XNLO

Documentation

S. M. Senior

University of Southampton

March 22, 2017

Acknowledgments

Abstract

Contents

1 Introduction

2 Compiling XNLO

2.1 Prerequisites

2.1.1 CMake

CMake is an open-source system that manages the build process of a project in an operating system and compiler-independent manner, it can be obtained from www.cmake.org. CMake is used in XNLO to build the makefile that is then run to build the execuatable.

CMake version 3.1 is required, though version 3.6.2 has been tested and is known to work, so later version to 3.1 should work as well.

2.1.2 GCC

The C++ compiler of GCC is used to compile the source code of XNLO, though other compilers that support the C++14 standard and the OpenMP standard should also work (provided the CMakeLists.txt and comp files are altered to use that compiler). GCC can be obtained from www.gcc.gnu.org.

The ISO C++ Standard C++14 is required and G++ version 6.3.0 has been tested and is known to work. GCC should be installed without multilib support so that OpenMP is supported in it.

As a general note, if using OSX then GCC needs to be installed and linked to properly in either the comp file or the CMakeLists.txt file as the system links gcc and g++ on OS X link to the Clang compiler rather than a preinstalled GCC compiler.

2.1.3 MPI

MPI is a Message Passing Interface standard that is a communication protocol for programming on parallel computers. There are several implementations of MPI with the one being used in XNLO being OpenMPI, though other implementations such as MPICH should also work.

OpenMPI can be obtained from www.open-mpi.org. OpenMPI version 2.0.1 has been tested and is known to work, as has version 2.02, though it was found that for 2.0.2 the environment variable TMPDIR had to be explicitly set whereas in 2.0.1 it did not. To do this on UNIX it can either be set by running the command `export TMPDIR=/tmp` or by running XNLO with the command `TMPDIR=/tmp mpiexec -np X ./XNLO` instead of the previous command mpiexec -np X ./XNLO for version 2.0.1, where X is the number of threads.

If OpenMPI is installed on OS X through Homebrew then by default it wraps around Clang and Clang++ instead of gcc and g++. To fix this either build OpenMPI from source using gcc-6 or find the files **mpic++-wrapper-data.txt**, **mpicc-wrapper-data.txt**, and **mpicxx-wrapper-data.txt** and change the compiler option inside them to g++, gcc, and g++, respectively. The location of these files can be found by running the command `find /usr/local -name mpicc-wrapper-data.txt`. Though not the nicest work around it should work as Clang and GCC should adhere to the same ABI so mixing object code produced by these different compilers on the same platform should be fine (www.stackoverflow.com/questions/26812780/linking-homebrew-compiled-openmpi-or-mpich2-to-homebrews-gcc, the second answer).

2.1.4 Intel MKL

The Intel Maths Kernal Library (Intel MKL) is an optimised maths library for for such things as science applications. It can be obtained from for free from www.software.intel.com/en-us/intel-mkl.

Intel MKL version mkl_2017.0.102 has been tested and is known to work, though newer version should also work.

2.1.5 Eigen

Eigen is a C++ template library for linear algebra. It can be obtained from www.eigen.tuxfamily.org, though comes with XNLO and so shouldn't need to be downloaded.

2.2 Compiling

A cmakelists.txt file has been provided that has been tested, and is known to work, on macOS Sierra version 10.12.2. Minor modification may be needed to be made to it for to work on other operating systems. While it has been kept fairly generic, it has been written to work with g++ and so changes to things such as the compiler flags made be needed for it to work with other compilers. The cmakelists.txt links all the correct library and source files needed for a successful compilation and generates a make file, which when executed builds the executable.

A bash script 'version_incrememnt' has been provided that increments the version number when the correct input is provided, which is talked about in more detail in the Scripts section. As this is written in bash, it is depended on the machine running a UNIX OS or Windows with Cygwin or MinGW64 installed.

A bash script 'comp' has been provided in the build directory that can pass an argument to and run version_incrememnt, sets the correct environment for Intel MKL to be found, runs CMake whilst making sure MPIC++ is the specified compiler (which wraps around g++), and then compiles and builds the excutable. This has the effect of being able to run all the commands needed for a compilation from a single script. As it is a bash script it only works on a UNIX OS or Windows with Cygwin or Mingw64 installed.

3 Running XNLO

For XNLO to run OpenMPI, or an equivalent implementation of MPI, needs to be installed on the system. Also, XNLO needs to run on a minimum of two threads. To run XNLO use the following command

```
TMPDIR=/tmp mpiexec -np X ./XNLO-vY
```

where `X` is the number of threads and `Y` is the versio nnumber (such as `1-1-0`). The number of threads has to be two or more as the number of atoms is determined from the number of threads such that one thread corresponds to zero atoms.

By default XNLO looks for a config file named `config_file.txt` in the same directory as the executable from which it reads in the input parameters and settings, if this file cannot be found then XNLO uses default values set in the source code. A different config file can be specified by passing the flag `-cf` followed by the path and name of the config file as input arguments to XNLO, e.g.

```
TMPDIR=/tmp mpiexec -np X ./XNLO-vY -cf ../PathTo/my_config.txt
```

3.1 Input Parameters and Settings

This section gives an overview of how the config file works and then discusses the various input parameters and settings that can be specified in it.

3.1.1 The Config File

The config file can be used to set various variables and their descriptions. Each line in the file is for a single variable and consists of three pairs curly braces. The first pair of braces contains the name of the variable, if the variable named does not exist or cannot be set in this way then it is skipped with a warning message. The second pair of braces contains the value for that variable. For ints and doubles, the variable can be written in a normal notation or in an engineering notation. The third pair of braces contains a

description of the variable. This description could be used to give a brief overview of what the vairable does and the type needed for the vairable. The default decriptions all start with the string `(default)` to help differentiate when a variable is being set correctly from a config file and when it is just taking the dault value. A minimal example of a config file is given below

```
A minimal example of a config file, anything outside braces is
treated as a comment
{t_min}       {-100E-15}            {(double) Start time}
{l_0}         {795E-9}              {(double) Central wavelength}
{path_dipole} {./output/dipole.txt} {(std::string) Dipole path}
```

The way the config file is read in to XNLO means that anything outside a pair of braces is treated as a comment and is ignored. This also means that spaces can be put between two pairs of braces to make rows easier to read by lining up neatly. If there are less than three pairs of braces on a line then that line is skipped and isn't read in. If there are more than three pairs of braces on a line then only the first three braces are read in and the rest ignored. If a config file sets only some variables and not all of them then the vairables not set by it take the default values set in the source code.

