Hussar 1.0 software manual

March 14, 2017

Contents

Chapter 1

Introduction

1.1 License

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

1.2 About Hussar

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

Hussar uses an unidirectional pulse propagation equation (UPPE [1]):

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

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

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

or:

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

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 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 ω_j.

For example in the case of second harmonic generation:

$$\partial_z A_F = ik_z^F A_F + i\frac{d_{\text{eff}}\tilde{\omega_F}^2}{c^2 k_z^F} F\left\{A_F^* A_{SH}\right\} e^{i\Delta k z}$$

$$\partial_z A_{SH} = ik_z^{SH} A_{SH} + i\frac{d_{\text{eff}}\tilde{\omega_{SH}}^2}{2c^2 k_z^{SH}} F\left\{A_F^2\right\} e^{-i\Delta k z}$$

and sum frequency generation:

$$\partial_z A_1 = ik_z^1 A_1 + i\frac{d_{\text{eff}}\tilde{\omega_1}^2}{c^2 k_z^1} F\left\{A_2^* A_3\right\} e^{i\Delta k z}$$

$$\partial_z A_2 = ik_z^2 A_2 + i\frac{d_{\text{eff}}\tilde{\omega_2}^2}{c^2 k_z^2} F\left\{A_1^* A_3\right\} e^{i\Delta k z}$$

$$\partial_z A_3 = ik_z^3 A_3 + i\frac{d_{\text{eff}}\tilde{\omega_3}^2}{c^2 k_z^3} F\left\{A_1 A_2\right\} e^{-i\Delta k z}$$

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 ".m" files come as a compressed archives. 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 changes MATLAB's path until MATLAB restart. 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).

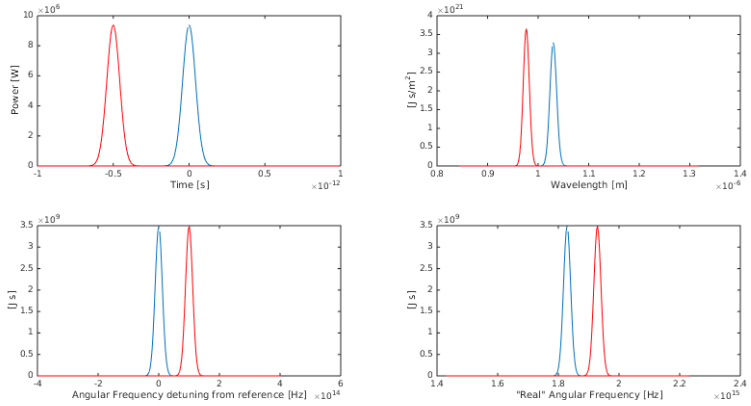
Chapter 2

List of examples

The bast way to start using Hussar is through examples. Included are examples of OPA, NOPA, 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.

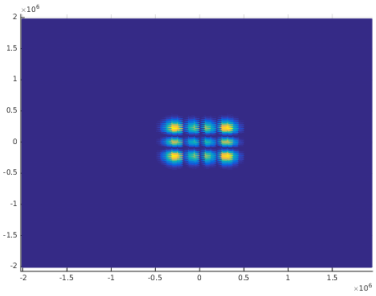
2.1 Tutorials

- Materials in Hussar - basic usage:
  `/Hussar-1.0/work/tutorial/T1_Materials.m`
- Envelope creation and visualization:
  `/Hussar-1.0/work/tutorial/T2_Pulse.m`



