*Mapana Journal of Sciences*
2021, Vol. 20, No. 1, 19-37
ISSN 0975-3303 | https://doi.org/10.12723/mjs.56.2

# On the Use of B-Splines as Ritz Variational Basis Functions to Solve the Schrodinger Equation (TISE) for a Free Quantum Particle

Mandyam N Anandaram*

## Abstract

B-Splines as piecewise adaptation of Bernstein polynomials (aka, B-polys) are widely used as Ritz variational basis functions in solving many problems in the fields of quantum mechanics and atomic physics. In this paper, these are used for solving the 1-D Time Independent Schrodinger Equation (TISE) for a free quantum particle subject to a fixed domain length by using the Python software SPLIPY with different sets of computation parameters. In every case, it was found that over 60 percent of energy levels had excellent accuracy, thereby proving that the use of B-spline collocation is a reliable method.

**Keywords**: B-Splines, Variational basis, TISE, free particle, Gauss Legendre quadrature, Collocation, Eigen solution.

## 1. Introduction

B-splines are versatile and convenient for use as basis functions of choice for solving many problems in atomic, molecular, and nuclear physics by applying the Ritz variational method. A good review of this subject as of 2001 is given in [1] with many references. Since then many advances have occurred in the last two decades helped by the availability of faster PCs and scientific computing software such as Python 3.7 used here. A shorter description of Bernstein polynomials (B-polys) and B-splines is also provided in [2] along with details of

* Professor of Physics (Retired), BangaloreUniversity, Bangalore, India; mnanandaram@gmail.com

solving a 4th order ordinary differential equation (ODE) specified as a boundary value problem (BVP).

In brief, the Bernstein basis polynomials (also known as B-polys) of any degree arise in the binomial expansion of unity which is the sum of two complementary parts (say, $\boldsymbol{x}$ and $\boldsymbol{1-x}$) raised to that degree. In the entire domain, the sum of all these Bernstein basis polynomials (also known as B-polys) is always unity at any ordinate within the domain. The solution curve can be drawn as a Bezier curve with undivided domains where each basis function is modified by a weighting factor determined by the nature of the problem and then added together at each point along the curve. B-splines are more versatile as they use the Carl deBoor (aka, Cox-deBoor) method to calculate and shift the Bernstein basis polynomials of specified degree and order for domains divided into many sub-intervals but can also give just one set of Bernstein basis functions of a chosen degree for use with undivided domains. Well formulated and freely available/usable Python based software such as **SPLIPY**[6] or **Bspline** [7] packages based on deBoor algorithm can be used to calculate and process all the spline basis function sets needed for the problem to be solved in the next section. Short introductory descriptions to the variational method as applied to TISE is given next, showing the expansion of a general eigenfunction in terms of basis spline functions and from it obtain expressions for needed expectation values as matrix elements.

**2.1 Consider the stationary Schrödinger equation (TISE) given by**

$$\boldsymbol{H\psi(x) = -\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x) = E\psi(x)}, \tag{2.1}$$

The eigenfunction $\boldsymbol{\psi(x)}$ can be expanded in terms of an infinite number of eigen states $\boldsymbol{\phi_j}$, by a summation over all states $\boldsymbol{j}$ where $\boldsymbol{C_j}$ is the weight coefficient and is given by

$$\boldsymbol{\psi \equiv \sum_j C_j \phi_j} \tag{2.2}$$

The Ritz variational method, which is also an approximation method, asserts that the expectation value of the Hamiltonian, $\boldsymbol{\hat{H}}$ computed with any normalized trial function, is always higher than or equal to the energy of the ground state. It can be formally stated

as $\langle\boldsymbol{\psi}|\widehat{\boldsymbol{H}}|\boldsymbol{\psi}\rangle \geq \boldsymbol{E_0}$ , where $\widehat{\boldsymbol{H}}\boldsymbol{\phi_j} = \boldsymbol{E_j}\boldsymbol{\phi_j}$. A short proof is provided below. If $\{\boldsymbol{\phi_j}\}$ is a basic set of **orthonormal eigenfunctions** of the Hamiltonian $\widehat{\boldsymbol{H}}$ then from the expansion $\boldsymbol{\psi} = \sum_j \boldsymbol{C_j}\boldsymbol{\phi_j}$, where $\boldsymbol{C_j}$ is the coefficient of the state $\boldsymbol{j}$, the expectation value is obtainable as,

$$\langle\boldsymbol{\psi}|\widehat{\boldsymbol{H}}|\boldsymbol{\psi}\rangle = \sum_j\sum_k \boldsymbol{C_k^*}\boldsymbol{C_j}\langle\boldsymbol{\phi_k}|\widehat{\boldsymbol{H}}|\boldsymbol{\phi_j}\rangle = \sum_j\sum_k \boldsymbol{C_k^*}\boldsymbol{C_j}\,\boldsymbol{E_j}\boldsymbol{\delta_{kj}}$$

$$= \sum_j\sum_k \boldsymbol{C_k^*}\boldsymbol{C_j}\,\mathbf{E_j} \;\geq\; \mathbf{E_0}\sum_j \boldsymbol{C_j^*}\boldsymbol{C_j} = \boldsymbol{E_0} \qquad (2.3)$$

Since $\sum_j \boldsymbol{C_j^*}\boldsymbol{C_j} = \mathbf{1}$. In the variational approach, one starts with an initial trial function $\psi$ defined by a set of expansion coefficients $\left\{\boldsymbol{C_j^{(0)}}\right\}$, aAnd finds the optimum solution of an arbitrary problem defined by the Hamiltonian $\widehat{H}$ by minimizing the expected value given by $\langle\boldsymbol{\psi}|\widehat{\boldsymbol{H}}|\boldsymbol{\psi}\rangle$ with respect to the expansion coefficients. Once the ground state $|\boldsymbol{\psi_0}\rangle$ is found, one can obtain the first excited state similarly by adding to the expectation value of energy, a penalty term proportional to the norm of the overlap between the ground and variational states, $\langle\boldsymbol{\psi}|\widehat{\boldsymbol{H}}|\boldsymbol{\psi}\rangle + \boldsymbol{\gamma}|\langle\boldsymbol{\psi_0}|\boldsymbol{\psi}\rangle|^{\mathbf{2}}$. Higher energy states $|\psi_n\rangle$ are similarly found by a minimization of the cost function $\langle\boldsymbol{\psi}|\widehat{\boldsymbol{H}}|\boldsymbol{\psi}\rangle + \boldsymbol{\gamma}\sum_{\boldsymbol{j=0}}^{\boldsymbol{n-1}}\left|\langle\boldsymbol{\psi_0}|\boldsymbol{\psi_j}\rangle\right|^{\mathbf{2}}$.