3.1.2 atoms_per_worker

- Variable Name: **atoms_per_worker**
- Default Value: **2**
- Default Description: **(default) (int) The number of atoms per worker**

3.1.3 x_min

- Variable Name: **x_min**
- Default Value: **0**
- Default Description: **(default) (double) The x_min value**

3.1.4 x_max

- Variable Name: **x_max**
- Default Value: **0**
- Default Description: **(default) (double) The x_max value**

3.1.5 N_t

- Variable Name: **N_t**
- Default Value: **262144**
- Default Description: **(default) (int) The N_t value**

3.1.6 t_min

- Variable Name: **t_min**
- Default Value: **-100e-15**
- Default Description: **(default) (double) The t_min value**

3.1.7 t_max

- Variable Name: **t_max**
- Default Value: **100e-15**
- Default Description: **(default) (double) The t_max value**

3.1.8 P_av

- Variable Name: **P_av**
- Default Value: **0.18**
- Default Description: **(default) (double) The P_av value**

3.1.9 RR

- Variable Name: **RR**
- Default Value: **1000**
- Default Description: **(default) (double) The RR value**

3.1.10 Full Width at Half Maximum (FWHM)

The full width at half maximum duration of the laser pulse, in units of seconds.

- Variable Name: **FWHM**
- Default Value: **15e-15**
- Default Description: **(default) (double) The FWHM value**

3.1.11 Central Wavelength of the Laser (l_0)

The central wavelength of the laser, in units of meters.

- Variable Name: **l_0**
- Default Value: **795e-9**
- Default Description: **(default) (double) The l_0 value**

3.1.12 CEO

- Variable Name: **CEO**
- Default Value: **0**
- Default Description: **(default) (double) The CEO value**

3.1.13 spot_radius

- Variable Name: **spot_radius**
- Default Value: **42e-6**
- Default Description: **(default) (double) The spot_radius value**

3.1.14 alpha

- Variable Name: **alpha**
- Default Value: **1.45**
- Default Description: **(default) (double) The alpha value**

3.1.15 pend_path

The switch to `preprend` or `postpend` the path, or to turn off the pending with `false`.

- Variable Name: **pend_path**
- Default Value: **prepend**
- Default Description: **(default) (std::string) Pending switch**

3.1.16 path_dipole

The output path `dipole`.

- Variable Name: **path_dipole**
- Default Value: **./output/dipole.txt**
- Default Description: **(default) (std::string) Output path to dipole.txt**

3.1.17 path_w

The output path of `w`.

- Variable Name: **path_w**
- Default Value: **./output/w.txt**
- Default Description: **(default) (std::string) Output path to w.txt**

3.1.18 Config File Path (path_config_file)

The input path of the config file.

- Variable Name: **path_config_file**
- Default Value: **./config.txt**
- Default Description: **(default) (std::string) config.txt path**

3.1.19 Config Log Path (path_config_log)

The output path of the config log file.

- Variable Name: **path_config_log**
- Default Value: **./output/config_log.txt**
- Default Description: **(default) (std::string) Output path to config_log.txt**

3.2 Output

Two data files are outputted at the end of a simulation. By default these take names of the form

- `dipole.txt`
- `w.txt`

These files are written in a plain text format.

A log file is also generated which takes the default name

- `config_log`.txt

This file is a log of all the input parameters and settings used in the simulation that generated the other files and can be used as an easy reference to see which parameters were used for a simulation run. The file is written as in a plain text format. The input parameters and settings are outputted in this file in the same format used to read them in so that UPPE can be run again, pointing to a particular `config_log.txt` file as the input file, in case it is desired to rerun UPPE with that particular configuration.

All files outputted can have a prepending or postpending of a three digit number, in the form `XYZ`, i.e. the file `dipole.txt` can have a prepending or postpending

- `XYZ_dipole.txt`
- `dipole_XYZ.txt`

respectively, where `XYZ` is a three digit number. The default behaviour is to have a `prepend`, but a `postend` can be set in the config file, as can `false` to switch off pending. The three digit number `XYZ` is an automatic pending of the filename that is done so that the data from a previous run of UPPE is not overwritten. For prepending the digits are inserted at the start of the filename, but after the path to the file. For postpending the digits are inserted at the end of the filename but before the start of the file extension. They start at `000` and increment by one every time a file is found to exist with the current number. All the files of a run are outputted with a particular number `XYZ` so that they are grouped together.

If this number `XYZ` increments past `999` then the `X` of that number continues to increment on its own as if it was its own integer, i.e. it would become `10`, then `11`, then `12`, etc.

3.3 Scripts

A python script is in the main directory. This script reads in the data and plots it on various graphs. Reading this scripts should be able to give an

idea of how the files are formatted.

4 Technical Overview of XNLO

Apart from `main.cpp` and `version.hpp`, each set of source files (the `.cpp` and associated `.hpp` file) each contains a single class. An overview is given for each set of files, with a brief description of what each one does, as well as a brief overview of the input arguments needed when initialising each class, and the public data and functions.

4.1 config_settings

Reads in input parameters and settings from a config file.

4.1.1 Public Functions

- `void read_in(std::string)`
  Reads in the input parameters and settings found in the file located at `path`.
- `void check_paths()`
  Checks that the output paths already exist. If they do and the pending switch is on then the incrementing number is added on to the file name and incremented until an output path is found that does not yet exist.
- `void print()`
  Prints the input parameters and settings names, values, and descriptions to the standard output.
- `void print(std::string)`
  Prints the input parameters and settings names, values, and descriptions to the file specified in `path_`.

4.1.2 Private Functions

- `void set_variable(std::string& variable_name, std::string& variable_value_str, std::string& input_description_char)`
  Sets the variable `variable_name` to the value of `variable_value_sr` and

sets the variable description to `input_description_char`. Each variable has an expected type and when a variable value is set the variable value in string form is converted to that expected type. Currently there are no checks that `variable_value_str` can be converted to the correct type and therefore trying to pass in the wrong type could cause undefined behaviour.

