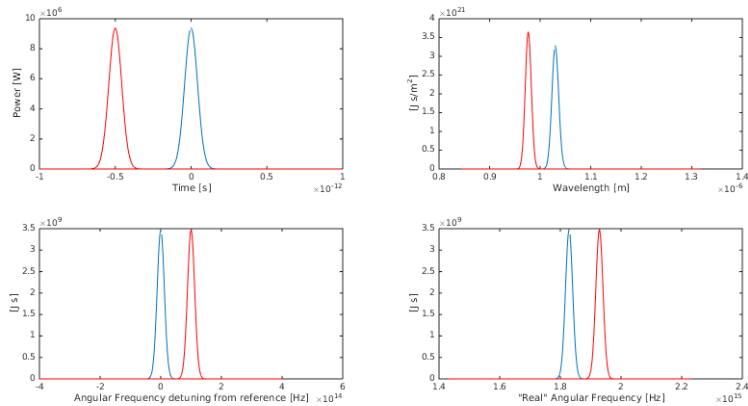
# Chapter 3

## List of examples

The best way to start using Hussar is through examples. Included are examples of OPA, SHG, SFG, SRS, SPM, Supercontinuum generation in a fiber and consecutive X-Frog measurement, SPIDER setup, linear effects (double refraction). Additionally as a preview tests of components required for 3D supercontinuum generation in bulk materials are provided. This includes Photoionization according to Keldysh and multi-photon model and Drude model for light interaction with free currents.

### 3.1 Tutorials

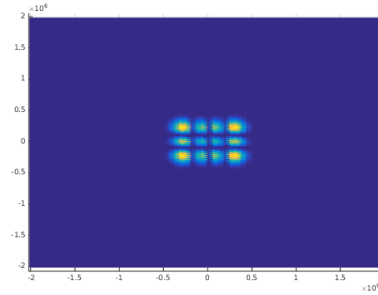
- Materials in Hussar - basic usage :  
/Hussar-1.1/work/tutorial/T1\_Materials.m
- Envelope creation and visualization:  
/Hussar-1.1/work/tutorial/T2\_Pulse.m





- 3D pulse composition, shifting in space, spectral phase manipulation, serialization, visualization.

/Hussar-1.1/work/tutorial/T3\_Pulses.m

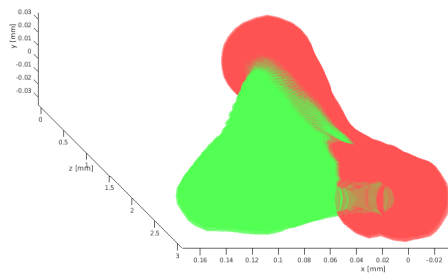


- SPM in 1D - simple propagation of big beam or in a fiber

/Hussar-1.1/work/tutorial/T4\_Propagation.m

- SHG - simple 3D propagation

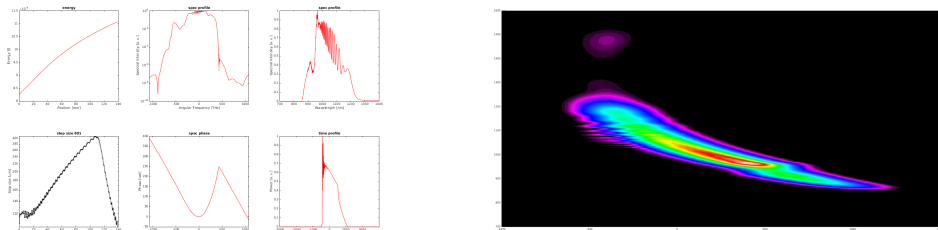
/Hussar-1.1/work/tutorial/T5\_SHG\_Movie.m



## 3.2 Examples

- fiber supercontinuum

/Hussar-1.1/work/example\_Fibers/FiberSupercontinuum\_XFROG.m



- Nonlinear Polarization Evolution in Polarization Maintaining Fibers - Features SPM/XPM, DFWM - see our publication [3]. for details.

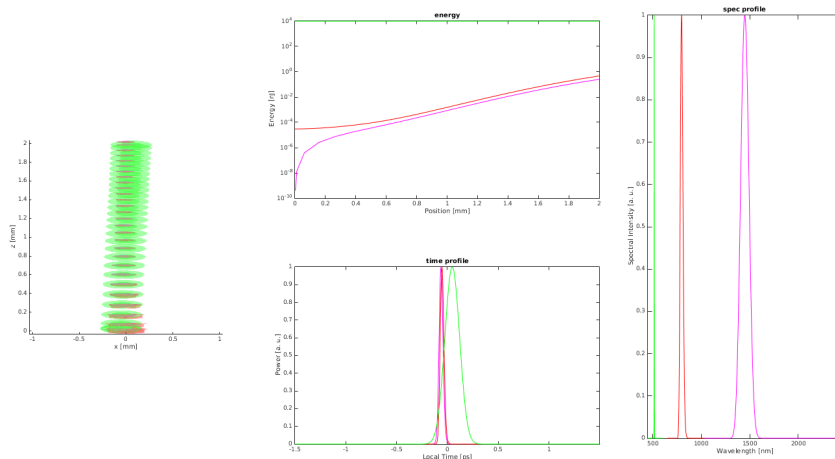
/Hussar-1.1/work/example\_Fibers/NPE\_in\_PM.m

- OPA - various configurations of OPA setups

/Hussar-1.1/work/example\_OPA/OPA\_FirstStage.m

/Hussar-1.1/work/example\_OPA/OPA\_SecondStage.m

/Hussar-1.1/work/example\_OPA/OPA2.m



- SRS, FSRS - Stimulated Raman Scattering with two long overlapping pulses with shear

/Hussar-1.1/work/examples/SRS\_TwoLongPulses.m

and overlapping short and long pulse (like in the femtosecond SRS experiment)