Here this method will make use of B-spline basis functions as the trial function set.

### 2.2 Using B-spline Basis Sets in TISE

A few important advantages of using basis spline function sets are stated here. Atomic eigenfunctions are smooth functions and so can be represented by piecewise basic polynomials. A finite number of B-spline sets can represent the effects of an infinite series of eigenfunctions. This statement means that a smaller number of basis splines suffice to calculate eigenenergies to the same or better accuracy than possible with a much larger number of eigenfunctions.

The energies and eigenfunctions of a quantum mechanical state $|\boldsymbol{\psi}\rangle$ can be found by solving the Schrodinger equation. (see Eqn. (2.1))

$$H|\boldsymbol{\psi}\rangle = \boldsymbol{E}|\boldsymbol{\psi}\rangle \qquad (2.4)$$

We postulate that the eigenfunction can be expanded similar to (2.2) in terms of spline basis functions of order $\boldsymbol{k}$ (degree, $\boldsymbol{p = k - 1}$) and denote $\boldsymbol{B_j \equiv B(k, j, x)}$. Then we can write,

$$|\boldsymbol{\psi}\rangle \equiv \sum_j \boldsymbol{C_j}\, |\boldsymbol{B_j}\rangle \tag{2.5}$$

Substituting (2.5) into (2.4) in order to determine the coefficient vector $\boldsymbol{C} = \{\boldsymbol{C_j}\}$ we get

$$\boldsymbol{H} \sum_j \boldsymbol{C_j}\, |\boldsymbol{B_j}\rangle \equiv \boldsymbol{E} \sum_j \boldsymbol{C_j}\, |\boldsymbol{B_j}\rangle \tag{2.6}$$

Now we multiply (2.6) from the left by $\sum_i |\boldsymbol{B_i}\rangle\langle\boldsymbol{B_i}|$ and here, we note that as the B-spline functions are normalized but not mutually orthogonal, there will exist a non-zero overlap integral which can be calculated later. This implies that

$$\sum_i |\boldsymbol{B_i}\rangle\langle\boldsymbol{B_i}| = \int \boldsymbol{B(k, i, x) B(k, j, x)}\, \boldsymbol{dx} > 0 \tag{2.7}$$

This is known as the overlap integral, which now has a non-zero positive value. Hence (2.6) may be rewritten as

$$\sum_i \sum_j \boldsymbol{C_j}\, |\boldsymbol{B_i}\rangle\langle\boldsymbol{B_i}|\, \boldsymbol{H}|\boldsymbol{B_j}\rangle = \boldsymbol{E} \sum_i \sum_j \boldsymbol{C_j}\, |\boldsymbol{B_i}\rangle\langle\boldsymbol{B_i}|\boldsymbol{B_j}\rangle \tag{2.8}$$

Since the $|\boldsymbol{B_i}\rangle$ are linearly independent, equation (2.8) can be satisfied only if

$$\sum_j \langle\boldsymbol{B_i}|\boldsymbol{H}|\boldsymbol{B_j}\rangle \boldsymbol{C_j} = \boldsymbol{E} \sum_j \langle\boldsymbol{B_i}|\boldsymbol{B_j}\rangle \boldsymbol{C_j} \text{ for each i} \tag{2.9}$$

This expression can be written in matrix form as the Eigen equation,

$$\boldsymbol{HC} = (\boldsymbol{T} + \boldsymbol{V})\boldsymbol{C} = \boldsymbol{ESC} \tag{2.10}$$

According to (2.10), the solution of TISE is required as input in the Hamiltonian matrix elements between the spline basis functions (aka, B-splines) and the overlap matrix elements, $\boldsymbol{S}$, between the B-splines themselves as they are not orthogonal but only linearly independent. Both the kinetic energy matrix $\boldsymbol{T}$ and the potential energy matrix $\boldsymbol{V}$can be separately evaluated between the B-splines and then be added together, as shown below.

$$\langle \boldsymbol{H} \rangle = \sum_j \langle \boldsymbol{B_i}|\boldsymbol{H}|\boldsymbol{B_j}\rangle = \sum_j \langle \boldsymbol{B_i}|\boldsymbol{T} + \boldsymbol{V}|\boldsymbol{B_j}\rangle = \sum_j \langle \boldsymbol{B_i}|\boldsymbol{T}|\boldsymbol{B_j}\rangle + \sum_j \langle \boldsymbol{B_i}|\boldsymbol{V}|\boldsymbol{B_j}\rangle \tag{2.11}$$

And, $\boldsymbol{S} = \sum_j \langle \boldsymbol{B_i}|\boldsymbol{B_j}\rangle$(2.12)

The process of solving(2.10) involves the following sequential steps.

**Step 1**. Define the problem to be solved.

**Step 2**. Define the Domain length and choose a suitable Open Uniform knot Vector.

**Step 3.** Determine the B-splines and their first derivatives using the chosen knot vector.

**Step 4**. Determine the matrix elements of the kinetic and potential energies.

**Step 5**. Determine the eigenvalues and eigenvectors of (2.10) by using a suitable Eigen equation solver. (This is followed by Step 6 to plot results and analyze Values and their Errors).

## 3. Implementation of the Programming Steps

### 3.1 Step 1: Define the problem to be solved.

Consider a one-dimensional quantum particle of mass $\boldsymbol{m}$ which has only kinetic energy as it can move freely within a 1-D box of dimension $[-\boldsymbol{a}, \boldsymbol{a}]$. All eigenfunctions must vanish at the edges of the box and have their maximum amplitudes at the center of the box. Since the potential energy term $\boldsymbol{V(x) = 0}$, the particle has only non-zero kinetic energy, and its operator form from (2.1) is

$$\widehat{\boldsymbol{H}} = \widehat{\boldsymbol{T}} = \widehat{\boldsymbol{p}}^{\boldsymbol{2}}/\boldsymbol{2m} = -(\hbar^{\boldsymbol{2}}/\boldsymbol{2m})\boldsymbol{\nabla}^{\boldsymbol{2}} = -\boldsymbol{d}^{\boldsymbol{2}}/\boldsymbol{dx}^{\boldsymbol{2}} \tag{3.1}$$