- `std::string set_path(std::string path, std::string pending_string)`
  Passes the variable path and pending string on to the relevant pending function, depending on the value of the pend switch, i.e. `set_pre_path` if the pending switch has the value `prepend` or `set_post_path` if the pending switch has the value `postpend`. If the pending switch has the value `false` then the file name remains unchanged.

- `std::string set_pre_path(std::string pre_path, std::string path)`
  Prepends the file name with the incrementing number `XYZ_`, where `XYZ` is inserted before the file name but after the path to the file's directory.

- `std::string set_post_path(std::string path, std::string post_path)`
  Appends the file name with the incrementing number `_XYZ`, where `XYZ` is inserted after the file name but before the file extension, if the extension exists.

4.1.3 Private Data

- `enum class SN{ ... }`
  A scoped enumeration containing a list of the variables that can be set through use of the config file. It ends with the value `LAST_SN_ENTRY` for ease of finding out the length of the enumeration.

- `static const char * setting_name[]`
  An array of character arrays of the names of the variables that can be set through use of the config file. These names have to match the order of those in the scoped enumeration `SN` to allow for correct use of switches in `set_variable`.

4.2 grid_tw

"grid_tw" is a linear temporal grid. The spectral counterpart of this grid is evaluated and made accessible.

4.2.1 Class Constructor

To initialise an instance of grid_tw the following parameters are needed to be passed in:

- `int N_t_`
- `double t_min_`
- `double t_max_`

4.2.2 Public Data

- `ArrayXd t`
- `ArrayXd w`
- `int N_t`
- `double t_min`
- `double t_max`
- `double dt`

4.3 grid_xkx

"grid_xkx" is a linear 1D spatial grid. The spectral counterpart of this grid is evaluated and made accessible.

4.3.1 Class Constructor

To initialise an instance of grid_xkx with constructor arguments the following parameters are needed to be passed in:

- `int N_x_`
- `double x_min_`
- `double x_max_`

4.3.2 Public Data

- `ArrayXd x`
- `ArrayXd kx`
- `int N_x`
- `double x_min`
- `double x_max`
- `double dx`

4.4 laser_pulse

"laser_pulse" contains a time varying electric field. The intital conditions are passed to the constructor and field can be updated as it propagates.

4.4.1 Class Constructor

To initialise an instance of laser_pulse the following parameters are needed to be passed in:

- `double P_av_`
- `double RR_`
- `double FWHM_`
- `double l_0_`
- `double CEO_`
- `double spot_radius_`

- `double ROC_`
- `grid_xkx xkx_`
- `grid_tw tw_`

4.4.2 Public Data

- `ArrayXXd E`

4.5 main

4.6 maths_textbook

"maths_textbook" is a container for mathematical constants.

4.6.1 Public Data

- `double pi`

4.6.2 Public Functions

- `double trapz(ArrayXd x_, ArrayXd y_)`

4.7 physics_textbook

"physics_textbook" is a container for physical constants.

4.7.1 Public Data

- `double E_at`
- `double l_at`
- `double m_at`

- `double q_at`
- `double t_at`
- `double w_at`
- `double c`
- `double eps_0`
- `double mu_0`
- `double h_bar`
- `double k_B`

4.8 Schrodinger_atom_1D

"Schrodinger_atom_1D" encapsulates the interaction of an isolated atom with a strong laser field. The interaction is restricted to a single active electron and spatial dimension. Atomic units.

4.8.1 Class Constructor

To initialise an instance of Schrodinger_atom_1D the following parameters are needed to be passed in:

- `double alpha_`

4.8.2 Public Data

- `grid_xkx xkx`
- `double alpha`
- `ArrayXd V_model`
- `ArrayXcd wfn_GS`
- `ArrayXcd wfn`
- `double energy`

4.8.3 Public Functions

- `void set_GS(int N_it_)`
  Finds the ground state wavefunction by imaginary time propagation.
- `ArrayXd get_dipole(int N_it_, double dt_, ArrayXd E_)`
  Finds the dipole moment generated by an arbitary field.
- `ArrayXd solve_TDSE_PS(int N_it_, std::complex<double> dt_, ArrayXd E_, int e_)`
  Solves the time dependent Schrodinger equation by a pseudo-spectral method.

4.9 text

"text" is a tab-delimited file.

4.9.1 Class Constructor

To initialise an instance of text the following parameters are needed to be passed in:

- `text(ArrayXXd data, std::string path);`

4.10 version

5 Compiling UPPE

5.1 Prerequisites

5.1.1 CMake

CMake is an open-source system that manages the build process of a project in an operating system and compiler-independent manner, it can be obtained from www.cmake.org. CMake is used in UPPE to build the makefile that is then run to build the execuatable.

CMake version 3.1 is required, though version 3.6.2 has been tested and is known to work, so later version to 3.1 should work as well.

5.1.2 GCC

The C++ compiler of GCC is used to compile the source code of UPPE, though other compilers that support the C++14 standard and the OpenMP standard should also work (provided the CMakeLists.txt and comp files are altered to use that compiler). GCC can be obtained from www.gcc.gnu.org.

The ISO C++ Standard C++14 is required and G++ version 6.3.0 has been tested and is known to work. GCC should be installed without multilib support so that OpenMP is supported in it.

As a general note, if using OSX then GCC needs to be installed and linked to properly in either the comp file or the CMakeLists.txt file as the system links gcc and g++ on OS X link to the Clang compiler rather than a preinstalled GCC compiler.

5.1.3 MPI

MPI is a Message Passing Interface standard that is a communication protocol for programming on parallel computers. There are several implementations of MPI with the one being used in UPPE being OpenMPI, though other implementations such as MPICH should also work.

OpenMPI can be obtained from www.open-mpi.org. OpenMPI version 2.0.1 has been tested and is known to work, as has version 2.02, though it was found that for 2.0.2 the environment variable TMPDIR had to be explicitly set whereas in 2.0.1 it did not. To do this on UNIX it can either be set by running the command `export TMPDIR=/tmp` or by running UPPE with the command `TMPDIR=/tmp mpiexec -np X ./UPPE` instead of the previous command mpiexec -np X ./UPPE for version 2.0.1, where X is the number of threads. As UPPE only needs one thread to run it can also been called with the command `TMPDIR=/tmp ./UPPE`, though it is recommended to run it with mpiexec and specifying the number of threads.