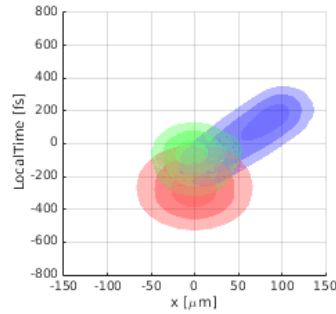
/Hussar-1.1/work/examples/SRS\_ShortAndLongPulse.m

- SHG - second harmonic generation with tightly focused beam

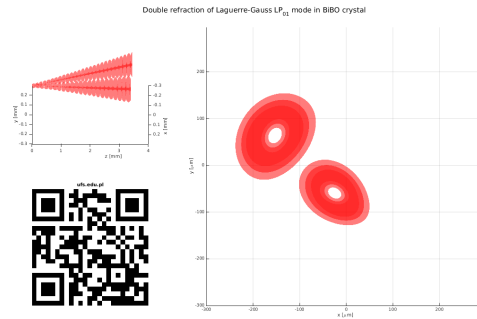
/Hussar-1.1/work/examples/Movie\_SHG.m

- THG - cascaded THG through consecutive SHG and SFG, the pulse delay and overlapped is fixed by a set of birefringent crystals (see our article [1] on the subject).

/Hussar-1.1/work/examples/Movie\_THG\_BBO.m



- Linear effects - double refraction (walk-off) in BiBO crystal



`/Hussar-1.1/work/examples/Movie_DoubleRefraction.m`

- SPIDER example  
`/Hussar-1.1/work/examples/SPIDER.m`
- XPM - cross-phase modulation between two overlapping pulses a short and a long one  
`/Hussar-1.1/work/examples/XPM_LongAndShortPulse.m`

### 3.3 Tests

- Material - calculation and visualization of electric field, displacement and Poynting vectors for slow and fast optical rays ( $\vec{E}_s, \vec{E}_f, \vec{D}_s, \vec{D}_f, \vec{P}_s, \vec{P}_f$ ) refractive index ( $n_s, n_f$ ), walk-off, dispersion terms ( $v_g, \beta_2 - \beta_3$ ) search for phase-matching conditions.  
`/Hussar-1.1/work/tests/testBiBO.m`
- Multiphoton absorption  
`/Hussar-1.1/work/tests/testMPA.m`  
`/Hussar-1.1/work/tests/testMPA_ThroughPICI.m`

- Self-focusing  
/Hussar-1.1/work/tests/testSelfFocusing.m
- third order dispersion  
/Hussar-1.1/work/tests/testTOD.m
- Drude model of electric field – current interaction  
/Hussar-1.1/work/tests/testDrude\_ThroughPICI.m
- Third harmonic generation  
/Hussar-1.1/work/tests/testTHG\_Processes.m

# Chapter 4

## Tutorials walk-through

### 4.1 OPA example

In this tutorial a collinear optical parametric amplifier pumped with a UV pulse will be simulated. The script is located in `PathToHussarDirectory/work/tutorial/T6_OPA.m`.

We begin by including Hussar:

```
%% include Hussar
run('../..//includeAll');
```

the “includeAll.m” script is located in `PathToHussarDirectory/` it is therefore pointed by a relative path `../..`

First the simulation’s grid and its physical dimensions have to be defined. This is done by creation of a `Cspace` class object:

```
%% space
space = Cspace('TXY');
```

In this case a Cartesian 3D grid will be constructed with temporal “T”, and two spatial “X” and “Y” dimensions. Alternative options are:

Cspace argument	notes
'T'	1D e.g. fiber
'TX'	2D Cartesian e.g. big non-collinear beams
'TXY'	Full 3D simulation e.g. focused beams in birefringent crystals
'TR'	cylindrical e.g. supercontinuum in non birefringent medium
'X'	CW beam in 1D
'XY'	CW beams in 2D, e.g. for birefringent media
'R'	cylindrical CW beam in non birefringent media e.g. self-focusing

The Temporal and spatial window sizes and the number of  $T, X, Y$  grid points can be set with `setDimension` method of `CSpace`:

```
fTimeSpan = 3e-12; % time window span in SI units [s]
iTimeSize = 2^9; % number of time grid points
space.setDimension('T', fTimeSpan, iTimeSize);
space.setDimension('X', 2e-3, 2^5);
space.setDimension('Y', 2e-3, 2^5);
```

Here a grid with size of  $2^9 \times 2^5 \times 2^5 = 512 \times 32 \times 32 = 524288$  points is created. The temporal window spans 3 ps, and spatial window is 2 mm in both X and Y directions. The grid size doesn't have to be a power of 2 it, however, has to be even. Also the X and Y direction sizes and points number don't have to be equal.

The envelopes of electric field related to particular beams are represented by `CEnvelope` class objects:

```
%% envelopes
% signal
fSignalWavelength = 800e-9;
AS = CEnvelope('A_S', space, fSignalWavelength);
```

The first argument is a label for the envelope which will be displayed by Listeners on various plots. This particular envelope represents the signal ( $A_S$ ). The space and the envelopes reference wavelength (corresponding to reference frequency  $\omega_S$ ) is a mandatory argument.