Here we set $(\hbar^{\boldsymbol{2}}/\boldsymbol{2m}) \equiv \boldsymbol{1}$ so that all energies are in atomic units (au). Then (3.1) becomes,

$$\boldsymbol{E\psi = H\psi = T\psi = -d^{2}\psi/dx^{2} \equiv -\psi''} \tag{3.2}$$

The TISE to be solved is now: $\boldsymbol{\psi'' + E\psi = 0}$ (3.3)

If $\boldsymbol{V(x) > 0}$then (3.3) would be replaced by$\boldsymbol{\psi'' + (E - V)\psi = 0}$

(3.4)

The theoretical solution of (3.3) specifies that the energy of the level $\boldsymbol{n = 1, 2, 3}$, ... is given by,

$$\boldsymbol{En = n^{2}\,\pi^{2}/L^{2}\ ,\ n = 1, 2, 3, \ldots} \tag{3.5}$$

The theoretical ground state energy is given for level $\boldsymbol{n = 1}$ by $\boldsymbol{E1 = \pi^2/L^2}$and the energies of all excited states $\boldsymbol{n \geq 2}$can be found from (3.5) in the same atomic units.

(Note that if we take $(\hbar^2/m) \equiv 1$, then(3.3) changes to$\psi'' + 2E\psi = 0$. This changes the ground state energy to $E1 = 0.5(\pi^2/L^2)$and $En = n^2E1$for $n \geq 2$. If $V(x) > 0$also then instead of (3.4), we would have $\psi'' + 2(E - V)\psi = 0$ to solve.)

We now proceed to carry out all the steps from Eqn. (2.4) to Eqn. (2.10) and begin with a selection of a suitable knot vector in Step 1 in order to calculate all the required number of spline basis functions (see Eqn.2.5). As Python software is available to solve this problem, all the required steps are replaced by the corresponding code statements with relevant comments also.

### 3.2 Step 2:Define the Domain Length and Choose a Suitable Open Uniform Knot Vector.

Let the 1-D domain box limits (end-knots) be defined as $[-a, a]$ so that its length is $2a$ units**.** To begin with, set $a = 1$ so that it is confined to a box of length $L = 2a = 2$ units $(-1 \leq x \leq 1)$ so that at both ends, all eigenvectors vanish, and the free particle moves within those limits. The domain end-knots will now be converted into the open uniform (**augmented**) knot vector able to use B-splines of selectable degree $p$ and order $k = p + 1$ by repeating both end-knots $p$ times each and also introduce a number $m > 1$ of internal knots so that the domain length $2a$ is divided into a large number $N = m + 1$ of small intervals. This helps to use basis splines of a much **lower degree** or order than would be needed for an undivided domain length. The SPLIPY software follows this specification and computes the required number of basis spline functions to fit each sub-interval (piece) of the domain. The values set in the program are shown in Table 3.2.2, which also calls the snippet listed in Table 3.2.1 to create the appropriate augmented knot vector. A brief explanation is given below, but more details are to be found in [1] and [2].

The computation of B-splines (Step 2) requires a specified sequence of fixed coordinates spanning the length of the domain between its two limiting end-points, which are also fixed for a given domain. The length of the domain for this problem should be relatively large

enough so that all eigenfunctions do vanish at the end-points. Now, these fixed coordinates of end-points are referred to as **endknots,** and all points between end-knots are called **internal knots**. The simplest knot sequence is a **uniform** sequence in which all knots are equally spaced out in the domain and also monotonically increase from the starting knot-point to the ending knot-point. The minimum possible number of knots that a domain can have is just the two end-knots with only one interval between them. However, it is more usual to subdivide a domain length into many sub-intervals in order to increase the accuracy of the solution. Thus the number of intervals chosen for the domain also fixes the number of knots to be just one more than the number of intervals. This step will enable the use of basis functions of a much lower degree (or order). To help fix the order $\boldsymbol{k}$ and the degree $(\boldsymbol{k-1})$ of the B-splines desired to be used for the solution, the uniform knot sequence is augmented by the addition of the so-called **ghost knots** at both end-knots of the domain**.** These ghosts simply repeat both the domain end knots $(\boldsymbol{k-1})$ **times.** Thus if a domain end knots are $[\boldsymbol{a},\boldsymbol{b}]$ then B-splines of order $\boldsymbol{k=5}$ will be used if the end-knots are augmented by **4** ghost knots as in $[\boldsymbol{a},\boldsymbol{a},\boldsymbol{a},\boldsymbol{a},\boldsymbol{a},\ \boldsymbol{b},\boldsymbol{b},\boldsymbol{b},\boldsymbol{b},\boldsymbol{b}]$**.** This is known as an **open uniform** or **augmented knot sequence.** In a short form, the above-augmented sequence may be denoted by $[\boldsymbol{a}(\boldsymbol{k-1\ times}),\boldsymbol{a},\ \boldsymbol{n\_intervals},\ \boldsymbol{b},\boldsymbol{b}(\boldsymbol{k-1\ times})]$**.** Alternatively, this may also be denoted as $[\boldsymbol{a}(\boldsymbol{multiplicity},\ \boldsymbol{k}),\ \boldsymbol{n\_intervals},\ \boldsymbol{b}(\boldsymbol{multiplicity},\ \boldsymbol{k})]$**.** The thing to note about the $\boldsymbol{k}$ repeated end-knots at either side is that there exists a zero interval only between any two repeated knots, and so they all coincide with the domain end-knots. This augmentation procedure is a **necessary step** in the computation of B-splines according to the method devised and prescribed by Carl de Boor. The python code snippet to calculate the augmented knots sequence is given below.

Table 3.2.1. Define a function to calculate an augmented knots sequence. Code starts with importing needed modules shown below :

```
# -*- coding: utf-8 -*-
import numpy as np
from scipy import linalg as spLA
import matplotlib.pyplot as plt
#import splipy as Splnp  #--> done in Table 3.3.1
```

```
defget_augmented_knots ( degree,  a,  b,  N_intervals, type='uniform'):
N_points  =  N_intervals + 1
    if  type ==  'uniform' :       # then do the following
       #tile([a],  nrep) will repeat all elements [a]  nrep times (see below)
aug_knots  =  np.hstack( (np.hstack ( (np.tile (a, degree),
np.linspace (a,  b,  N_points) ) ),
np.tile (b, degree)) );
    return  aug_knots
# Set  values  of  all  the  argument  parameters below
degree_x = 14 ;  Xa,  Xb,  N_intervals  =  -1.0,  1.0,   201
# Now call the above function  by the  statement below
agknots = get_ augmented_knots ( degree_x,  Xa,  Xb,  N_intervals)
# N_augknots  =  [Xa (14 times) + (N_intervals + 1 = 202) +
Xb (14 times)]  =  230 knots
```

Then all problem-related parameters are defined and confirmed as follows.

