Maximilian Bechtel

User Manual – Project 5: Simulation of Quantum Dynamics

1 Introduction

This user manual should give a short insight and explanation of the project *Simulation of Quantum Dynamics (1 particle, 1-dimensional)*. To this reason, the important functions and subroutines of the Python and Fortran source code are explained in 5 Description of the source code. Additionally, the keywords for input files are given in 4 Description of input files. To drive everything, the main program was written in the Python programming language. However, since the implementation of computational-intensive parts are too slow using Python, these were written in the Fortran programming language and compiled with the *f2py* compiler tool of Python´s *numpy* module.

2 Installation

The source code of the program and sample input files can be found on a Github repository. Simply clone the corresponding folder by using ssh or https:

```
git clone git@github.com:CodeSeneca/QuantumDynamics.git
git clone https://github.com/CodeSeneca/QuantumDynamics.git
```

In the following it is assumed that a Linux work station is used. On Windows or MacOS systems the commands can slightly differ. To compile the source code of the Fortran part, proceed in the following way:

```
cd QuantumDynamics
cd pkg
f2py -m les -c les.f90
```

A file with extension *.pyd will be written out.

Adjust the python interpreter in the very thirst line of quantum_dynamics.py and animate.py. The location of the interpreter can vary depending on the Linux distribution. For instance, at Linux Ubuntu it is found at: */usr/bin/python3*.

For this the Shebang line looks like this:

#! /usr/bin/python3

3 Usage

The project consists of two main programs: **quantum_dynamics.py** and **animate.py**.

These are used in the following way:

./quantum_dynamics.py [input file]

./animate.py [delay between each frame in the animation in ms]

(each value between 20-100) was seen as suitable

quantum_dynamics.py is the computational part. This program calculates the propagation of the chosen wave function by solving the time dependent Schrödinger equation numerically by the *Crank-Nicolson algorithm*. Moreover, expectation values are calculated for each time step.

animate.py is the script that can be used to animate the time dependent behavior of the wave function. By using it, a *matplotlib* figure like the following should appear:

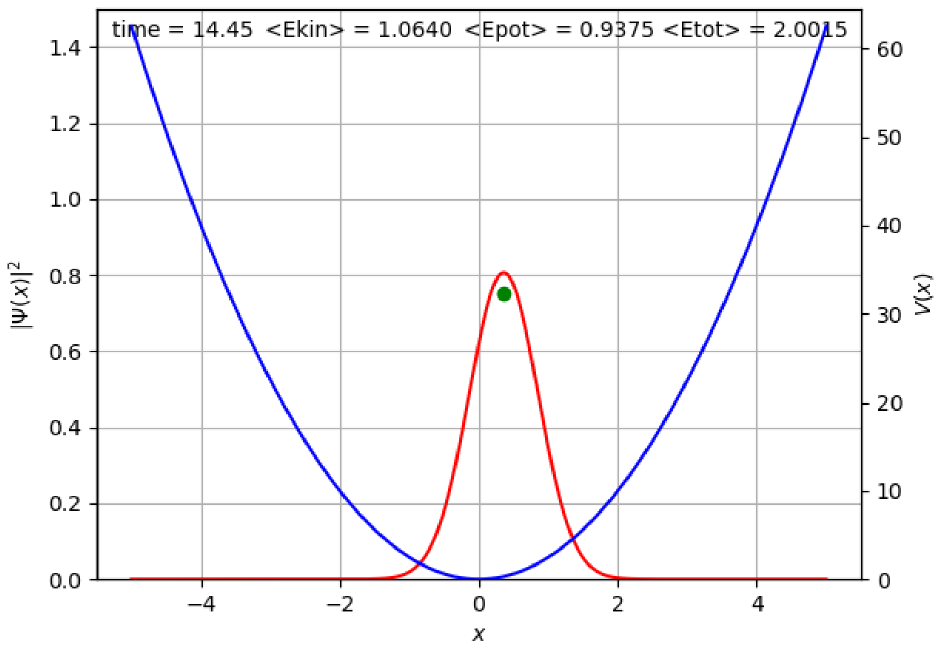

Figure 1: Animation window produced by the script animate.py by using matplotlib.

Additionally, three different output files are written out:

potential.dat	**Values of the chosen potential (harmonic, Morse, …)**
plot.dat	**Values of the wave function for each time step**
energy.dat	**Expectation values for each time step**

Structure of the file energy.dat:

Time step Norm of the wavefunction E_{pot} E_{kin} E_{tot} <p> <x>

with <…> indicating quantum mechanical average/expectation values of the wavefunction

To work, animate.py needs to be executed in the same folder where these three output files are located.

4 Description of input files

Sample input files can be found in the folder *input_samples*. Each keyword in the input has to be in the following format:

keyword = value

Reading in the input file is case sensitive, everything needs to be in lowercase letters! The program only reads in such keywords. Everything else is simply ignored. So, any comment can be put at every location in the input file. Atomic units are used.