If OpenMPI is installed on OS X through Homebrew then by default it wraps around Clang and Clang++ instead of gcc and g++. To fix this either build OpenMPI from source using gcc-6 or find the files **mpic++-wrapper-data.txt**, **mpicc-wrapper-data.txt**, and **mpicxx-wrapper-data.txt** and change the compiler option inside them to g++, gcc, and g++, respectively. The location of these files can be found by running the command `find /usr/local -name mpicc-wrapper-data.txt`. Though not the nicest work around it should work as Clang and GCC should adhere to the same ABI so mixing object code produced by these different compilers on the same platform should be fine (www.stackoverflow.com/questions/26812780/linking-homebrew-compiled-openmpi-or-mpich2-to-homebrews-gcc, the second answer).

5.1.4 Intel MKL

The Intel Maths Kernal Library (Intel MKL) is an optimised maths library for for such things as science applications. It can be obtained from for free from www.software.intel.com/en-us/intel-mkl.

Intel MKL version mkl_2017.0.102 has been tested and is known to work, though newer version should also work.

5.1.5 Eigen

Eigen is a C++ template library for linear algebra. It can be obtained from www.eigen.tuxfamily.org, though comes with XNLO and so shouldn't

need to be downloaded.

5.2 Compiling

A cmakelists.txt file has been provided that has been tested, and is known to work, on macOS Sierra version 10.12.2. Minor modification may be needed to be made to it for to work on other operating systems. While it has been kept fairly generic, it has been written to work with g++ and so changes to things such as the compiler flags made be needed for it to work with other compilers. The cmakelists.txt links all the correct library and source files needed for a successful compilation and generates a make file, which when executed builds the executable.

A bash script 'version_incrememnt' has been provided that increments the version number when the correct input is provided, which is talked about in more detail in the Scripts section. As this is written in bash, it is depended on the machine running a UNIX OS or Windows with Cygwin or MinGW64 installed.

A bash script 'comp' has been provided in the build directory that can pass an argument to and run version_incrememnt, sets the correct environment for Intel MKL to be found, runs CMake whilst making sure MPIC++ is the specified compiler (which wraps around g++), and then compiles and builds the excutable. This has the effect of being able to run all the commands needed for a compilation from a single script. As it is a bash script it only works on a UNIX OS or Windows with Cygwin or Mingw64 installed.

6 Running UPPE

For UPPE to run OpenMPI, or an equivalent implementation of MPI, needs to be installed on the system. To run UPPE use the following command

```
TMPDIR=/tmp mpiexec -np X ./UPPE-vY
```

where `X` is the number of threads and `Y` is the versio nnumber (such as `1-1-0`).

By default UPPE looks for a config file named `config_file.txt` in the same directory as the executable from which it reads in the input parameters and settings, if this file cannot be found then UPPE uses default values set in the source code. A different config file can be specified by passing the flag `-cf` followed by the path and name of the config file as input arguments to UPPE, e.g.

```
TMPDIR=/tmp mpiexec -np X ./UPPE-vY -cf ../PathTo/my_config.txt
```

6.1 Input Parameters and Settings

This section gives an overview of how the config file works and then discusses the various input parameters and settings that can be specified in it.

6.1.1 The Config File

The config file can be used to set various variables and their descriptions. Each line in the file is for a single variable and consists of three pairs curly braces. The first pair of braces contains the name of the variable, if the variable named does not exist or cannot be set in this way then it is skipped with a warning message. The second pair of braces contains the value for that variable. For ints and doubles, the variable can be written in a normal notation or in an engineering notation. The third pair of braces contains a description of the variable. This description could be used to give a brief overview of what the vairable does and the type needed for the vairable. The default decriptions all start with the string `(default)` to help differentiate

when a variable is being set correctly from a config file and when it is just taking the dault value. A minimal example of a config file is given below

```
A minimal example of a config file, anything outside braces is
treated as a comment
{n_m}          {20}                    {(int) Number of modes}
{l_0}          {800E-9}                {(double) Central wavelength}
{path_A_w_R}   {./output/A_w_R.bin}    {(std::string) A_w_R path}
```

The way the config file is read in to UPPE means that anything outside a pair of braces is treated as a comment and is ignored. This also means that spaces can be put between two pairs of braces to make rows easier to read by lining up neatly. If there are less than three pairs of braces on a line then that line is skipped and isn't read in. If there are more than three pairs of braces on a line then only the first three braces are read in and the rest ignored. If a config file sets only some variables and not all of them then the vairables not set by it take the default values set in the source code.

6.1.2 Number of Propagation Steps in Z (n_z)

The number of steps to propagate the laser pulse along the z-axis.

- Variable Name: **n_z**
- Default Value: **50**
- Default Description: **(default) (int) Number of steps in Z**

6.1.3 n_r

- Variable Name: **n_r**
- Default Value: **20**
- Default Description: **(default) (int) The n_r value**

6.1.4 Number of Modes (n_m)

The number of modes present.

- Variable Name: **n_m**
- Default Value: **20**
- Default Description: **(default) (int) Number of modes**

6.1.5 n_t

- Variable Name: **n_t**
- Default Value: **4096**
- Default Description: **(default) (int) The n_t value**

6.1.6 T

- Variable Name: **T**
- Default Value: **500.0e-15**
- Default Description: **(default) (double) The T value**

6.1.7 Minimum Positive Angular Frequency (w_active_min)

The minimum positive angular frequency to sample from.

- Variable Name: **w_active_min**
- Default Value: **2.0e14**
- Default Description: **(default) (double) Minimum angular frequency**

6.1.8 Maximum Positive Angular Frequency (w_active_max)

The maximum positive angular frequency to sample to.

- Variable Name: **w_active_max**
- Default Value: **8.0e15**
- Default Description: **(default) (double) Maximum angular frequency**

6.1.9 Length of the Capillary (Z)

The length of the capillary, in units of meters.

