ASSIGNMENT 4 – Quantum Information And Computing

Carlo Divari
Dipartimento di Fisica e Astronomia, Università di Padova, Via Marzolo 8, 35131, Padova, Italy

(Dated: November 28, 2023)

Introduction

This report presents numerical computations and analyses related to the solution of the independent Shrodinger equation in 1D. The main focus is to implement the finite difference method with second order approximations or higher to compute a second order differential equations. Once obtained the results, they will be compared with the theoretical results given by the analytical solutions. For simplicity throughout the report all the physical constants are set equal to 1, and the hamiltonian has been rescaled by a factor 2.

Correctness and Stability

In solving the Schrödinger equation, a second-order differential equation central to quantum mechanics, I opted for the central finite difference method. This choice was deliberate, considering its increased accuracy by a factor of 'n' in comparison to both the forward and backward alternatives. The central finite difference method offers a balanced approach, exhibiting superior precision in approximating derivatives and capturing the nuances of the Schrödinger equation more effectively. This decision reflects a commitment to computational accuracy in simulating quantum phenomena, ensuring that the numerical solution aligns closely with the theoretical expectations inherent in the Schrödinger equation. In order to test the performance of the code we computed the first three energy levels and we iterated these procedure 100 times to evaluate the variance of each eigenvalue.

Energy Level (k)	std	Mean	Theoretical reult
k=0	10^{-13}	0.9999(7)	1.0
k=1	10^{-13}	2.999(8)	3.0
k=2	10^{-13}	4.999(6)	5.0

TABLE I: Finite difference method 2^{nd} order approximation

Then we implemented the possibility to achieve higher order approximations. In particular we computed the 4^{th} order aproximation.

Energy Level (k)	std	Mean	Theoretical reult
k=0	10^{-13}	0.99999999(6)	1.0
k=1	10^{-13}	2.9999999(7)	3.0
k=2	10^{-13}	4.9999999(1)	5.0

TABLE II: Finite difference method 4^{th} order approximation

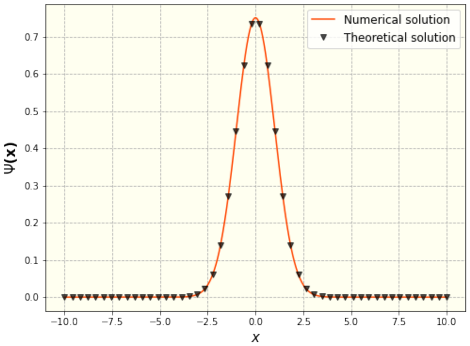

FIG. 1: Ground state

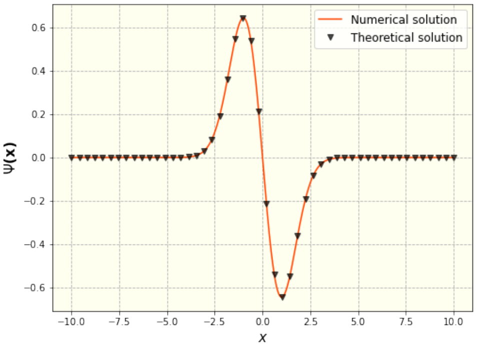

FIG. 2: E_1 state

In order to compare the analytical solutions and the eigenstates computed, the latter have been normalized respect to the gridspace interval [-10, 10].

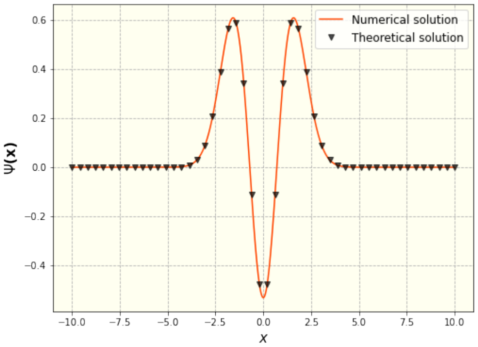

FIG. 3: E_3 state

Flexibility and Error Handling

The implemented code consists of a class comprising four distinct methods. The first method is responsible for initializing the matrix, setting the groundwork for subsequent computations. Following this, the second and third methods construct the kinetic and potential energy components, respectively, contributing to the overall functionality of the program. The final method is designed to compute the first k eigenvalues and eigenvectors of the matrix. The code lacks of the module to help debugging wrote in a prior assignment. In order to implemente higher-order approximations in the finite difference method, a flag has been incorporated. This flag, when appropriately set during initialization, enables the adjustment of the number of indexes of the nonzero elements, thereby enhancing the flexibility and versatility of the codebase. Overall the code is quite flexible but lacks of good routines to debug and a proper documentation in order to help solving duture problems and implementations.

Efficiency

The code is designed to save memory by using sparse matrices, which ignore storing all the zeros in the Hamiltonian, making computations more efficient. To find the smallest eigenvalues, the code utilizes ARPACK, a reliable method, although it's generally better at handling large values than small ones. To improve accuracy in finding smaller eigenvalues, a good idea is to consider the shift-invert method, which can be a smart tweak for better performance. This ensures that the code is both memory-friendly and effective in dealing with quantum mechanics computations.

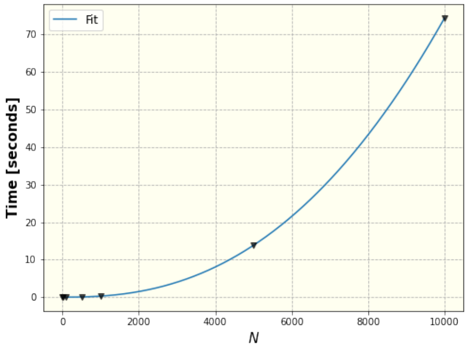

FIG. 4: x^α $\alpha = 2.42852(6)$. This is how the code scales with the size of the matrix.