Possible keywords are:

dt	**time step**
nsteps	**number of time steps**
dx	**grid spacing/distance between two grid points**
ngridpoints	**total number of grid points**
x0	**start location of wavefunction**
p0	**start momentum wave function**
sigma	**standard deviation for Gaussian wave function**

k	**force constant for a harmonic potential**
alpha	**stiffness for a Morse potential**
box	**height of a potential barrier/box**
mass	**particle mass**
potential	**1 = particle in the box, 2 = harmonic, 3 = Morse, 4 = potential wall**
wavefunction	**1 = gaussian wave packet, 2 = sinus (eigenfunction of the particle in a box)**
output_mode	**extent of output written 0 = no output files are written, 1 = potential.dat, energy.dat, plot.dat are written**
output_step	**write out energy for only each nth time step**

If no value is given, the following default values will be used:

dt = 0.005

nsteps = 100

dx = 0.01

ngridpoints = 1000

x0 = 0.0

p0 = 1.0

sigma = 1.0

k = 5.0

alpha = 0.5

box = 1000

mass = 1.0

potential = 2

wavefunction = 1

output_mode = 1

output_step = 10

The program can be used in two ways: By giving a *positive, real time step* dt, the dynamic behavior of the chosen wavefunction is simulated. By giving a *negative, imaginary time step*, the program will find the correct eigenfunction for the given potential by reverting the Schrödinger equation. The format for imaginary values is:

dt = -0.005j

using j as imaginary unit and not i!

5 Description of the source code

5.1 Quantum_dynamics.py

The main program to drive everything is found at quantum_dynamics.py:

```
Import all needed modules (time, sys, numpy, pkg.IO, pkg.functions, pkg.les)
Read in command line argument = input file name
Set the needed simulation parameters
Write settings to console
Create and open output files for writing
Initializations for t = 0 s
    Create simulation grid array
    Create potential array
    Create wavefunction array
    Normalize start wavefunction
Write output for t = 0 s
Create initial d vector for Thomas algorithm
MAIN LOOP
    STEP 1: With current wavefunction calculate new vector d and overwritten
            vector b
    STEP 2: Solve LES with Thomas algorithm -> new wavefunction
    STEP 3: Calculate expectation values and write output for each nth step
Close output files and end program properly
```

5.2 Animate.py

Animate.py is used to read in the outputfiles energy.dat, plot.dat, potential.dat and to animate everything by using matplotlib:

```
Import all needed modules (sys, time, numpy, matplotlib.pyplot, matplotlib.animation.FuncAnimation)
Read in command line argument = frame delay
Create figure and axis objects
Create objects changing in each time step
        psi = wavefunction
        loc = green dot
        time_text, ekin_text, epot_text, etot_text
Read in expectation values from energy.dat
Read in potential from potential.dat
Read in wavefunction from plot.dat
    ➔ Used function: genfromtxt() is the fastest in this case
Extract the needed data from read in files by array slicing (time_steps, norm, epot, ekin, etot, x_loc = expectation value of x for particle)
Determine time step dt from array time_steps by substraction of two entries following each other
Extract x values and y values for potential
Update function used by FuncAnimation
        Set up new wavefunction for each frame (next row in array wavefunction_dat)
        Update time_text, ekin, epot, etot, wavefunction to be plotted
        Set green dot (expectation value of x) in the middle of the y axis
Axis settings for plotting
        Set maximum of shown y value to last element of potential array + 2.0 buffer
Show plot
```

5.3 Pkg.IO

Module for reading input and writing output

read_input(filename:str) -> list:

Read in input parameters from input file

Set default parameters

write_output(step, plot_file, output_file, psi, x_values, dx, dt, epot, ekin, etot, p, x):

Write expectation values for i^{th} time step to output files

5.4 Pkg.functions

Module for defining all needed mathematical functions

harmonic_potential(x: float, k: float, x0:float) -> float:

Harmonic potential

morse_potential(x: float, alpha:float, x0:float) -> float:

Morse potential

gaussian(x:float, x0:float, sigma:float, p0:float) -> complex:

Gaussian wave packet

5.5 Pkg.simulation

Module to calculate all expectation values and to solve the LES by using the Thomas algorithm

The implementation was too slow. Therefore pkg.simulation is not used anymore. Instead, the much faster subroutines of pkg.les written in Fortran are used in quantum_dynamics.py.

5.6 Pkg.les

Module to calculate all expectation values and to solve the LES by using the Thomas algorithm

solve_les(b, d, psi, n)

Subroutine to solve the LES by using the Thomas algorithm. A detailed description of the algorithm can be found at: *https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm*. In each time step the vectors b and d are overwritten. Therefore, they are calculated in each time step again. An algorithm where b and d are not replaced is possible. However, the necessary bookkeeping is much more time consuming the simply overwriting the two vectors. The bookkeeping algorithm can be found in pkg.simulation/solve_les_thomas() written in Python.

calc_b(V, n, dx, dt, mass, b)

Calculate main diagonal of LES matrix

calc_d(V, psi, dt, dx, mass, n, d)

Calculate right hand side vector of LES

calc_norm(psi, n, dx, norm_real)

Calculate norm of wavefunction

calc_epot(psi, n, V, dx, epot_real)

Calculate $<E_{pot}>$

calc_ekin(psi, n, dx, m, ekin)

Calculate $<E_{kin}>$

calc_p(psi, n, p_real)

Calculate expectation value of particle momentum p

calc_x(psi, x_values, n, dx, x_real)

Calculate expectation value of particle location x