- Variable Name: **Z**
- Default Value: **5.0e-3**
- Default Description: **(default) (double) Length of capillary**

6.1.10 Radius of the Capillary (R)

The radius of the capillary, in units of meters.

- Variable Name: **R**
- Default Value: **75.0e-6**
- Default Description: **(default) (double) Radius of capillary**

6.1.11 Capillary Tube Gas Pressure (press)

The pressure of the gas in the capillary, in units of bars.

- Variable Name: **press**
- Default Value: **100.0e-3**
- Default Description: **(default) (double) Pressure of the gas**

6.1.12 p_av

- Variable Name: **p_av**
- Default Value: **1.0**
- Default Description: **(default) (double) The p_av value**

6.1.13 rep

- Variable Name: **rep**
- Default Value: **1.0e3**
- Default Description: **(default) (double) The rep value**

6.1.14 Full Width at Half Maximum (fwhm)

The full width at half maximum duration of the laser pulse, in units of seconds.

- Variable Name: **fwhm**
- Default Value: **40e-15**
- Default Description: **(default) (double) Full width at half max**

6.1.15 Central Wavelength of the Laser (l_0)

The central wavelength of the laser, in units of meters.

- Variable Name: **l_0**
- Default Value: **800e-9**
- Default Description: **(default) (double) Laser central wavelength**

6.1.16 ceo

- Variable Name: **ceo**
- Default Value: **0.0**
- Default Description: **(default) (double) The ceo value**

6.1.17 waist

- Variable Name: **waist**
- Default Value: **48.0e-6**
- Default Description: **(default) (double) The waist value**

6.1.18 Pre/Post-Pending Switch for Output Files (pend_path)

The switch to `preprend` or `postpend` the path, or to turn off the pending with `false`.

- Variable Name: **pend_path**
- Default Value: **prepend**
- Default Description: **(default) (std::string) Pending switch**

6.1.19 Input Path to `J0_zeros.bin` (path_input_j0)

The input path of the file `J0_zeros.bin`.

- Variable Name: **path_input_j0**
- Default Value: **./intput/J0_zeros.bin**
- Default Description: **(default) (std::string) Path to J0_zeros.bin**

6.1.20 Real Spectral Amplitude Path (path_A_w_R)

The output path of the real component of the spectral amplitude, `A_w_R`.

- Variable Name: **path_A_w_R**
- Default Value: **./output/A_w_R.bin**
- Default Description: **(default) (std::string) Path of A_w_R**

6.1.21 Imaginary Spectral Amplitude Path (path_A_w_I)

The output path of the imaginary component of the spectral amplitude, `A_w_I`.

- Variable Name: **path_A_w_I**
- Default Value: **./output/A_w_I.bin**
- Default Description: **(default) (std::string) Path of A_w_I**

6.1.22 path_w_active

The output path of `w_active`.

- Variable Name: **path_w_active**
- Default Value: **./output/w_active.bin**
- Default Description: **(default) (std::string) Path of w_active**

6.1.23 Config File Path (path_config_file)

The input path of the config file.

- Variable Name: **path_config_file**
- Default Value: **./config.txt**
- Default Description: **(default) (std::string) config.txt path**

6.1.24 Config Log Path (path_config_log)

The output path of the config log file.

- Variable Name: **path_config_log**
- Default Value: **./output/config_log.txt**
- Default Description: **(default) (std::string) config_log.txt path**

6.2 Output

Three data files are outputted at the end of a simulation. By default these take names of the form

- `A_w_R.bin`
- `A_w_I.bin`
- `w_active.bin`

These files are written in a binary format.

A log file is also generated which takes the default name

- `config_log.txt`

This file is a log of all the input parameters and settings used in the simulation that generated the other files and can be used as an easy reference to see which parameters were used for a simulation run. The file is written as in a plain text format. The input parameters and settings are outputted in this file in the same format used to read them in so that UPPE can be run again, pointing to a particular `config_log.txt` file as the input file, in case it is desired to rerun UPPE with that particular configuration.

All files outputted can have a prepending or postpending of a three digit number, in the form `XYZ`, i.e. the file `A_w_R.bin` can have a prepending or postpending

- `XYZ_A_w_R.bin`
- `A_w_R.bin_XYZ.bin`

respectively, where `XYZ` is a three digit number. The default behaviour is to have a `prepend`, but a `postend` can be set in the config file, as can `false` to switch off pending. The three digit number `XYZ` is an automatic pending of the filename that is done so that the data from a previous run of UPPE is not overwritten. For prepending the digits are inserted at the start of the filename, but after the path to the file. For postpending the digits are inserted at the end of the filename but before the start of the file extension. They start at `000` and increment by one every time a file is found to exist with the current number. All the files of a run are outputted with a particular number `XYZ` so that they are grouped together.

If this number `XYZ` increments past `999` then the `X` of that number continues to increment on its own as if it was its own integer, i.e. it would become `10`, then `11`, then `12`, etc.

6.3 Scripts

A python script and matlab script are present in the main directory. These scripts read in the data and plot it on various graphs. Reading these scripts should be able to give an idea of how the binary files are formatted.

Currently UPPE prints to the standard output how many rows and columns are present in the outputted binary data files, and these numbers are then inserted manually into the scripts.

7 Technical Overview of UPPE

Apart from `main.cpp` and `version.hpp`, each set of source files (the `.cpp` and associated `.hpp` file) each contains a single class. An overview is given for each set of files, with a brief description of what each one does, as well as a brief overview of the input arguments needed when initialising each class, and the public data and functions.

7.1 capillary_fibre

"capillary_fibre" describes the dimensions and dispersion properties of a dielectric capillary type fibre. To initialise it various variables are needed to be passed into the constructor, as described below.

7.1.1 Class Constructor Input

- `double Z_`
  The length of the capillary fibre.
- `grid_rkr& rkr_`
  The radial grid, passed in by reference.
- `grid_tw& tw_`
  The temporal grid, passed in by reference.
- `physics_textbook& physics_`
  The container for the physical constants, passed in by reference.
- `maths_textbook& maths_`
  The container for the mathematical constants and functions, passed in by reference.