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

  `/Hussar-1.0/work/tutorial/T3_Pulses.m`

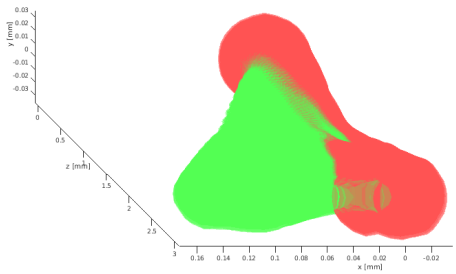


- SPM in 1D - simple propagation of big beam or in a fiber

  `/Hussar-1.0/work/tutorial/T4_Propagation.m`

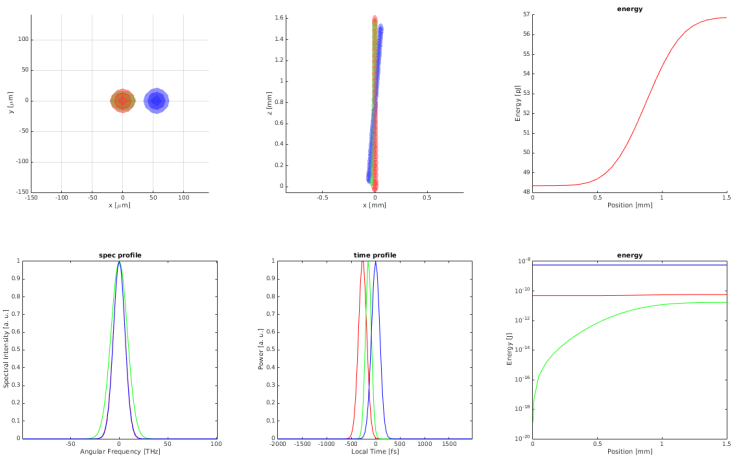
- SHG - simple 3D propagation

  `/Hussar-1.0/work/tutorial/T5_SHG_Movie.m`



- NOPA tutorial

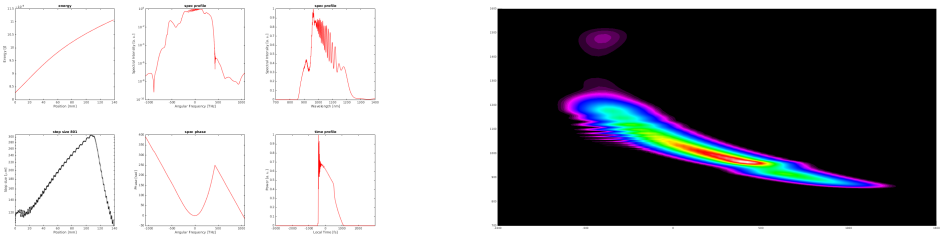
  `/Hussar-1.0/work/tutorial/T6_NOPA_PumpUV.m`