Table 3.2.2. Initial programming steps assigning parameter values

```
################---------- MAIN ----------##################
## Set all physical parameters assumed for the TISE problem :
#me = 1.0; ## mass of the particle
## size of Box or x-domain range: Try different box sizes
#Xa, Xb = -0.5, 0.5
Xa,  Xb = -1.0,  1.0;  # Domain Box  limits used
#Xa,  Xb = -2.0,  2.0;
X_ab = Xb - Xa;  E1 = np.pi**2/X_ab**2   # L = box length;  E1 = GS Energy
# Set the number of intervals, quadrature points, degree/order, etc. below
N_intervals_x = 201   ## Try different values like  50, 100, 150 etc
N_quad = 181 #200;  number of quadrature points
degree_x = 14;  #degree of Bsplines
k_ordr_x = degree_x + 1;
N_splines_x = N_intervals_x + degree_x;   #   Number of  bsplines
N_base_x = N_splines_x - 2; # number of basis functions
N_dim = N_base_x; # N_base_x sets the dimensional size of the matrix

print("# All energies are in atomic units (hbar/(2m) == 1).")
print("# Order of the B-splines functions, k = {0}".format(k_ordr_x))
print("# Degree of the B-splines functions, k-1 = {0}".format(degree_x))
print("# Num of intervals {0} ans size of the basis set {1} in x".
                                format(N_intervals_x, N_base_x) )
print("#  Total  size  of  the  matrices,  N_dim  =  N_base_r  =
{0}".format(N_dim))
```

```
print("# Order of GL quadrature: N_quad = {0}".format(N_quad))
print("# Total number of basis sets = {0}".format(N_base_x* N_quad))
print("# Integration (= box) interval [{0}, {1}]".format(Xa, Xb))
print("# Domain (= box size) Length, X_ab = Xb-Xa = [{0}]".format(X_ab) )
print("# Ground State Energy (Theory), E1 = pi^2 / L^2 = %3.14f"%E1)
print("# Eigen-energies and eigen-functions are computed and plotted.\n")
```

Table 3.2.3. This is the printout of Table 3.2.2 to confirm all the defined parameters used in the solution code.

```
# All energies are in atomic units (hbar/(2m) == 1).
# Order of the B-splines functions, k = 15
# Degree of the B-splines functions, k-1 = 14
# Number of intervals 201 and size of the basis set 213 in x
# Total size of the matrices, N_dim = N_base_x = 213
# Order of GaussLegendre quadrature:  N_quad = 181
# Total number of basis sets = N_base_x * N_quad =  38553
# Integration (= box) interval [-1.0, 1.0]
# Domain (= box size) Length, X_ab = Xb-Xa =  [2.0]
# Ground State Energy (Theory) =  π^2/L^2  = 2.46740110027234
# Eigen-energies and functions are computed and plotted.
```

### 3.3 Step 3: Determine the B-splines and their First Derivatives

**Definition of Basis Splines**

B-Splines are piecewise polynomial functions defined on a given interval that contains a certain number of points, $\boldsymbol{t_i}$ referred to as a knot sequence, where$\boldsymbol{t_i \le t_{i+1}}$. The B-Splines of order $\boldsymbol{k}$ (degree $\boldsymbol{k-1}$) are denoted by $\boldsymbol{B(k,i,x)}$**.** This is the $\boldsymbol{i^{th}}$ basis function out of a possible set of $\boldsymbol{k}$ basis functions of order $\boldsymbol{k}$. They are given by the deBoor recurrence equations on this knot sequence as follows.

For first order B-spline,$k = 1$(or, degree, p = 0):

$B(1,i,x) = 1\ if\ t_i \le x \le t_{i+1}$(3.3.1a)

$B(1,i,x) = 0\ \ if\ x \le t_i\ or\ x \ge t_{i+1}$(3.3.1b)

For higher orders,$k > 1$(or, degrees, $p > 0$) :

$B(k,i,x) = [(x - t_i)/(t_{i+k-1} - t_i)]\, B(k-1,i,x) +$

$+[(t_{i+k} - x)/(t_{i+k} - t_{i+1})]B(k-1,i+1,x)$(3.3.2)

The calculation of B-splines of any higher-order always starts with the calculation of the first-order function $\boldsymbol{B}(\mathbf{1}, \boldsymbol{i}, \boldsymbol{x})$ using Eqn. (3.3.1). This should be substituted in Eqn. (3.3.2) to obtain the 2nd order basis set $\boldsymbol{B}(\mathbf{2}, \boldsymbol{i}, \boldsymbol{x})$ which is again substituted in Eqn. (3.3.2) to get the 3rd order basis set $\boldsymbol{B}(\mathbf{3}, \boldsymbol{i}, \boldsymbol{x})$**.** By repeating such substitutions into Eqn. (3.3.2), the basis spline functions of any desired order can finally be obtained. Efficient and fast Python software to evaluate B-splines and their derivatives have been obtained from [6] and used in this work (Alternatively, John Foster's bspline.py obtainable from [7] can be used). The code for computing basis splines and their derivatives is given below.

Table 3.3.1 Code to compute Basis Functions and call usage

```
#========== SPLIPY v.> 1.3.1 ==================**
deffSplipy(ordr, t_knots, t_nodes):   # <-- use ordr = degree+1
   import splipy as Splnp  # from Splipy v.1.3.1
   Basis = Splnp.BSplineBasis(order, t_knots)
Bs_t = Basis.evaluate(t_nodes,  d=0);
   d1Bs_t = Basis.evaluate(t_nodes,d=1); # 1st derivative
returnnp.array(Bs_t),np.array(d1Bs_t)
# call usage:
#bsr,dbsr= fSplipy(order, r_knots, r_nodes)
#bsz, dbsz= fSplipy(order, z_knots, z_nodes)
```

### 3.4 Step 4:Determine the Matrix Elements of T, V, and S (overlap) Integrals

Now the augmented knot vector is formed, and the nodes and weights of the Gauss-Legendre polynomials are obtained for the selected order N_quad at which the B-splines must be evaluated. This is similar to the use of Greville abscissas collocation done in the example of Sec.4 of [2]) to evaluate B-splines, but the use of GL quadrature ensures high accuracy. Now the basis spline piecewise functions are applied to each sub-interval of the knot vector. This results in a large number of basis spline functions to be handled, and this is why computer calculation is required. The Python code snippets, taken from [8] and suitably modified for use here, are is given in Table 3.4.1 below.