7.1.2 Public Data

- `ArrayXXcd gamma`

- `double Z`
  The length of the capillary fibre, takes the same value as the value of Z_ from the constructor input.
- `double R`
  The same value of R as that for R in grid_rkr.
- `double n_glass`
  The refractive index of glass.

7.2 config_settings

"config_settings" reads in the input parameters and settings from a config and sets the corresponding variables to these read in values.

7.2.1 Public Functions

- `void read_in(std::string path)`
  Reads in the input parameters and settings found in the file located at `path`.
- `void check_paths()`
  Checks that the output paths already exist. If they do and the pending switch is on then the incrementing number is added on to the file name and incremented until an output path is found that does not yet exist.
- `void print()`
  Prints the input parameters and settings names, values, and descriptions to the standard output.
- `void print(std::string path_)`
  Prints the input parameters and settings names, values, and descriptions to the file specified in `path_`.
- `int n_z()`
  Returns the value of `n_z`.
- `void n_z_set(int value)`
- `std::string n_z_description()`

- `void n_z_description_set(std::string description)`
- `int n_r()`
  Returns the value of `n_r`.
- `void n_r_set(int value)`
- `std::string n_r_description()`
- `void n_r_description_set(std::string description)`
- `int n_m()`
  Returns the value of `n_m`.
- `void n_m_set(int value)`
- `std::string n_m_description()`
- `void n_m_description_set(std::string description)`
- `int n_t()`
  Returns the value of `n_t`.
- `void n_t_set(int value)`
- `std::string n_t_description()`
- `void n_t_description_set(std::string description)`
- `double T()`
  Returns the value of `T`.
- `void T_set(double value)`
- `std::string T_description()`
- `void T_description_set(std::string description)`
- `double w_active_min()`
  Returns the value of `w_active_min`.
- `void w_active_min_set(double value)`
- `std::string w_active_min_description()`
- `void w_active_min_description_set(std::string description)`

- `double w_active_max()`
  Returns the value of `w_active_max`.
- `void w_active_max_set(double value)`
- `std::string w_active_max_description()`
- `void w_active_max_description_set(std::string description)`
- `double Z()`
  Returns the value of `Z`.
- `void Z_set(double value)`
- `std::string Z_description()`
- `void Z_description_set(std::string description)`
- `double R()`
  Returns the value of `R`.
- `void R_set(double value)`
- `std::string R_description()`
- `void R_description_set(std::string description)`
- `double press()`
  Returns the value of `press`.
- `void press_set(double value)`
- `std::string press_description()`
- `void press_description_set(std::string description)`
- `double p_av()`
  Returns the value of `p_av`.
- `void p_av_set(double value)`
- `std::string p_av_description()`
- `void p_av_description_set(std::string description)`
- `double rep()`
  Returns the value of `rep`.

- `void rep_set(double value)`
- `std::string rep_description()`
- `void rep_description_set(std::string description)`
- `double fwhm()`
  Returns the value of `fwhm`.
- `void fwhm_set(double value)`
- `std::string fwhm_description()`
- `void fwhm_description_set(std::string description)`
- `double l_0()`
  Returns the value of `l_0`.
- `void l_0_set(double value)`
- `std::string l_0_description()`
- `void l_0_description_set(std::string description)`
- `double ceo()`
  Returns the value of `ceo`.
- `void ceo_set(double value)`
- `std::string ceo_description()`
- `void ceo_description_set(std::string description)`
- `double waist()`
  Returns the value of `waist`.
- `void waist_set(double value)`
- `std::string waist_description()`
- `void waist_description_set(std::string description)`
- `std::string pend_path()`
  Returns the value of `pend_path`.
- `void pend_path_set(std::string value)`
- `std::string pend_path_description()`

- `void pend_path_description_set(std::string description)`
- `std::string path_input_j0()`
  Returns the path to `input_j0`.
- `void path_input_j0_set(std::string value)`
- `std::string path_input_j0_description()`
- `void path_input_j0_description_set(std::string description)`
- `std::string path_A_w_R()`
  Returns the output path of `A_w_R`.
- `void path_A_w_R_set(std::string value)`
- `std::string path_A_w_R_description()`
- `void path_A_w_R_description_set(std::string description)`
- `std::string path_A_w_I()`
  Returns the output path of `A_w_I`.
- `void path_A_w_I_set(std::string value)`
- `std::string path_A_w_I_description()`
- `void path_A_w_I_description_set(std::string description)`
- `std::string path_w_active()`
  Returns the output path of `w_active`.
- `void path_w_active_set(std::string value)`
- `std::string path_w_active_description()`
- `void path_w_active_description_set(std::string description)`
- `std::string path_config_file()`
  Returns the path to `config_file`.
- `void path_config_file_set(std::string value)`
- `std::string path_config_file_description()`
- `void path_config_file_description_set(std::string description)`

- `std::string path_config_log()`
  Returns the output path of `config_log`.
- `void path_config_log_set(std::string value)`
- `std::string path_config_log_description()`
- `void path_config_log_description_set(std::string description)`

7.2.2 Private Functions

- `void set_variable(std::string& variable_name, std::string& variable_value_str, std::string& input_description_char)`
  Sets the variable `variable_name` to the value of `variable_value_sr` and sets the variable description to `input_description_char`. Each variable has an expected type and when a variable value is set the variable value in string form is converted to that expected type. Currently there are no checks that `variable_value_str` can be converted to the correct type and therefore trying to pass in the wrong type could cause undefined behaviour.
- `std::string set_path(std::string path, std::string pending_string)`
  Passes the variable path and pending string on to the relevant pending function, depending on the value of the pend switch, i.e. `set_pre_path` if the pending switch has the value `prepend` or `set_post_path` if the pending switch has the value `postpend`. If the pending switch has the value `false` then the file name remains unchanged.
- `std::string set_pre_path(std::string pre_path, std::string path)`
  Prepends the file name with the incrementing number `XYZ_`, where `XYZ` is inserted before the file name but after the path to the file's directory.
- `std::string set_post_path(std::string path, std::string post_path)`
  Appends the file name with the incrementing number `_XYZ`, where `XYZ` is inserted after the file name but before the file extension, if the extension exists.