2.2 Examples

- fiber supercontinuum

  `/Hussar-1.0/work/example_FiberSupercontinuum_XFROG/FiberSupercontinuum_XFROG.m`

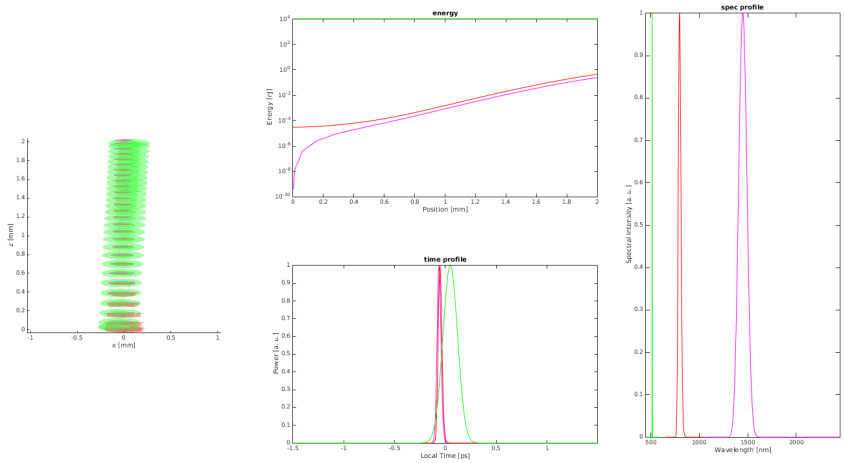


- OPA - various configurations of OPA setups

  `/Hussar-1.0/work/example_OPA/OPA_FirstStage.m`

  `/Hussar-1.0/work/example_OPA/OPA_SecondStage.m`

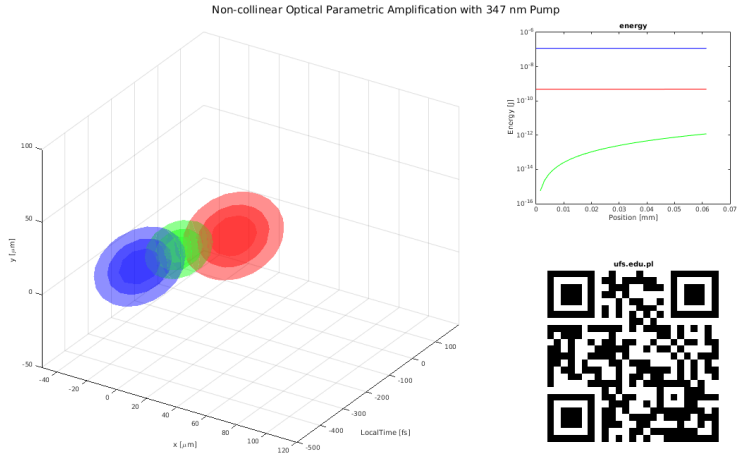
  `/Hussar-1.0/work/example_OPA/OPA2.m`



- NOPA

  `/Hussar-1.0/work/examples/NOPA_PumpUV1.m`

  `/Hussar-1.0/work/examples/Movie_NOPA_PumpUV.m`



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

  `/Hussar-1.0/work/examples/SRS_TwoLongPulses.m`

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

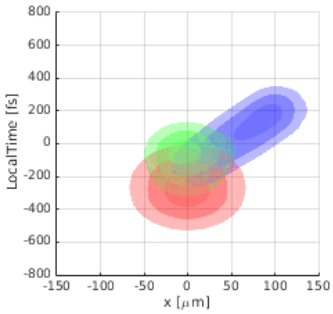
  `/Hussar-1.0/work/examples/SRS_ShortAndLongPulse.m`

- SHG - second harmonic generation with tightly focused beam

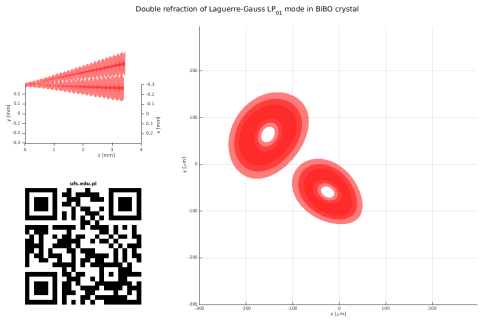
  `/Hussar-1.0/work/examples/Movie_SHG.m`

- THG - cascaded THG through consecutive SHG and SFG, the pulse delay and overlaped is fixed by a set of birefringent crystals

  `/Hussar-1.0/work/examples/Movie_THG_BBO.m`



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



  `/Hussar-1.0/work/examples/Movie_DoubleRefraction.m`

- SPIDER example

  `/Hussar-1.0/work/examples/SPIDER.m`

- XPM - cross-phase modulation between two overlaping pulses a short and a long one

  `/Hussar-1.0/work/examples/XPM_LongAndShortPulse.m`

2.3 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 $(v_g, \beta_2 - \beta_5)$ search for phase-matching conditions.

`/Hussar-1.0/work/tests/testBiBO.m`

- Multiphoton absorption
  `/Hussar-1.0/work/tests/testMPA.m`
  `/Hussar-1.0/work/tests/testMPA_ThroughPICI.m`
- Self-focusing
  `/Hussar-1.0/work/tests/testSelfFocusing.m`
- third order dispersion
  `/Hussar-1.0/work/tests/testTOD.m`
- Drude model of electric field – current interaction
  `/Hussar-1.0/work/tests/testDrude_ThroughPICI.m`
- Third harmonic generation
  `/Hussar-1.0/work/tests/testTHG_Processes.m`