The electric field inside the envelope can be arbitrary (user data) or composed from “Pulse Functions” with `CPulseComposer` instance:

```
composer = CPulseComposer(space);
composer.append('T', CSincPF('FWHM', 2e-15));
composer.append('X', CGaussPF('Waist', 200e-6));
composer.append('Y', CGaussPF('Waist', 200e-6));

fSignalEnergy = 30e-9; % SI [J] (30e-9 J = 30 nJ)
AS.put(fSignalEnergy, composer);
```

The selected “Pulse Function” has to be passed to the `CPulseComposer` with the `append` method (or `append2D` for two dimensional functions). The `append` method takes the dimension designation string (one of: 'T', 'X', 'Y', 'R' or their mix like 'XY' for `append2D`) as the first argument. The second argument is the object of a class derived from `CPulseFunction` class. Currently available “Pulse Functions” are:

Gaussian	CGaussPF
super Gaussian space or time profile	CSuperGaussPF
super Gaussian spectrum	CSuperGaussSpectrumPF
Hermite-Gauss profile	CHermiteGaussPF
shape from user provided function	CArbitraryPF
hyperbolic secant space or time profile	CSecantPF
hyperbolic secant spectrum	CSecantSpectrumPF
sinc temporal or spatial profile	CSincPF
cylindrical (2D) super Gaussian profile	C2DSuperGaussPF
Laguerre-Gaussian beam profile	CLaguerreGaussPF

For each “Pulse Functions” a different parameters defining the distribution width can be used, check the reference manual [http://ufs.edu.pl/hussar\\_doc](http://ufs.edu.pl/hussar_doc) to see what are the possibilities for a particular pulse function. All constants in Hussar are in basic SI units, thus, `composer.append('T', CSincPF('FWHM', 2e-15))`; appends a  $\frac{\sin t/\tau}{t/\tau}$  temporal profile with intensity FWHM of  $2 \cdot 10^{-15}$  seconds (2 fs), while `composer.append('X', CGaussPF('Waist', 200e-6))`; appends a Gaussian spatial profile with  $200 \cdot 10^{-6}$  meters (200  $\mu m$ ). The signal energy `fSignalEnergy = 30e-9`; is  $30 \cdot 10^{-9}$  J (30 nJ) and the `fSignalWavelength = 800e-9`; sets the signal wavelength to 800 nm.

The final 1,2 or 3D envelope is constructed by product of the selected pulse functions for all used dimensions. This is done by use of the `put` method of `CEnvelope`.

In our example we would like to model a signal created through supercontinuum generation in sapphire. It would be best to tak the actual experimental spectrum of the supercontinuum. We limit ourselves, however, to simple model with a square like spectrum – the spectrum of a sinc function (note that there a `CSuperGaussianSpectrumPF` could be used here as well). We still have to chirp the pulse, this is done by applying the spectral phase equal to  $i\beta_2\omega^2 d_{\text{sapphire}}$ , where  $\beta_2$  is and  $d_{\text{sapphire}}$  are the group delay dispersion and thickness of a sapphire plate in which the supercontinuum is generated. To apply such a phase to the pulse we use the `Sapphire` material class (see further text for more on materials) and its `dispersion` method, which takes the wavelength, the order of dispersion and the value of the  $\theta$  angle as arguments:

```
% Signal pre stretching – could also be done through propagation
mStretch = Sapphire();
fStretchMaterialThickness = 5e-3;
fBeta2 = mStretch.dispersion(fSignalWavelength, 2, 0);
AS.addSpectralPhase([0 0 fBeta2*fStretchMaterialThickness]);
```

the `CEnvelope` class method `addSpectralPhase` is used to add a spectral phase defined by a Taylor series in frequency to the pulse. Here no constant and no linear phase term are present, thus, two first elements of the vector are equal to 0.

Apart of the Signal envelope the Pump and idler envelopes have to be defined.

```

% pump
fPumpWavelength = 515e-9;
AP = CEnvelope('A_P', space, fPumpWavelength);

composer = CPulseComposer(space);
composer.append('T', CGaussPF('FWHM', 170e-15));
composer.append('X', CGaussPF('Waist', 200e-6));
composer.append('Y', CGaussPF('Waist', 200e-6));

fPumpEnergy = 10e-6;
AP.put(fPumpEnergy, composer);

% idler
fIdlerWavelength = 1/(1/fPumpWavelength - 1/fSignalWavelength);
AI = CEnvelope('A_I', space, fIdlerWavelength);

```

There is no energy in the idler beam initially, thus, pulse composition is not required here.

The envelopes have to be grouped into a vector. For the case of three envelope interactions (SFG/DFG/OPA) the third envelope in the envelope vector has to have the shortest wavelength.

```

A(1) = AS;
A(2) = AI;
A(3) = AP;

```

The envelopes are constructed in vacuum. To transfer them into the non-linear material a material manager (`CMaterialManager`) for both materials is required. Here the manager for vacuum is prepared.

```

%% materials
% start in vacuum
vacuum = CVacuum();
mmVacuum = CMaterialManager(vacuum, 0);

```

The material manager constructor accepts an instance of material class (derived from `CMaterial`) in this case `CVacuum`. other possibilities currently available are:



AgGaGeS <sub>4</sub>	AgGaGeS4()
AgGaS <sub>2</sub>	AgGaS2()
BaF <sub>2</sub>	BaF2()
BBO	BBO()
BiBO	BiBO()
CaF <sub>2</sub>	CaF2()
Calcite	Calcite()
Cs <sub>2</sub> TeMo <sub>3</sub> O <sub>12</sub>	CTM()
Cs <sub>2</sub> TeW <sub>3</sub> O <sub>12</sub>	CTW()
Diamond	Diamond()
FusedSilica	FusedSilica()
GaAs	GaAs()
GaSe	GaSe()
Germanium	Germanium()
HgGa <sub>2</sub> S <sub>4</sub>	HgGa2S4()
KDP	KDP()
KTP	KTP()
LBO	LBO()
LiGaS <sub>2</sub>	LiGaS2()
LiIO <sub>3</sub>	LiIO3()
LiNbO <sub>3</sub>	LiNbO3()
MgF <sub>2</sub>	MgF2()
N-BK7	NBK7()
N-SF10	NSF10()
N-SF11	NSF11()
Quartz	Quartz()
Sapphire	Sapphire()
SiC <sub>4</sub> H	SiC_4H()
Silicon	Silicon()
Vacuum	Vacuum()
YAG	YAG()
YVO <sub>4</sub>	YVO4()
ZnS	ZnS()
ZnSe	ZnSe()
Birefringent fiber from other fiber and beat length	BirefringentFiber()
Material with dispersion Taylor series	FiberFromTaylor()
Material from a dispersion curve	FiberFromBeta2()