7.2.3 Private Data

- `enum class SN{ ... }`
  A scoped enumeration containing a list of the variables that can be set through use of the config file. It ends with the value `LAST_SN_ENTRY` for ease of finding out the length of the enumeration.
- `static const char * setting_name[]`
  An array of character arrays of the names of the variables that can be set through use of the config file. These names have to match the order of those in the scoped enumeration `SN` to allow for correct use of switches in `set_variable`.

7.3 DHT

"DHT" evaluates the forward and backward discrete Hankel transform. Based on Fisk, Computer Physics Communications, 43 (1987).

An instance of DHT can be initialised with or without class constructor arguments, though as the transformation matrix is determined when an instance of DHT is initialised with constructor arugments, it is not recommended to initialise an instance of DHT without constructor arguments as this will cause undefined behaviour when doing Hankel transforms.

7.3.1 Class Constructor

To initialise an instance of DHT with constructor arguments the following parameters are needed to be passed in:

- `int n_r_`
- `maths_textbook& maths_`
  The container for the mathematical constants and functions, passed in by reference.

When an instance of DHT is initialised with constructor arguments, a Hankel transformation matrix is generated, which is used within the `forward` and `backward` functions.

7.3.2 Public Functions

DHT contains two public functions, a foward and a backwards Hankel transform.

- `ArrayXcd forward(ArrayXcd f_r_)`
  Returns the forward Hankel transform as type `ArrayXcd`.
- `ArrayXcd backward(ArrayXcd f_kr_)`
  Returns the backwards Hankel transform as type `ArrayXcd`.

7.3.3 Private Data

- `MatrixXcd H`

7.4 grid_rkr

"grid_rkr" is a non-uniform radial grid. The spectral counterpart of this grid is evaluated and accessible.

An instance of grid_rkr can be initialised with or without class constructor arguments.

An instance of grid_rkr can be initialised with or without class constructor arguments, though as the non-uniform radial grids are generated only in the parametised constructor, it is not recommended to initialise an instance of grid_rkr without constructor arguments as this will cause undefined behaviour when the radial grids are used.

7.4.1 Class Constructor

To initialise an instance of grid_rkr with constructor arguments the following parameters are needed to be passed in:

- `int n_r_`
- `double R_`
  The radius of the capillary.

- `int n_m_`
  The number of modes.
- `maths_textbook& maths_`
  The container for the mathematical constants and functions, passed in by reference.

When an instance of grid_rkr is initialised with constructor arguments, the non-uniform radial grids `r` and `kr` are generated.

7.4.2 Public Data

grid_rkr contains five pieces of public data,

- `ArrayXd r`
- `ArrayXd kr`
- `int n_r`
- `double R`
  The radius of the capillary.
- `int n_m`
  The number of modes.

7.5 grid_tw

"grid_tw" is a linear temporal grid. The spectral counterpart of this grid is evaluated and made accessible.

An instance of grid_tw can be initialised with or without class constructor arguments.

An instance of grid_tw can be initialised with or without class constructor arguments, though as the temporal grids are generated only in the parametised constructor, it is not recommended to initialise an instance of grid_tw without constructor arguments as this will cause undefined behaviour when the temporal grids are used.

7.5.1 Class Constructor Input

To initialise an instance of grid_tw with constructor arguments the following parameters are needed to be passed in:

- `int n_t_`
- `double T_`
- `double w_active_min_`
  The minimum positive angular frequency.
- `double w_active_max_`
  The maximum positive angular frequency.
- `maths_textbook& maths_`
  The container for the mathematical constants and functions, passed in by reference.

7.5.2 Public Data

grid_tw contains five pieces of public data,

- `ArrayXd t`
- `ArrayXd w_active`
- `int n_t`
- `int n_active`
- `int w_active_min_index`

7.6 IO

"IO" objects enable reading/writing of binary files to/from eigen arrays.

7.6.1 Public Functions

IO contains four public functions,

- `Array<unsigned short, Dynamic, Dynamic> read_uint16(const char* path_, int N_row_, int N_col_)`
  Reads in a binary file of unsigned 16 bit ints to a dynaamic two dimensional Eigen array of shorts. The number of rows and columns of ints in the file need to be known before hand.

- `ArrayXXi read_int(const char* path_, int N_row_, int N_col_)`
  Reads in a binary file of ints to a dynamic two dimensional Eigen array of ints. The number of rows and columns of ints in the file need to be known before hand.

- `ArrayXXd read_double(const std::string path_, int N_row_, int N_col_)`
  Reads in a binary file of doubles to a dynamic two dimensional Eigen array of doubles. The number of rows and columns of doubles in the file need to be known before hand.

- `void write_double(const std::string path_, ArrayXXd input_, int N_row_, int N_col_)`
  Writes a dynamic two dimensional Eigen array of doubles to a file in a binary format. The number of rows and columns of doubles in the Eigen array need to be known before hand.

7.7 keldysh_gas

"keldysh_gas" contains the medium response model, which is dependent on the gas being used.

7.7.1 Class Constructor Input

To initialise an instance of keldysh_gas the following parameters are needed to be passed in:

- `double press_`
  The pressure of the gas in the capillary.

- `grid_tw& tw_`
  The temporal grid, passed in by reference.

- `DFTI_DESCRIPTOR_HANDLE& ft_`

- `maths_textbook& maths_`
  The container for the mathematical constants and functions, passed in by reference.

7.7.2 Public Data

keldysh_gas contains five pieces of public data,

- `double atom_density`
- `double U`
- `double C_kl`
- `double n_star`
- `double kappa`

7.7.3 Private Data

- `physics_textbook physics`
  The container for the physical constants.
- `maths_textbook maths`
  The container for the mathematical constants and functions.
- `grid_tw tw`
  The temporal grid.
- `DFTI_DESCRIPTOR_HANDLE ft`

7.7.4 Public Function

keldysh_gas contains four public functions,

- `ArrayXcd nl_polarization(ArrayXd E_t_)`
  Evaluates the nonlinear polarisation for the active frequencies.
- `ArrayXd ionization_rate(ArrayXd E_t_)`
  Calculates the ionisation rate of the gas.