In Table 3.4.2, given next, the evaluation of the required matrices can be obtained for **Vx()** the potential energy matrix. This is **zero for a free quantum particle** and will not be called.

In Table 3.4.3, given later, the evaluation of the required matrices are obtained for the overlap integral **S**, kinetic energy **T,** and non-zero potential energy **V,** according to Eqn. (2.11) and (2.12).

Table 3.4.1: code for B-spline calculation, GL quad Collocation, and application to all Knot Vector sub-intervals

```
#get knots sequence to define Bspline under the uniform distribution
x_agknots =  augment_knots_vector(degree_x, 'uniform',  Xa,  Xb,
N_intervals_x)
print("augknots.shape = ",x_agknots.shape)
#get nodes and weights for Gauss-Legendre quadrature
x, w = np.polynomial.legendre.leggauss(N_quad);
## now use the nodes and weights for the integrals in x
x_nodes, wx_weights = np.array([]), np.array([])
for i in range(N_intervals_x+1):
aux_x = (0.5*(x_agknots[i+degree_x+1] - x_agknots[i+degree_x])*x
        +0.5*(x_agknots[i+degree_x+1] + x_agknots[i+degree_x]));
aux_w =  0.5*(x_agknots[i+degree_x+1] - x_agknots[i+degree_x])*w;
x_nodes = np.hstack((x_nodes, aux_x));
wx_weights = np.hstack((wx_weights, aux_w));
wx_weights = np.tile(wx_weights, (N_splines_x, 1));
#--------------------Using SPLIPY ver 1.3.1-------------------------
Bsx,  dBsx  =  fSplipy ( k_ordr_x,  x_agknots,  x_nodes )
Splines = np.array( [ [ Bsx[i, j]  for  j  in  range( 1,  N_splines_x-1 ) ]
                        for i  in  range( N_quad * ( N_intervals_x+1 ) ) ] )
#plt.figure();  plt.plot( x_nodes, splines );  plt.grid();  plt.show()
```

Table 3.4.2: Code to compute PE Matrix Vx

```
defV_aho_sho( x ,  gb = .1): # gb is a strength factor
    #Standard form of SHO: U = 0.5*me*omega**2*x**2;
    # Energies are in atomic units (au): m = hbar = omega = 1
Vx = 0.5*x*x ;   Vstr = "$0.5 x^2$"  % gb;
    #Vx = x*x*(0.5 + gb*x**4) ;  Vstr="$0.5x^2+%.3f x^6$" % gb;
    #Vx = x*x*(0.5 + gb*x**6) ;  Vstr="$0.5x^2+%f x^8$" % gb;
    return  Vx,  Vstr
# Potential Energy matrix  Vx  due to SHO or AnHO
Vx, Vstr = V_aho_sho(x_nodes)  # Vstr is a label string for Fig
```

```
Vx = np.tile( Vx, ( N_splines_x,  1 ) ); #repeat Vx once here
Vx = np.dot ( Bsx.T,  (Vx.T * wx_weights.T * Bsx));
Vx = np.array( [ [Vx[ i , j ] for i in range( 1 , N_splines_x - 1)]
                    for j in range ( 1 ,  N_splines_x – 1 )  ]  );
```

Table 3.4.3: Code to Compute Matrices for S and T using B-splines
The coefficient array C (Eqn 2.10) has also been folded in here

```
# Calculate the overlap matrix Sx in x
Sx = np.dot(Bsx.T, wx_weights.T * Bsx);  # X.T ==  X  Transposed
Sx = np.array( [ [Sx[i,j] for i in range ( 1, N_splines_x-1) ]
                          for j in range ( 1, N_splines_x-1)  ] );
# Calculate the Kinetic Energy matrix Tx  in  x
# Use this  identity: <(- B''(k,i,x), B(k,j,x) > = ( B'(k,i,x),  B'(k,j,x) )
# that is, (-np.dot(d2Bsx, dBsx) ) <==> ( dot(dBsx, dBsx) ) <--  below
Tx = np.dot(dBsx.T, wx_weights.T * dBsx);# −<ψ.ψ''> = <ψ'.ψ'>
Tx = np.array( [ [Tx[i,j]  for  i  in  range ( 1, N_splines_x - 1 ) ]
                           for  j  in  range (1, N_splines_x - 1) ] );
## Form the Hamiltonian H(x) = T(x) since V(x) = 0
Hx  =  Tx              # Vx() = 0 for a free particle
# Hx = Tx  +  Vx   # only for non-zero  potential  Vx
#plt.matshow(Hx[:7,:8],cmap=plt.cm.jet);plt.grid(); #Matrix image
```

### 3.5 Step 5: Determine the Eigenvalues and Eigenvectors

Here we implement the code for Eqn. (2.10) by using the well-known Eigen-equation solver module **eigh(Hx, Sx)** from Scipyas shown in Table 3.5.1 below.

Table 3.5.1 Code to Compute Eigenvalues and Eigenvectors

```
Evals, Evecs = spLA.eigh(Hx, Sx);   # Evals[Ndim]; Evecs[Ndim,Ndim]
print("len(Evals)=",len(Evals),"; Evecs.shape =",Evecs.shape)
print("Evals[] in atomic units (m = hbar =1; hbar^2 /(2 m) = 1):-")
print("# Ground State energy (B-Splines) = %3.14f\n"%Evals[0])
#Compute theoretical E[n] and print Error = Evals[] - Eth[] alongside
#E1 = (np.pi/X_ab)**2 # Theoretical Ground State Energy (n=1)
En = [n*n*E1 for n in range(1,N_dim+1)]  # theoretical levels (n >= 1)
print(" Eigen Energies (Bsplines)       Error = E(bspl) - E(n^2.(pi/L)^2)" )
print("-"*65)
for  ii  in   range ( N_dim ) :
    #get abs(rel err in EvalswrtEn(theory) ); this is either +ve or -ve
    # depending on if Evals[n] <= En(theory) or if Evals[n] >En(theory)
    print( "Evals[{:3d}] :  {:3.12e} ;    Err[ {:3d} ] : { :3.4e } ".format
```

```
                    ( ii, Evals[ii], ii , abs( 1.0 – Evals [ii] / En [ii] ) ) )
# collect a few lowest Evecs, set by nplot below, to plot
nplot = 5  # form the matrix wfnplt[nplot,N_dim]
wfnplt = np.array([[Evecs[i,j] for i in range(N_dim)]
                   for j in range(nplot)]) # wfnplt[nplot,N_dim]
print("wfnplt:",wfnplt.shape)
nEvecs = []   #  np.zeros( ( nplot  ,  N_dim),  float)
for  ipl  in  range  (nplot)  :
nEvecs.append( np.dot(wfnplt[ipl,:], Splines.T))
nEvecs  =  np.array(nEvecs)
print( " nEvecs.shape  =  ",  nEvecs.shape)
```