The second argument of `CMaterialManager` constructor is the thickness of the material, we don't intend to use this particular manager for propagation, but for transfer between media only. Thus the thickness is irrelevant and is set to 0 m here. For birefringent media the constructor of the material manager accepts also the values of angles ( $\theta$  and for biaxial materials also  $\varphi$ ) between the simulation axis and the optical axis of the crystal.

```
%nonlinear medium
m = BBO();
```

```
fCrystalThickness = 2e-3;
```

The `BBO` class derived from `CMaterial` provides several useful methods. In particular it can be used for calculation of phase matching angle and effective nonlinear coefficient:

```
fPhi = 90*pi/180;
fTheta = m.phaseMatch([fIdlerWavelength fSignalWavelength], ...
    ['o' 'o'], fPumpWavelength, 'e', pi/4);
deff = m.getDeff([fIdlerWavelength fSignalWavelength, ...
    fPumpWavelength], ['o' 'o' 'e'], fTheta, fPhi);

mm = CMaterialManager(m, fCrystalThickness, fTheta);
```

see `/tests/testBiB0.m` for more advance usage. Note that calculation of  $d_{\text{eff}}$  requires value of both  $\theta$  and  $\phi$  angle, the second one, however, does not influence the phase-matching, and, thus, has to be provided explicitly.

In this model we want to assure that the pulses meet in the center of the crystal and that's where the beams have their focuses. This is not necessarily the most optimal configuration and an optimization of parameters should in practice be performed. Anyway the group velocities, walk-off angles and the refractive index in BBO have to be calculated:

```
%% initial pulses temporal and spatial separation (cross the pulses in
% the center of the medium)
% group velocity mismatch
[fSigV] = m.groupVelocity(fSignalWavelength, fTheta, fPhi);
[~, fPumpV] = m.groupVelocity(fPumpWavelength, fTheta, fPhi);
GVM = (1/fSigV - 1/fPumpV); % s/m
```

The `groupVelocity` provides the values of group velocity (for the ordinary and extraordinary rays - first and second output argument respectively) at a given wavelength and propagation angle. For biaxial crystals the values for 'slow' and 'fast' ray are provided and the method requires additionally the value of  $\varphi$  angle.

Now, knowing the value of group velocity mismatch (GVM) the pump can be delayed by  $\tau = -\frac{d \text{GVM}}{2}$  so that it will meet in half the crystal thickness ( $\frac{d}{2}$ ) with the signal pulse. This can be done by addition of a spectral phase to the pulse as:

$$A(t - \tau) = F_T^{-1} \{ F_T \{ A(t) \} e^{i\omega\tau} \}.$$

The `addSpectralPhase` method of `CEnvelope` permits addition of arbitrary spectral phase approximated by a Taylor series. A vector of Taylor series coefficients has to be passed as an argument to `addSpectralPhase`.

```
AP.addSpectralPhase([0 0.5*GVM*fCrystalThickness]); % delay
```

Another useful method of `CMaterial` is `getWalkOffAngles`, it provides the values of walk-off angle (or two angles in biaxial crystals). It can be used for calculating the required spatial shift of the pump beam:

```

% walk-off and non-collinearity
[fWalkOffAngleE] = m.getWalkOffAngles(fPumpWavelength, fThetaPump);
fAlpha = -(fThetaPump-fThetaSig); % non-collinearity angle
fPumpXShift = -0.5* fCrystalThickness * (tan(fWalkOffAngleE)-tan(fAlpha));

AP.shiftInSpace('X', fPumpXShift);

```

The spatial shift is performed with the `shiftInSpace` method.

A pulse envelope created with the standard pulse functions with the pulse composer has no spatial or temporal phase - in other words it represents the envelope of a pulse in the beam focus. To obtain a divergent or convergent beam a forward or backward propagation in vacuum can be performed. Here we want to obtain the pump and signal beam focuses in the center of the crystal, we thus have to perform a back propagation by the distance equal to the half of the crystal thickness multiplied by the refractive index. The refractive index can be obtained by the `refractiveIndex` method of BBO class.

```

%% back propagation
% get the flat pulse front in the center of the crystal
[n] = m.refractiveIndex(fIdlerWavelength, fTheta);
vRefractiveIndices(1) = n;
[n] = m.refractiveIndex(fSignalWavelength, fTheta);
vRefractiveIndices(2) = n;
vBackPropagateLength(2) = 0.5*fCrystalThickness * n;
[~, n] = m.refractiveIndex(fPumpWavelength, fTheta);
vRefractiveIndices(3) = n;
vBackPropagateLength(3) = 0.5*fCrystalThickness * n;

```

A different back propagation distance for pump and signal are calculated (`vBackPropagateLength`), there's no need for back propagation of an empty idler. The back propagation for the pump and signal beam is now performed:

```

for it = 2:3 % only the signal and the pump
    Ai = A(it);

```

A `CMaterialManager` holding the material (vacuum) and its thickness is required:

```

backPropagateMM = CMaterialManager(vacuum, vBackPropagateLength(it));

```

The `CPropagationManager` class holds the material and the envelope information. The information on the polarization ('o' ordinary, 'e' extraordinary, 's' slow, 'f' fast) of the envelope is also provided here (for non birefringent media it should always be 'o').

```

pm = CPropagationManager(backPropagateMM, Ai, 'o');

```

A derivative provider (where the derivative refers to the right hand side of the UPPE) objects are used for selection of the modeled processes. A general `CProcessContainer` can be used for this purpose.

```

dp = CProcessContainer(pm);

```

Objects representing different processes can then be added to the process container (see the “Propagation” tutorial).

```
le = CLinearEffects('BackPropagation', 'on');
dp.addProcess(le);
```

Here an object of class `CLinearEffects` is added, this class represents the  $ik_z(\omega, k_x, k_y)A(\omega, k_x, k_y, z)$  part of the UPPE. The additional parameter `'BackPropagation'` enables back propagation. The solution method is also represented by an object.