- `ArrayXd electron_density(ArrayXd W_t_)`
  Calculates the electorn density in the gas.
- `ArrayXcd current_density(ArrayXd E_t_)`
  Evaluates the current density for the active frequencies.

7.8 laser_pulse

"laser_pulse" contains the active spectral amplitudes and governs their propagation over longitudinal step dz.

7.8.1 Class Constructor Input

To initialise an instance of laser_pulse the following parameters are needed to be passed in:

- `double p_av_`
- `double rep_`
- `double fwhm_`
  The full width at half maximum of the laser pulse
- `double l_0_`
  The central wavelength of the laser pulse.
- `double ceo_`
- `double waist_`
- `grid_tw& tw_`
  The temporal grid, passed in by reference.
- `grid_rkr& rkr_`
  The radial grid, passed in by reference.
- `DFTI_DESCRIPTOR_HANDLE& ft_`
- `DHT& ht_`
  The container for the discrete Hankel transform.

- `maths_textbook& maths_`
  The container for the mathematical constants and functions, passed in by reference.

7.8.2 Public Data

- `ArrayXXcd A_w_active`

7.8.3 Private Data

- `physics_textbook physics`
  The container for the physical constants.
- `maths_textbook maths`
  The container for the mathematical constants and functions.
- `grid_tw tw`
  The temporal grid.
- `grid_rkr rkr`
  The radial grid.
- `DFTI_DESCRIPTOR_HANDLE ft`
- `DHT ht`
  The container for the Hankel transform functions.
- `ArrayXXcd Y_4`
- `ArrayXXcd Y_5`
- `double e`

7.8.4 Public Functions

- `void propagate(double dz_, capillary_fibre& capillary_, keldysh_gas& gas_)`
  Propagates the spectral amplitudes of the pulse over a step `dz` along the `Z` axis. This is based on A. Couairon, et al., Eur. Phys. J. Special Topics, 199, 5 (2011).

7.8.5 Private Functions

- `void RK_F_45(double h_, capillary_fibre& capillary_, keldysh_gas& gas_)`
- `ArrayXXcd RHS_UPPE(double dz_, ArrayXXcd temp_1, capillary_fibre& capillary_, keldysh_gas& gas_)`

7.9 main

The structure of UPPE is as follows:

1. Command Line Arguments Passed to UPPE
2. MPI initialisation
3. Program Input
4. Constructors
5. Propagation
6. Output
7. Clean Up

Each part is discussed in more detail below.

7.9.1 Command Line Arguments Passed to UPPE

Currently there is only one flag that UPPE recognises, this is the "`-cf`" flag which sets the next input argument passed in to UPPE as the path and name of the config file.

7.9.2 MPI initialisation

Initialises MPI.

7.9.3 Program Input

By default, all the input parameters and settings are set to default values are defined in the source code of UPPE.

A default config file path and name is used if no others are passed to UPPE, this is `./config.txt`. If this file is not found then default values that are set within the source code of UPPE are used. The values found in this config file are set as the values of the corresponding input parameters and settings.

The path and name of a config file can be passed to UPPE through a commandline argument and the input parameters and settings found in this file are set to the corresponding input parameters and settings within UPPE. If this config file is not found then the default values will be used.

A config file does not need to contain every input parameter and setting as the default values are set before the values in a config file are read in, so the input parameters that are not present in the config file will just take the default values.

7.9.4 Constructors

The constructors are subdivided into three sections, general, grids, and physical, each subdivision contains a number of contructors.

General contains the constructors for maths_textbook, physics_textbook, Dfti, and DHT. maths_textbook contains the mathematical constants and formulas needed in UPPE. physics_textbook contains all the physical constants needed in UPPE, Dfti is used for discrete Fourier transforms, and DHT is used for discrete Hankel transforms.

Grids contains the constructors for the radial and temporal grids. grids_rkr is contains a non-uniform radial grid and grids_tw contains a linear temporal grid.

Physical contains laser_pulse, capillary_fibre, and keldysh_gas. laser_pulse contains the functions needed to propagate the laser pulse in the gas filled capillary, as well as the function that propgates it one step in the Z direction.

capillary_fibre describes the gas-filled capillary fibre in question. keldysh_gas contains the medium response model.

7.9.5 Propagation

Main loop of the program. Propagates the laser pulse through the gas filled capillary along the Z axis.

7.9.6 Output

Writes the data generated in a run to the output files.

7.9.7 Clean Up

Cleans up and frees the Dfti descriptor as well as MPI.

7.10 maths_textbook

"maths_textbook" is a container for mathematical constants and functions.

7.10.1 Class Constructor Input

To initialise an instance of maths_textbook the following parameters are needed to be passed in:

- `std::string path_input_j0_`
  The path to `J0_zeros.bin`.

7.10.2 Public Data

- `double pi`
  The value of pi, as calculated by the inverse cosine of -1.0.
- `ArrayXd J0_zeros`

7.10.3 Private Data

- `std::string path_input_j0`
  The path to `J0_zeros.bin`

7.10.4 Public Functions

- `double trapz(ArrayXd x_, ArrayXd y_)`
  Performs the trapezoidal integration of the array `y_`, over the range of `x_.tail(n_ints)` and `x_.head(n_ints)`, where `n_ints` is the number of intervals and is equal to `x_.rows()- 1`.
- `ArrayXd cumtrapz(ArrayXd x_, ArrayXd y_)`
  Performs the cumulative trapezoidal integration of the array `y_`, over the range of `x_.tail(n_ints)` and `x_.head(n_ints)`, where `n_ints` is the number of intervals and is equal to `x_.rows()- 1`.

7.11 physics_textbook

"physics_textbook" is a container for physical constants.

To initialise an instance of physics_textbook no constructor arguments are needed.

7.11.1 Public Data

- `double E_at`
- `double l_at`
- `double m_at`
  The mass of the electron.
- `double q_at`
  The elementary charge.
- `double t_at`
- `double w_at`

- `double c`
  The speed of light.
- `double eps_0`
  The permittivity of free space.
- `double mu_0`
  The permeability of free space.
- `double h_bar`
  The Planck constant.
- `double k_B`
  The Boltzmann constant.

7.12 version

Version stores the version number of UPPE that is used by the CMake-Lists.txt to append it to the name of the generated executable.

8 Conclusions