Chapter 3

Tutorials walk-through

3.1 NOPA example

In this tutorial a non-collinear optical parametric amplifier pumped with a UV pulse will be simulated. The script is located in `PathToHussarDirectory/work/tutorial/T6_NOPA_PumpUV.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 birefringent media
'R'	cylindrical CW beam 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 = 4e-12; % time window span in SI units [s]
iTimeSize = 2^7;   % number of time grid points
space.setDimension('T', fTimeSpan, iTimeSize);
space.setDimension('X', 0.3e-3, 2^5);
space.setDimension('Y', 0.3e-3, 2^5);
```

Here a grid with size of $2^7 \times 2^5 \times 2^5 = 128 \times 32 \times 32 = 131072$ points is created. The temporal window spans 2 ps, and spatial window is 300 μm 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 = 1030e-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 ω_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', CGaussPF('FWHM', 200e-15));
composer.append('X', CGaussPF('Waist', 25e-6));
composer.append('Y', CGaussPF('Waist', 25e-6));

fSignalEnergy = 5.0000e-11; % SI [J]
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 use `help` (e.g. `help CGaussPF.CGaussPF`) to see what are the possibilities for a particular pulse function. All constants in Hussar are in basic SI units, thus, `composer.append('T', CGaussPF('FWHM', 200e-15));` appends a Gaussian temporal profile with intensity FWHM of $200 \cdot 10^{-15}$ seconds (200 fs), while `composer.append('X', CGaussPF('Waist', 25e-6));` appends a Gaussian spatial profile with $25 \cdot 10^{-6}$ meters (25 μm). The signal energy `fSignalEnergy = 5.0000e-11;` is $5 \cdot 10^{-11}$ J (50 pJ - not a very powerful NOPA) and the `fSignalWavelength = 1030e-9;` sets the signal wavelength to 1030 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`.

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

```
%pump
fPumpWavelength = 347e-9;
AP = CEnvelope('A_P', space, fPumpWavelength);

composer = CPulseComposer(space);
composer.append('T', CGaussPF('FWHM', 200e-15));
composer.append('X', CGaussPF('Waist', 25e-6));
composer.append('Y', CGaussPF('Waist', 25e-6));

fPumpEnergy = 6e-9; % = 6 nJ
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 nonlinear 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:

BBO	`CBBO()`
BiBO	`CBiBO()`
KDP	`CKDP()`
KTP	`CKTP()`
LBO	`CLBO()`
YVO_4	`CYVO4()`
ZnSe	`CZnSe()`
$HgGa_2S_4$	`CHgGa2S4()`
$LiNbO_3$	`CLiNbO3()`
CaF_2	`CCaF2()`
Calcite	`CCalcite()`
Sapphire	`CSapphire()`
Diamond	`CDiamond()`
FusedSilica	`CFussedSilica()`
Quartz	`CQuartz()`
Material with dispersion Taylor series	`CFiberFromTaylor()`
Material from a dispersion curve	`CFiberFromBeta2()`

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 (θ and for biaxial materials also φ) between the simulation axis and the optical axis of the crystal.

```
%nonlinear medium
m = CBBO();
fCrystalThickness = 1.5e-3;
```

The nonlinear propagation will be performed in a 1.5 mm BBO crystal. Five precalculated NOPA configurations are presented here and one can be chosen for simulation. These are the two configurations where the pump walk-off is compensated with either signal or idler beam direction, two configurations with

1° and 0.5° non-colinearity angle and a colinear case. The angles with respect to the optical axis are given.

```
%% fully walk-off signal compensated
% fThetaPump = 33.18*pi/180;
% fThetaIdl = 31.37*pi/180;
% fThetaSig = 37.35*pi/180;

%% fully walk-off idler compensated
% fThetaPump = 33.18*pi/180;
% fThetaIdl = 35.03*pi/180;
% fThetaSig = 28.98*pi/180;

%% non-collinearity = 1 deg
% fThetaPump = 32.39*pi/180;
% fThetaIdl = 31*pi/180;
% fThetaSig = 34.58*pi/180;

%% non-collinearity = 0.5 deg
% fThetaPump = 32.2*pi/180;
% fThetaIdl = 31.92*pi/180;
% fThetaSig = 32.65*pi/180;

%% colinear
fThetaPump = 32.16*pi/180;
fThetaIdl = 32.16*pi/180;
fThetaSig = 32.16*pi/180;
```