The following methods are now available:

methods name	build in error estimation	order
Exponential Euler method	-	1
Runge-Kutta 4 method	-	4
Runge-Kutta 45 method 'Fehlberg', 'Cash-Karp', 'Dormand-Prince' (default)	+	4
Integrating Factor Runge-Kutta 45 method 'Fehlberg', 'Cash-Karp', 'Dormand-Prince' (default)	+	4

For nonlinear propagation the Integrating Factor Runge-Kutta 45 method is the one to start with. The Runge-Kutta 45 method should also perform well in most problems. The Exponential Euler method is also a good method for solving UPPE like problems, it is however, a 1 order method thus, reduction in step sizes might be required to keep the accuracy.

The linear problems can be solved in a single Fourier space step. The very idea of Runge-Kutta methods is subdivision of the step distance into substeps. The Exponential Euler method is best choice for linear problems as no subdivision is performed here.

```
ee = CExpEuler(dp, 1); % solution method Exponential Euler method
```

Here the parameter "1" refers to number of envelopes in simulation. The step size selection strategy is required for calculation of the solution. In this case a single step should be performed a `CConstantStepSizeStepper` enable division of the propagation distance into a number of equal steps (in this case 1).

```
stepper = CConstantStepSizeStepper(ee, 1);
```

At this point solution of the model (back propagation) could be performed, the user would, however, have no access to the data during solution. To plot the energy, temporal/spatial/spectral profile evolution, phase etc. the Listener object have to be provided to the stepper object. After every step of model solution each listener's `Listen` method is called and the current envelope vector, position within the medium and the step size are passed to the method. Therefore, various parameters can be extracted by the Listener object and saved or viewed on a plot.

The `CListenerFigure` has to be used for arrangement of various plots produced by the listener objects supplied with `Hussar`. Apart of the Matlab figure handle (second argument) the plot arrangement definition (same as for the standard Matlab `subplot` function) is supplied as an argument to the `CListenerFigure` constructor:

```
%% Listeners
caColor = {'m', 'r', 'g'}; % colors for the envelopes

hFig = figure('Position', [100+(it-1)*260, 100, 500, 800]);
lfigure = CListenerFigure([3 1], hFig);
```

Three vertical plot slots are created here on the `hFig`.

The energy listener can be used for plotting the energy during the propagation. The color of the line for each of the envelopes (one in this case) can be provided in the constructor.

```
hEnergyListener = CEnergyListener({caColor{it}});
```

The location of the energy listener plot on the figure has to be defined. The second argument to the `placeOn` function corresponds to the third argument of the Matlab `subplot` function.

```
hEnergyListener.placeOn(lfigure, [1]);
```

Finally the listener has to be added to the stepper's listeners list:

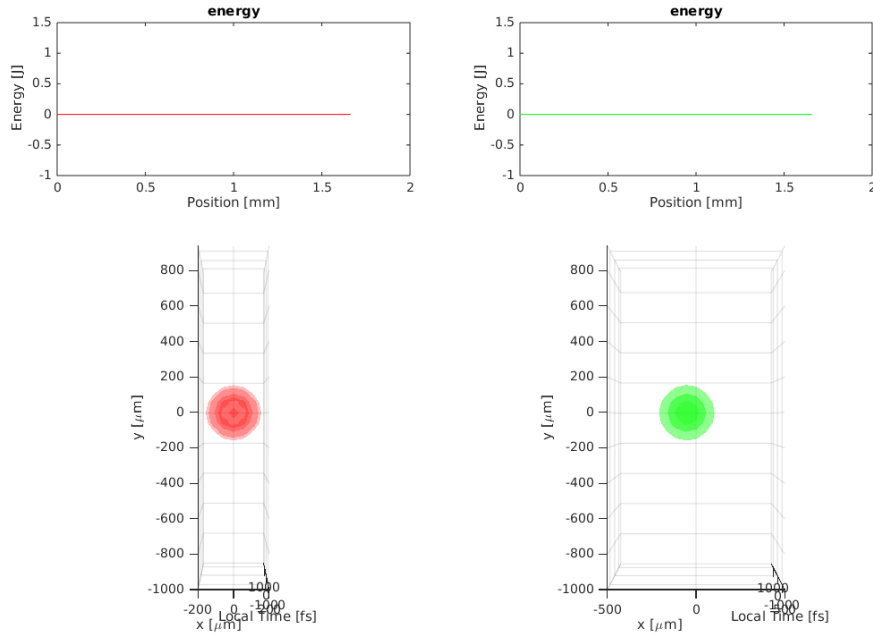
```
stepper.addListener(hEnergyListener);
```

The `C3DVisualizeListener` listeners is used for pulse visualization in the 3D space. It will be displayed on in the two bottom slots of the `lfigure` (note the `[2 3]` argument to `placeOn` method - in correspondance to standard Matlab `subplot` function behavior). The `C3DVisualizeListener` can be used to display the 3D visualization of the pulse from multiple perspectives in this case only one perspective will be used thus the last additional argument of the `placeOn` function is 1.

```
visual = C3DVisualizeListener({caColor{it}});
visual.placeOn(lfigure, [2 3], 1);
stepper.addListener(visual);
```

Finally the back propagation problem can be solved:

```
%% solve!
Ai = stepper.solve(Ai);
end
```



The envelopes have to be transferred into the material from the vacuum this is done by `CInterface` object (taking the material managers of the two media - vacuum and BBO) and its `transfer` method:

```
interface = CInterface(mmVacuum, mm);
A = interface.transfer(A, 'ooo', 'oe');
```

Again for propagation we need a propagation manager which apart of the material manager and envelopes will store the information on the polarization's of the envelopes.

```
% nonlinear propagation
pm = CPropagationManager(mm, A, 'oe');
```

This is a Type I OPA process with ordinary signal and idler rays 'oo' and extraordinary pump 'e'. The order of elements in the polarization argument 'oe' must correspond to the ordering of the envelopes' vector A.