### 3.6 Step 6: Code to Plot Results Obtained and Print the Values with Errors

Table 3.6.1 Code to Plot Energy levels and Probability Densities

```
plt.figure( 1,  figsize=(6,4),dpi=180)
#plt.plot(x_nodes, V[-1,:],"k") #,label ="$V(x,ħ=m=\omega=1)$")
for ipl in range(nplot): # draw lowest nplot energy levels, Evals
plt.plot([x_nodes[0],x_nodes[-1]],[Evals[ipl],Evals[ipl]],'k-',
label=r"$E[%d]=%2.3f(au)$"%(ipl,Evals[ipl]))
plt.plot(x_nodes,Evals[ipl]+5*nEvecs[ipl]**2,'b--',
ms=4,label=r"$\psi^2_{%d}(x)$"%ipl)
plt.grid(which="both");
plt.xlim(Xa-.1,Xb+.1); #plt.xlim(-4.1,4.1);plt.ylim(-0.1,Evals[6]) #8.1)
plt.xlabel("x (atomic units)")
plt.ylabel("Energy levels (au)") #Radial Wavefunction")
plt.legend(loc="best",ncol = 2,frameon=False, fontsize=8)
plt.title("Lowest Eigen Energies(au) and Prob Densities for $V(x)=0$")
plt.show()

#Plot Eigen energies and their errors from theoretical values
#nvals = N_dim; #pi2 = np.pi**2; E1 = pi2/(X_ab**2) # GS Energy (n=1)
#En = [n*n*E1 for n in range(1,nvals+1)]  # theoretical levels
n201 = np.linspace(1,201,201)
plt.figure( 2,  figsize=(6, 6),dpi=120)
plt.plot(n201,En[:201],"r",lw=2,label="$En(theory)$")
plt.plot(n201,Evals[:201],"k",lw=1,label="$En(Bsplines)$")
#plt.plot(n201,En[:201],"r",lw=1,label="$En(theory)$")
plt.plot(n201, Evals[:201]-En[:201],"g",lw=2,label="$Err=Evals-Etheo$")
plt.grid(which="both"); plt.minorticks_on()
plt.legend(loc="best",frameon=False)
```

```
plt.title("Evals, En(theory) and Err = Evals-En(theo) for $V(x)=0$")
plt.show()
```

### 3.7 Computation and Results

A complete computer program file to be named "**FreeParticle_TISE_Bsplin.py**" may be assembled (using copy and paste method) by combining only the listing in each of the above tables starting with Table 3.2.1, then Table 3.3.1, followed successively by Table 3.2.2, Table 3.4.1, Table 3.4.2, Table 3.4.3, Table 3.5.1 and end with Table 3.6.1 in a Python-3 set up with Spyder IDE where editing and executing the program can be done.

This program can be given fixed values of the parameters shown in Table 3.2.2, which Table 3.2.3 confirms as output. Generally, the integration domain length and the degree of B-splines to be used can be kept fixed during a run. Then we can vary the $\boldsymbol{N\_intervals}$= the number of intervals between end-knots say from 25 to 250 and $\boldsymbol{N\_quad}$= the order of GaussLegendre quadrature say from 20to 220($\boldsymbol{N\_quad \approx 0.9 * N\_intervals}$ **, for example,** is a good choice) in each run to view the plots and the computed energy levels along with their differences from their theoretical values given by $\boldsymbol{En = n^2\,\pi^2/L^2}$(3.5). The level energies are all positive since they concern the quantum motion of a free particle only bounded by the length**L= 2** of the domain box and with $\boldsymbol{n^2}$ they rapidly increase to large values. Both **Table 3.7.1** and **Table 3.7.2** show a selected listing of level energies in the 2nd column, a difference $\Delta\boldsymbol{E}$ with respect to the theoretical values in the 3rd column, and the fractional error in the last column for the computation parameter settings shown in the 2nd row of each table. Some of the differences in the 3rd column are negative due to the angular nature of the Legendre polynomials used for collocation.

Table 3.7.1 Energies (au) Difference Frac. Error

| E#n | E(n) (bsplines) | $\Delta E = En - (n\,\pi/L)^2$ | $\Delta E/(n\pi/L)^2$ |
|---|---|---|---|
| 1 | 2.46740110027236e+00 | 1.6875e-14 | -6.8834e-15 |
| 2 | 9.86960440108968e+00 | 3.2507e-13 | -3.2863e-14 |
| 3 | 2.22066099024505e+01 | -5.4712e-13 | 2.4647e-14 |
| 4 | 3.94784176043572e+01 | -2.7711e-13 | 6.9944e-15 |
| 5 | 6.16850275068088e+01 | 2.9843e-13 | -4.8850e-15 |