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
[fSignalGroupVelocity] = m.groupVelocity(fSignalWavelength, fThetaSig);
[~, fPumpGroupVelocity] = m.groupVelocity(fPumpWavelength, fThetaPump);
GVM = (1/fSignalGroupVelocity - 1/fPumpGroupVelocity); % s/m
```

The `CBBO` class derived from `CMaterial` provides several useful methods. 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 φ angle.

Now, knowing the value of group velocity mismatch (GVM) the pump can be delayed by $\tau = -\frac{d\ \mathrm{GVM}}{2}$ 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:

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

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 angles. Together with the non-collinearity angle in 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 `CBBO` class.

```
%% back propagation
% get the flat pulse front in the center of the crystal
n = m.refractiveIndex(fSignalWavelength, fThetaSig);
fBackPropagateLength = 0.5*fCrystalThickness * n ;
```

A different back propagation distance for pump and signal beams would be more appropriate (as the refractive indexes for the two beams differ slightly), we approximate it here with the same distance.

The back propagation for the pump and signal beam is now performed:

```
for it = [1, 3] % only the signal and the pump
    Ai = A(it);
```

A `CMaterialManager` holding the material (vacuum) an its thickness is required.

```
    mm = CMaterialManager(vacuum, fBackPropagateLength);
```

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(mm, Ai, 'o'); % ordinary polarization
```

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. Objects representing different processes can than be added to the process container (see the "Propagation" tutorial). For special cases of back propagation and sum frequency generation optimized derivative providers can be used. The `CDP1EnvBack` is a derivative provider for a single envelope (1Env) back propagation.

```
dp = CDP1EnvBack(pm); % single envelope 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
```

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 = {'r', 'g', 'b'}; % 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);
```

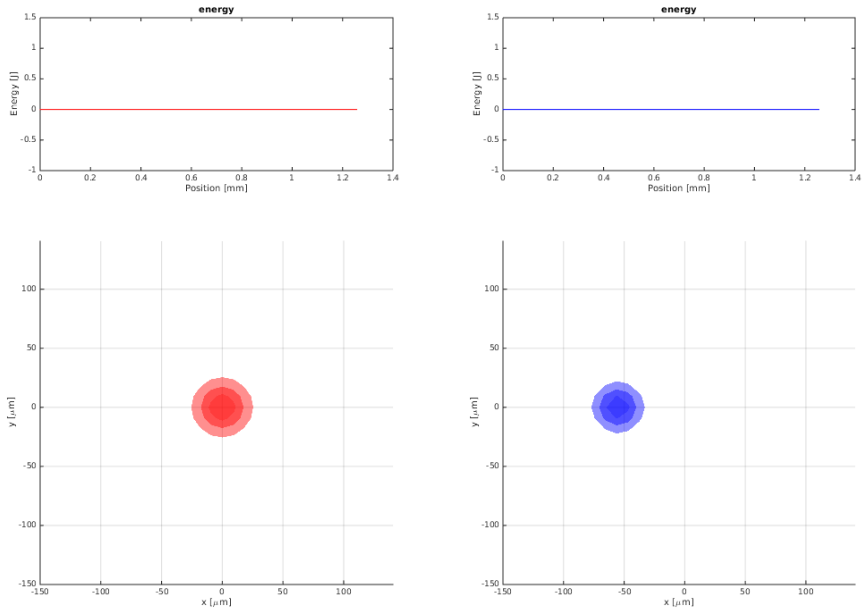
The `C3DVisualizeListener` listeners is used for pulse visualization in the 3D space. It will be displayed on in the two bottom slots of the `lfigure` (note the `[2 3]` argument to `placeOn` method - in correspondance to standard Matlab `subplot` function behavior). The `C3DVisualizeListener` can be 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
```