A derivative provider is required:

```
dp = CProcessContainer(pm);
```

Now particular processes can be added to the simulation:

```
dp.addProcess(CLinearEffects());
```

```

dp.addProcess(OPA(deff));
n2 = 6e-20; % m^2/W
dp.addProcess(SPM(n2));
dp.addProcess(XPM(n2));

```

The OPA process requires value of  $d_{\text{eff}}$ . The self/cross-phase modulation together with self-steepening can be also added. The value of the nonlinear refractive indexes  $n_2 \left[ \frac{\text{m}^2}{\text{W}} \right]$  has to be provided..

The method `CRK45Method` or `CIFRK45Method` can preferably be used for solution.

```

% method = CRK45Method(dp, space, length(A), 'Dormand-Prince');
method = CIFRK45Method(dp, space, length(A), 'Dormand-Prince');

```

An automatic step size selection based on the embedded Runge-Kutta method error estimation is represented by `CHairerStepper` object.

```
stepper = CHairerStepper(method);
```

The stepper absolute and relative accuracy as well as the minimum step size can be set with the `setAccuracy` method.

```

fAccuracy = 1e-6;
fMinStepSize = 0.5e-6;
fMaxAmplitude = max(max(max(AP.m_mGrid)));
stepper.setAccuracy(fAccuracy, 0.1*fAccuracy*fMaxAmplitude, fMinStepSize);

```

Here the absolute accuracy has been selected in relation to the pump beam maximum envelopes amplitude and the minimum step size equal to pump wavelength is used.

To view the calculation results during the propagation again a set of listeners has to be defined:

```

%% Listeners

caColor = {'m', 'r', 'g'};

hFig = figure('Position', [100, 100, 1400, 800], 'Color', [0.8, 0.8, 1.0]);
set(hFig, 'Renderer', 'zbuffer');
lfigure = CListenerFigure([1 3], hFig);

hFig2 = figure('Position', [100, 100, 1400, 800], 'Color', [0.8, 0.8, 1.0]);
set(hFig2, 'Renderer', 'zbuffer');
lfigure2 = CListenerFigure([2 3], hFig2);

hFig3 = figure('Position', [100, 100, 1400, 800], 'Color', [0.8, 0.8, 1.0]);
set(hFig3, 'Renderer', 'zbuffer');
lfigure3 = CListenerFigure([2 2], hFig3);

hFig4 = figure('Position', [100, 100, 1400, 800], 'Color', [0.8, 0.8, 1.0]);
set(hFig4, 'Renderer', 'zbuffer');
lfigure4 = CListenerFigure([1 2], hFig4);

```

After preparing the figures, one can start filing.

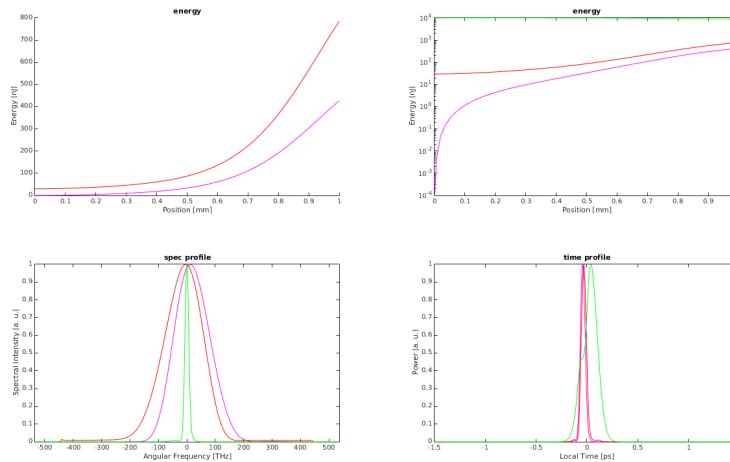
```

hEnergyListener = CEnergyListener(caColor);
hEnergyListener.setRefractiveIndex(vRefractiveIndices); % more or less
hEnergyListener.placeOn(lfigure3, 1);
hEnergyListener.setPlotType(@plot);
hEnergyListener.setVisible([1, 1, 0]);
hEnergyListener.setAxesUnits( 'Energy', 'nJ');
stepper.addListener(hEnergyListener);

hEnergyListener2 = CEnergyListener(caColor);
hEnergyListener2.setRefractiveIndex(vRefractiveIndices); % more or less
hEnergyListener2.placeOn(lfigure3, 2);
hEnergyListener2.setPlotType(@semilogy);
hEnergyListener2.setAxesUnits( 'Energy', 'nJ');
stepper.addListener(hEnergyListener2);

```

The energy is displayed in two figure, in first only the signal and idler envelopes are visible (`setVisible([1, 1, 0]);`), the scale is linear (`setPlotType(@plot);` – MATLAB `plot` function is used. In the second logarithmic vertical scale is used (`setPlotType(@semilogy);`, the other options like `@loglog` and `@semilogx` are also an option). The energy calculation within the medium requires information on the refractive index for each envelope, the values can be supplied with the `setRefractiveIndex(vRefractiveIndices)`.



```

bp11 = CProfileListener(1);
bp12 = CProfileListener(2);
bp13 = CProfileListener(3);
bp11.placeOn(lfigure2, 1, 1);
bp11.placeOn(lfigure2, 4, 2);
bp12.placeOn(lfigure2, 2, 1);
bp12.placeOn(lfigure2, 5, 2);
bp13.placeOn(lfigure2, 3, 1);
bp13.placeOn(lfigure2, 6, 2);

