TDSE Code - Developers Notes

J. Venzke

January 6, 2017

1 Introduction

The TDSE code discussed here was developed by Joel Venzke while working on his Ph.D. in the Ultrafast Theory Group in JILA. It solves the Time Dependent Schrodinger Equation (TDSE) for various targets in a laser field with two active electrons.

2 Build System

Currently using autoconfig, but that may change. See "build.${SYSTEM}" files for building examples on various systems.

3 Classes

The classes used in this code are written so they can be tested independently. Therefore, all classes that depend on other classes require those classes to be passed into the constructor. The developer is responsible to not delete these classes until other dependent classes are deleted.

3.1 Parameters

This class holds the input file information and will be used by all other classes. It depends on a json parser and reads inputs in json format. For more information about input parameter definitions and how they are used, see Section 5. This class is mainly used as a data structure. That being said, it does provided tools to validate various input parameters to limit user error.

3.2 HDF5Wappers

IO for this code is done using HDF5 files. HDF5 files provide the ability to utilize parallel file systems, generate self describing data files, and write compressed data. With these advantages, it becomes slightly more difficult to write data. To alleviate this issues, this class provides tools for writing to HDF5 files in a straight forward way.

Writing multidimensional arrays is difficult in HDF5. To avoid the added headache, one denominational arrays with the access pattern of

$$array[x + y * nx] = array[x][y] \tag{1}$$

and

$$array[x + y * nx + z * nx * ny] = array[x][y][z] \tag{2}$$

and so on.

3.3 Pulse

Pulse handles creating and storing various parts of the pulse. Much of the functionality is private rather than public to prevent accessing data that is not allocated. Since storing each individual pulse is only useful until it is printed to a file, it is deallocated inside the constructor. The result is a much more efficient use of memory, and protection from accessing garbage arrays. See Section 5 on how to define pulses.

4 Targets

These are the various targets that this code supports. Each one will only be supported for certain solvers, so please double check which ones are implemented. Validation will eventually be added to the Parameters class

4.1 He

A two active electron target of Helium. Currently being developed using finite differences.

5 Input

All input values are in atomic units. The input file name should be "input.json"

5.1 Parameters

The following is the base set of parameters. Those with sub parameters will have additional sub sections. An example input file can be found in Section 5.4.

- delta_t
  - The size of the time step in atomic units.
- dimensions
  - A list containing information about each dimension see Section 5.2 for more details.
- restart
  - When set to 1, it will restart the simulation from the last checkpoint in the hdf5 file. It also checks to see if the input files match the original one. Set to 0 if you want a new simulation.
- target
  - Tells the code what target the laser will be incident on. See Sections 4 for added information and the list of supported targets.
- gobbler
  - The percent of the grid that is "gobbled" by the absorbing potential so the wavefunction does not reflect off the edge of the box.
- pulses
  - A list of pulses used as a super position to create the laser field. See Section 5.3 for more information.

5.2 Dimensions

Each dimension requires the following two input parameters.

- dim_size
  - The length of that dimension in atomic units.
- delta_x
  - The step sizes in that dimension in atomic units.

5.3 Pulses

Each pulse requires the following input parameters.

- pulse_shape
  - The shape of the pulse envelope. Currently supports sin^2 envelopes.
- cycles_on
  - The number of cycles the pulse uses to turn on.
- cycles_plateau
  - The number of cycles the pulse is at max amplitude. Usually 0.0
- cycles_off
  - The number of cycles the pulse uses to turn off.
- cycles_delay
  - The number of cycles before the pulse starts to turn on.
- cep
  - The carrying phase envelope of the pulse. It is defined at the time the pulse starts to turn on.
- energy
  - The fundamental angular frequency of the pulse. Corresponds to the energy of the photons in atomic units.
- e_max
  - The maximum amplitude of the pulse in atomic units.

5.4 Example File

Input files are in json format. Here is an example input file:

```
{
  "delta_t": 0.2,
  "dimensions": [
    {"dim_size": 1.0,
    "delta_x":   0.01},
    {"dim_size": 2.05,
    "delta_x":   0.5}],
  "restart": 0,
  "target":  "He",
```

```
  "gobbler": 0.9,
  "pulses": [
    {"pulse_shape":   "sin2",
      "cycles_on":      3.0,
      "cycles_plateau": 1.0,
      "cycles_off":     3.0,
      "cycles_delay":   0.0,
      "cep":            0.0,
      "energy":         5.338e-3,
      "e_max":          1.0},
    {"pulse_shape":   "sin2",
      "cycles_on":      3.0,
      "cycles_plateau": 0.0,
      "cycles_off":     3.0,
      "cycles_delay":   1.0,
      "cep":            0.0,
      "energy":         5.338e-3,
      "e_max":          1.0}]
}
```

6 Testing

The plan is to use Travis CI, but no current work has been done to this extent.