By default the pulses propagate along the models z axis. Before the NOPA process simulation this has to be changed. This is done by setting the envelope propagation direction with respect to the crystals optical axis with the `changePropagationDirection` method:

```
%% the direction of the Signal and idler beams with
% respect to the crystal axis
AS.changePropagationDirection(fThetaSig, 0);
AI.changePropagationDirection(fThetaIdl, 0);
%% from vacuum into the crystal
mm = CMaterialManager(m, fCrystalThickness, fThetaPump);
```

By defining the `CMaterialManager` with the `fThetaPump` we set the models z axis to make the `fThetaPump` angle with the crystal's optical axis. Therefore the pump beam will propagate co-linearly with the model's axis. The signal and idler beam will propagate at different angles defined with `changePropagationDirection`.

Currently the pulse rotation through the `changePropagationDirection` method is an approximated one. It is, however a good approximation within the $[-5°, 5°]$ angle range. The more exact way of rotation will come with the next Hussar version.

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);
```

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

```
%% nonlinear propagation

pm = CPropagationManager(mm, A, 'ooe'); % define the polarization
% of envelopes
```

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

A derivative provider optimized for three envelope's propagation is required:

```
dp = CDP3Env(pm); % use 3 envelopes
```

The sum/difference-frequency generation and optical parametric amplification all are in fact the same process with different initial conditions. To enable the OPA process the `addSFG` method of the derivative provider has to be called with the effective nonlinear coefficient ($d_{\text{eff}} = 2\frac{pm}{V} = 2 \cdot 10^{-12}\ \frac{m}{V}$ as en argument:

```
dp.addSFG(2.01e-12); % add sum frequency generation
```

The self/cross-phase modulation together with self-steepening can be also added through `addPhaseModulation` method taking the values of nonlinear refractive indexes n_2 $[\frac{m^2}{W}]$ for the three beams. This is skipped here.

```
% dp.addPhaseModulation(n2S, n2I, n2P);
```

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

```
% Runge-Kutta 45 method
method = CRK45Method(dp, space, length(A), 'Dormand-Prince');
% integrating factor Runge-Kutta 45 method
% 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;
fMaxAmplitude = max(max(max(AP.m_mGrid)));
stepper.setAccuracy(fAccuracy, 0.1*fAccuracy*fMaxAmplitude, fPumpWavelength);
```

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

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

bpl1 = CProfileListener(1);
bpl2 = CProfileListener(2);
bpl3 = CProfileListener(3);
bpl1.placeOn(lfigure2, 1, 1);
bpl1.placeOn(lfigure2, 4, 2);
bpl2.placeOn(lfigure2, 2, 1);
bpl2.placeOn(lfigure2, 5, 2);
bpl3.placeOn(lfigure2, 3, 1);
bpl3.placeOn(lfigure2, 6, 2);
stepper.addListener(bpl1);
stepper.addListener(bpl2);
stepper.addListener(bpl3);
```

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)`)

```
hFig = figure('Position', [100, 100, 1400, 800], 'Color', ...
   [0.8, 0.8 ,1.0]);
set(hFig,'Renderer','zbuffer');
lfigure = CListenerFigure([2 3], hFig);
```

The energy calculation within the medium requires information on the refractive index for each envelope, here an approximate value (1.6) is given, setting the plot type to `@semilogy` will make the Y axis logarithmic (other possible plot types are `@loglog`, `@semilogx` and the default: `@plot`).

```
hEnergyListener = CEnergyListener(caColor);
hEnergyListener.setRefractiveIndex(1.6*[1 1 1]); % more or less
hEnergyListener.placeOn(lfigure, [6]);
hEnergyListener.setPlotType(@semilogy);
stepper.addListener(hEnergyListener);
```