| | | | |
|---|---|---|---|
| 6 | 8.88264396098041e+01 | -8.5265e-14 | 9.9920e-16 |
| 7 | 1.20902653913345e+02 | 4.2633e-14 | -4.4409e-16 |
| 8 | 1.57913670417429e+02 | -4.8317e-13 | 3.1086e-15 |
| 9 | 1.99859489122059e+02 | -3.6948e-13 | 1.8874e-15 |
| 10 | 2.46740110027234e+02 | -3.6948e-13 | 1.4433e-15 |
| 11 | 2.98555533132952e+02 | -7.9581e-13 | 2.6645e-15 |
| 12 | 3.55305758439216e+02 | -9.6634e-13 | 2.6645e-15 |
| 13 | 4.16990785946025e+02 | -6.8212e-13 | 1.6653e-15 |
| 15 | 5.55165247561277e+02 | 3.4106e-13 | -6.6613e-16 |
| 16 | 6.31654681669719e+02 | 4.5475e-13 | -6.6613e-16 |
| 17 | 7.13078917978706e+02 | -3.4106e-13 | 4.4409e-16 |
| 18 | 7.99437956488237e+02 | -9.0949e-13 | 1.1102e-15 |
| 19 | 8.90731797198314e+02 | -6.8212e-13 | 7.7716e-16 |
| 20 | 9.86960440108936e+02 | -1.1369e-13 | 1.1102e-16 |
| 25 | 1.54212568767027e+03 | 5.7298e-11 | -3.7081e-14 |
| 26 | 1.66796314378435e+03 | 2.5352e-10 | -1.5210e-13 |
| 28 | 1.93444246261834e+03 | 4.8301e-09 | -2.4969e-12 |
| 30 | 2.22066099032555e+03 | 8.0440e-08 | -3.6223e-11 |
| 32 | 2.52661872785510e+03 | 1.1762e-06 | -4.6553e-10 |
| 34 | 2.85231568721150e+03 | 1.5297e-05 | -5.3629e-09 |
| 36 | 3.19775200691514e+03 | 1.8096e-04 | -5.6590e-08 |
| 38 | 3.56292919689410e+03 | 2.0081e-03 | -5.6361e-07 |
| 40 | 3.94786295230101e+03 | 2.1192e-02 | -5.3680e-06 |
| 41 | 4.14776803568850e+03 | 6.6786e-02 | -1.6102e-05 |
| 43 | 4.56281101076271e+03 | 5.8638e-01 | -1.2853e-04 |
| 45 | 5.00051383856655e+03 | 4.0266e+00 | -8.0589e-04 |
| 46 | 5.23098902109365e+03 | 9.9683e+00 | -1.9093e-03 |
| 48 | 5.75019478300871e+03 | 6.5303e+01 | -1.1487e-02 |
| 49 | 6.03377565809428e+03 | 1.0955e+02 | -1.8491e-02 |
| 50 | 6.16850275065835e+03 | -2.2502e-08 | 3.6479e-12 |

Table 3.7.2 Energies (au) Difference Frac. Error

(params: L=2; Degree=14; N_intervals = 100; N_quad = 90)

| #n | E(n) (bsplines) | $\Delta E = En - (n\,\pi/L)^2$ | $\Delta E/(n\pi/L)^2$ |
|---|---|---|---|
| 1 | 2.46740110027229e+00 | -5.0626e-14 | 2.0539e-14 |
| 2 | 9.86960440109300e+00 | 3.6469e-12 | -3.6948e-13 |
| 3 | 2.22066099024531e+01 | 2.0570e-12 | -9.2593e-14 |
| 4 | 3.94784176043612e+01 | 3.7943e-12 | -9.6145e-14 |
| 5 | 6.16850275068159e+01 | 7.4394e-12 | -1.2057e-13 |
| 6 | 8.88264396098129e+01 | 8.6260e-12 | -9.7033e-14 |
| 7 | 1.20902653913355e+02 | 1.0559e-11 | -8.7264e-14 |

| | | | |
|---|---|---|---|
| 8 | 1.57913670417438e+02 | 8.4412e-12 | -5.3513e-14 |
| 9 | 1.99859489122070e+02 | 1.0402e-11 | -5.1958e-14 |
| 10 | 2.46740110027243e+02 | 9.4929e-12 | -3.8414e-14 |
| 15 | 5.55165247561284e+02 | 7.9581e-12 | -1.4433e-14 |
| 20 | 9.86960440108938e+02 | 2.0464e-12 | -1.9984e-15 |
| 30 | 2.22066099024510e+03 | -1.8190e-12 | 7.7716e-16 |
| 40 | 3.94784176043574e+03 | 9.0949e-13 | -2.2204e-16 |
| 50 | 6.16850275068108e+03 | 2.3101e-10 | -3.7526e-14 |
| 57 | 8.01658617482659e+03 | 4.1761e-08 | -5.2094e-12 |
| 58 | 8.30033730140095e+03 | 8.4803e-08 | -1.0217e-11 |
| 65 | 1.04247696583489e+04 | 9.6982e-06 | -9.3031e-10 |
| 66 | 1.07479992113375e+04 | 1.8551e-05 | -1.7260e-09 |
| 72 | 1.27910081064049e+04 | 8.0259e-04 | -6.2747e-08 |
| 73 | 1.31487819403591e+04 | 1.4770e-03 | -1.1233e-07 |
| 77 | 1.46292373653994e+04 | 1.6242e-02 | -1.1102e-06 |
| 81 | 1.61887853651727e+04 | 1.6675e-01 | -1.0300e-05 |
| 86 | 1.82515404430037e+04 | 2.6419e+00 | -1.4477e-04 |
| 90 | 2.00059411216333e+04 | 1.9992e+01 | -1.0003e-03 |
| 96 | 2.30344381469224e+04 | 2.9487e+02 | -1.2967e-02 |
| 99 | 2.45318141508106e+04 | 3.4882e+02 | -1.4424e-02 |
| 100 | 2.46740110022089e+04 | -5.1448e-07 | 2.0851e-11 |

In both tables, about the first 40% of the energies have a fractional error less than $\mathbf{10^{-15}}$and the next 20% less than $\mathbf{10^{-10}}$ and thereafter, the fractional error increases to 0.01 levels. The lowest five energy levels overlaid with eigen-probability densities are shown in **Figure 3.7.1**. The **50** rows of values in **Table 3.7.1** are shown graphed in **Figure 3.7.2,** while the 100 rows of values in **Table 3.7.2** are similarly graphed in **Figure 3.7.3**.