```

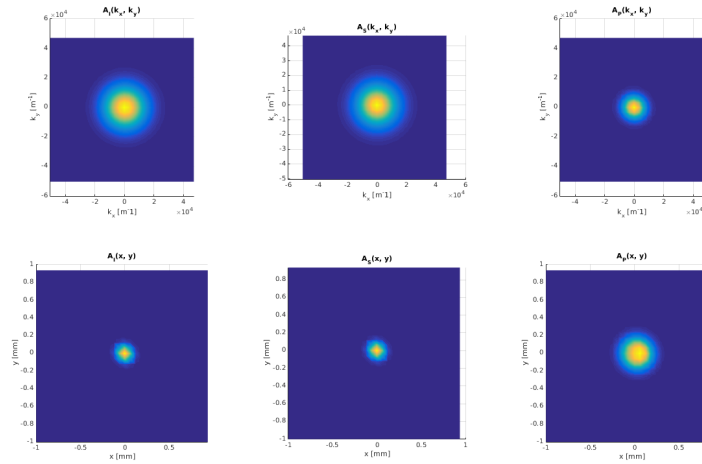


```

stepper.addListener(bpl1);
stepper.addListener(bpl2);
stepper.addListener(bpl3);

```

Six plots of the beam profiles and the spatial spectra are shown. Beam profile listeners `CProfileListener` provide two axes - one with the profile in the  $X - Y$  space and one spectral in the  $K_X - K_Y$  space. Therefore, for each beam profile listener the `placeOn` method has to be called twice. Single `CProfileListener` for each envelope (identified by the envelope index passed through the constructor (e.g. `CProfileListener(2)`)



```

visual = C3DVisualizeListener(caColor);
visual.placeOn(lfigure, 1, 1);
stepper.addListener(visual);

```

```

visual2 = CTrailListener(caColor);
visual2.placeOn(lfigure, 2, 1);
visual2.setGradientSteps([0.1 0.5]);
visual2.setAlpha(0.2);
visual2.setView(0,90);
stepper.addListener(visual2);

```

Two 3D visualizations from different points of view, the second one with memory are shown.

```

specL = CSpectrumListener(caColor, 'Wavelength');
specL.placeOn(lfigure, 3);
specL.normalize('on');
stepper.addListener(specL);

```

```

specL = CSpectrumListener(caColor);
specL.placeOn(lfigure3, 3);
specL.normalize('on');
stepper.addListener(specL);

```

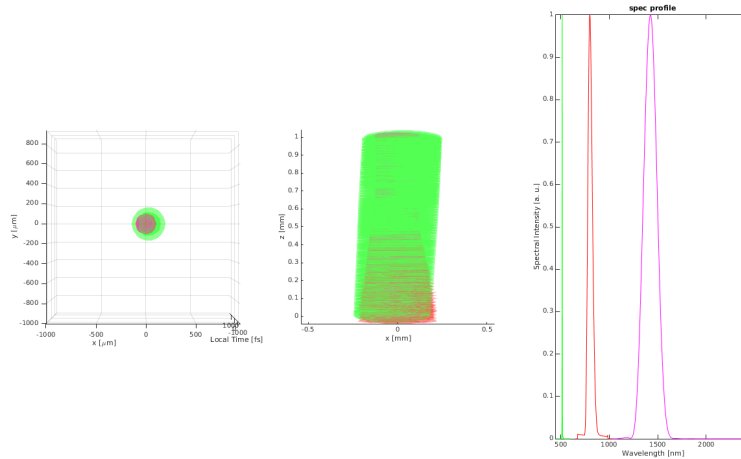
The spectra of the three pulses can be displayed, by default the x axis is angular frequency detuning from the reference frequency ('Angular Frequency') this can be changed by passing one of the following as the additional listener's constructor argument ('Wavelength', 'Frequency', 'Real Frequency', 'Real Angular Frequency'). By default the spectra are normalized this can be changed by calling the `normalize('off')` method.

```
tpl = CTimeProfileListener(caColor);
tpl.placeOn(lfigure3, 4);
tpl.setAxesUnits('Local Time', 'ps');
stepper.addListener(tpl);
```

And finally the temporal power distribution.

Finally the propagation problem can be solved:

```
%% solve
A = stepper.solve(A);
```



Afterwards the envelopes have to be transferred back to vacuum:

```
interface = CInterface(mm, mmVacuum);
A = interface.transfer(A, 'ooe', 'ooo');
```

Now the output characteristics like energy can be extracted:

```
disp(['output energy: ' num2str(1e12* AS.energy()) ' pJ']);
```

## 4.2 1D Propagation

A simple propagation of a large beam or within a fiber is described in this tutorial the corresponding script is located in: `PathToHussarDirectory/work/tutorial/T4_Propagation.m`.

We begin by including Hussar:

```
run('../../includeAll');
```

A single dimensional 'T' space is required.

```
space = CSpace('T');
space.setDimension('T', 1e-12, 2^8);
% space.setDimension('T', 1.8e-12, 2^10);
```

A domain spanning 1 ps and containing 256 points is used, however, as the user will see on the logarithmic spectrum and temporal profiles this will not be enough (as during the spectrum broadening the electric field will exit the simulation window, end enter it again on the other side). Thus finally the 1.8 ps domain of 1024 points will become a better choice.

```
fDuration = 30e-15;
composer = CPulseComposer(space);
composer.append('T', CGaussPF('FWHM', fDuration));
```

A 30 fs Gaussian pulse is declared with pulse composer.

```
fWL = 800e-9;
fEnergy = 10e-9;
fBeamWaist = 10e-6;
A = CEnvelope('A', space, fWL);
A.put(fEnergy, composer, fBeamWaist, fBeamWaist);
```

The pulse is construed as the pulse composer is passed to the `put` method of the `CEnvelope` (the reference wavelength `fWL` of 800 nm and energy of 10 nJ is used). As the model is one-dimensional to calculate the electric field the size of the beam (assumed Gaussian) in the X and Y directions has to be provided. This is done by two additional arguments to the `put` method. The beam size (presumably the fiber core radius) is 10  $\mu\text{m}$ .