Another energy plot this time presenting only signal beam (`setVisible`) and with pico Joules on the `'Energy'` axis is prepared (the second `'Position'` axis units can also be changed from default `'mm'` to for example `'um'`).

```
hEnergyListener2 = CEnergyListener(caColor);
hEnergyListener2.setRefractiveIndex(1.6*[1 1 1]); % more or less
hEnergyListener2.placeOn(lfigure, [3]);
hEnergyListener2.setAxesUnits('Energy', 'pJ');
hEnergyListener2.setVisible([true false false]);
stepper.addListener(hEnergyListener2);
```

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:

```
specL = CSpectrumListener(caColor);
specL.placeOn(lfigure, 4);
stepper.addListener(specL);
```

The pulse time profile can be piloted with `CTimeProfileListener`

```
tpl = CTimeProfileListener(caColor);
tpl.placeOn(lfigure, 5);
stepper.addListener(tpl);
```

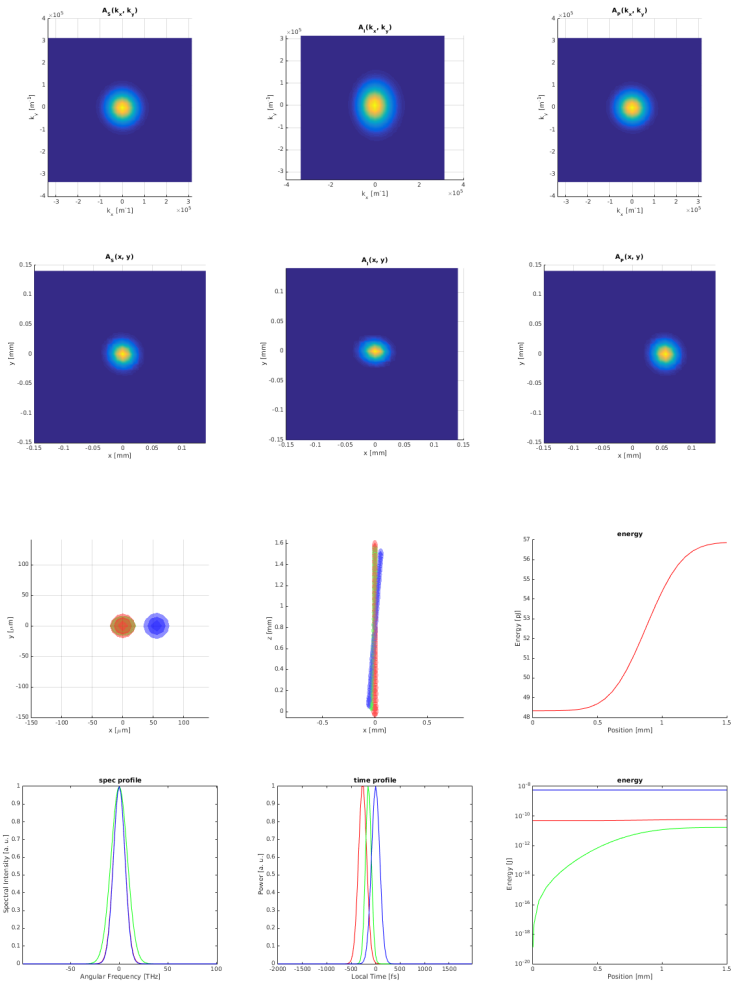
```
visual = C3DVisualizeListener(caColor);
visual.placeOn(lfigure, 1, 1);
stepper.addListener(visual);
```

`CTrailListener` can be used for plotting the pulses in the 3D XYZ space.

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

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);
```

Now the output characteristics like energy can be extracted:

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

Apparently for these particular conditions the co-linear configuration gives the highest output energy.

3.2 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 μm.

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 difraction 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] M. Kolesik, J. V. Moloney, and M. Mlejnek. Unidirectional Optical Pulse Propagation Equation. *Physical Review Letters*, 89(28), December 2002.