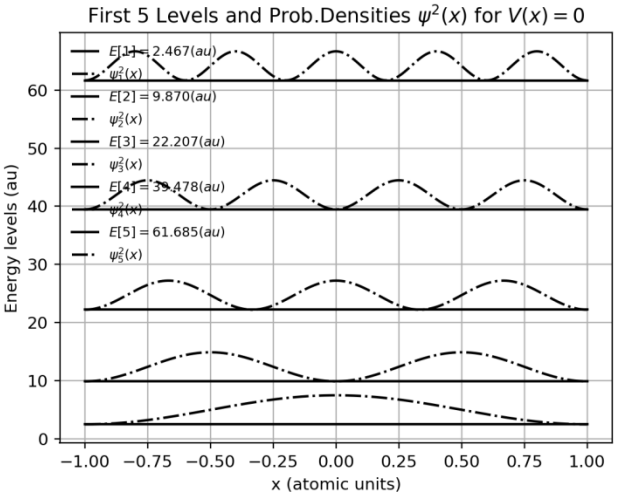


**Figure 3.7.1** The five lowest levels are shown overlaid by respective eigen-densities.

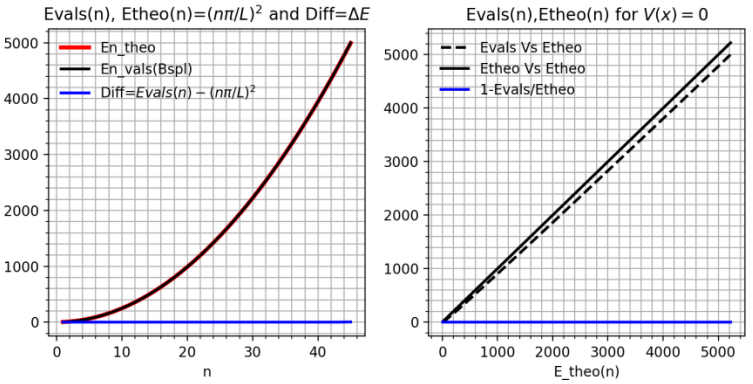


**Figure 3.7.2:** On the left, both computed and theoretical energies obtained for parameters of Table 3.7.1 increase quadratically with $n$ and are seen to coincide well. On the right, the gap due to the energy difference between the solid diagonal line of Etheo(n) and the dashed line of Evals(n) is seen to increase with the quantum number $n$as $n^2$ but the fractional error coincides with the abscissa indicating good accuracy.

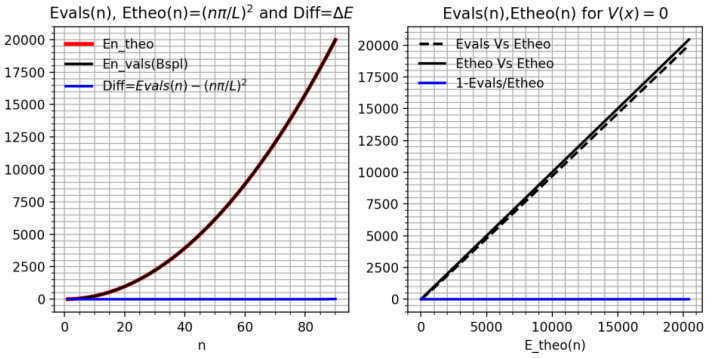


**Figure 3.7.3**: On the left, both computed and theoretical energies obtained for parameters of Table 3.7.2 increase quadratically with $n$ and are seen to coincide well. On the right, the gap due to the energy difference between the solid diagonal line of Etheo(n) and the dashed line of Evals(n) is seen to increase with the quantum number $n$as $n^2$ but the fractional error coincides with the abscissa indicating good accuracy.

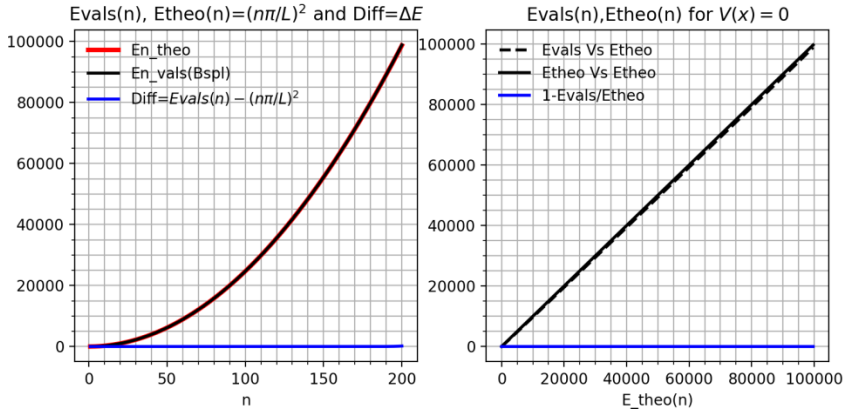


**Figure 3.7.4:** This graph shows all the 200 levels plotted obtained by using parameter settings of L= 2; Degree=14; N_intervals = 220, and N_quad = 200. The caption under figure 3.7.3 applies here also but only for the last two parameters given here. However, while no tabulated values are provided for this computation, the values display similar behaviour regarding values, differences, and fractional errors apparent in both the tables given above.

## 4. Conclusion

The reasonably excellent results obtained and tabulated in the two Tables accompanied by three sets of graphs shown in Sec. 3.7 were

made possible by the use of suitable open uniform (augmented) knot vectors with enough internal knots, computing B-spline functions of suitable degree, and carrying out collocation with GL quadrature generated nodes as well as weights of a chosen order. Thus, the use of B-spline collocation enabled the achievement of high accuracies by the application of the Rayleigh Ritz variational method to the computational work done here. Similarly, B-spline collocation has enabled accurate calculations in a wide array of applications and many of these applications are described in [1].

## References

[1] H.Bachauet.al."Applications of B-splines in atomic and molecular physics", Rep. Prog. Phys. Vol. 64, pp. 1815-1942, 2001.
[2] M.N. Anandaram, "Bernstein Polynomials: Properties and Applications to Bezier Curves, B-Splines and Solution of BVPs", Submitted to MJOS
[3] https://en.wikipedia.org/wiki/De_Boor's_algorithm
[4] CarldeBoor, "B-Spline Basics", https:// apps.dtic.mil/ dtic/tr/ fulltext/u2/a172773.pdf
[5] C. deBoor and B. Swartz, "Collocation at Gaussian Points", SIAM. Journal of Numerical Analysis, Vol.10, No.4, pp. 582-606, 1973. Download pdf copy from https:// www.researchgate.net/ publication/ 238747998_Collocation_at_Gaussian_Points
[6] Access "SPLIPY 1.3.1" from https://github.com/sintefmath/Splipy/
[7] Access"Bspline.py" from https://github.com/johntfoster/bspline
[8] M.Garagiola, "Quantum Harmonic Oscillator in 1-D", GitHub.com, 2016. https:// github.com/ marianogaragiola/ Schrodinger_ Equation_python

## Conflict of Interest statement

No funding or any help has been received from any funding Agency or persons respectively. Hence there is no conflict of interest with any entity.

## Author contribution statement

The work reported in this single-authored paper has been fully (100%) carried out by M.N. Anandaram.