The 10 mm of fused silica will be selected as the material, although, for fibers special materials that use dispersion Taylor expansion can be used instead.

```
%% material
m = CFusedSilica();
fThickness = 10e-3; % 10 mm
mm = CMaterialManager(m, fThickness);
```

Transfer into fused silica from vacuum where the pulse envelope have been defined is required and is done through interface object.

```

%% from vacuume into fused silica
hVac = CMaterialManager(CVacuum(), 0);
interface = CInterface(hVac, mm);
A = interface.transfer(A); % the electric field gets
% modified: E_2 = sqrt(n1/n2) E_1

```

Propagation manager will hold the material, envelope and the polarization information. For non-birefringent media should always be 'o'.

```

%% get ready for propagation
sPolarization = 'o';
bUseMaterialFiles = true;
pm = CPropagationManager(mm, A, sPolarization, bUseMaterialFiles);

```

In this tutorial a derivative provider is replaced by the CProcessContainer.

```

% dp = CDP1Env(pm);
dp = CProcessContainer(pm);

```

Various processes with different options can be added to the CProcessContainer:

```

%processes:
% dp.addProcess(CLinearEffects('SpatialEffects', 'off'));
dp.addProcess(CLinearEffects());

```

The linear effects are represented by CLinearEffects() class. The self-phase modulation (together with intrinsic self-steepening effect) is represented by SPM class. The self-steepening effect can be switche off by passing the 'SelfSteepening', 'off', 'ConstantRefractiveIndex', 'on' options to the SPM constructor.

```

n2 = 3e-20; % m^2/W
% dp.addProcess(SPM(n2, 'SelfSteepening', 'off', ...
% 'ConstantRefractiveIndex', 'on'));
dp.addProcess(SPM(n2));
% help SPM/SPM

```

Other possible processes currently available are:

LinearEffects	dispersion diffraction walk-off
LinearAbsorption	linear absorption
SPM	self-phase modulation
XPM	cross-phase modulation
DFWM	degenerate four-wave mixing
SRS	stimulated Raman scattering
THG	direct third harmonic generation
HOKE	higher order Kerr effects
MPA	multiphoton absorption
PICI	Photoionization and carier interaction (Multiphoton ionization or Keldysh model + Drude model)

The second harmonic and sum/difference frequency generation can be used via optimized 2-3 envelope derivative providers CDP2Env and CDP3Env (described in the NOPA tutorial).

In this case an Integrating Factor Runge-Kutta 45 solution method is the right choice. As can be verified by the user, when pure RK45 method is used artifacts on the frequency window edges appear.

```
%% RungeKutta 45 method
method = CRK45Method(dp, space, length(A), 'Dormand-Prince');
%%Integrating Factor Runge Kutta 45 method
% method = CIFRK45Method(dp, space, length(A), 'Dormand-Prince');
stepper = CHairerStepper(method); % does step size adaptation

fMaxAmplitude = max(max(max(abs(A(1).m_mGrid))));
fAccuracy = 1e-6;
stepper.setAccuracy(fAccuracy, 0.1*fAccuracy*fMaxAmplitude, fWL);
```

an alternative Exponential Euler method with Richardson Extrapolation step selection scheme is an other - a little slower and less accurate option.

```
%% alternative Exponential Euler method
% method = CExpEuler(dp, length(A));
% stepper = CRichardsonExtrapolationStepper(method);
% fGoalLocalError = 1e-4;
% fMinStepSize = fWL;
% stepper.setAccuracy(fGoalLocalError, fMinStepSize);
```

Uncomment following lines to enable data plotting (see NOPA tutorial for details on listeners usage). Energy, spectrum, time profile and step size are plotted on the figure represented by hFig.

```
%% Listeners
% hFig = figure('Position', [100, 100, 1200, 800], 'Color', [0.8, 0.8, 1.0]);
% lfigure = CListenerFigure([2 2], hFig);
%
% caColors = {'r'};
% hEnergyListener = CEnergyListener(caColors);
% hEnergyListener.setRefractiveIndex(m.refractiveIndex(fWL));
% hEnergyListener.placeOn(lfigure, 1);
% stepper.addListener(hEnergyListener);
%
% specL = CSpectrumListener(caColors);
% % specL.setPlotType(@semilogy);
% specL.placeOn(lfigure, 2);
% stepper.addListener(specL);
%
% tpl = CTimeProfileListener(caColors);
% % tpl.setPlotType(@semilogy);
% tpl.placeOn(lfigure, 3);
% stepper.addListener(tpl);
%
% ssl = CStepSizeListener();
% ssl.placeOn(lfigure, 4);
```

```
% stepper.addListener(ssl);
```

Finally solve the problem and transfer the electric field back to the vacuum.

```
A = stepper.solve(A);
```

```
interface = CInterface(mm, CMaterialManager(CVacuum(), 0));  
A = interface.transfer(A);
```

# Bibliography

- [1] Tomasz M. Kardaś, Michał Nejbauer, Paweł Wnuk, Bojan Resan, Czesław Radzewicz, and Piotr Wasylczyk. Full 3d modelling of pulse propagation enables efficient nonlinear frequency conversion with low energy laser pulses in a single-element tripler. *Scientific Reports*, 7:42889, February 2017.
- [2] M. Kolesik, J. V. Moloney, and M. Mlejnek. Unidirectional Optical Pulse Propagation Equation. *Physical Review Letters*, 89(28), December 2002.
- [3] Jan Szczepanek, Tomasz M. Kardaś, Czesław Radzewicz, and Yuriy Stepanenko. Ultrafast laser mode-locked using nonlinear polarization evolution in polarization maintaining fibers. *Optics Letters*, 42(3):575, February 2017.