Time-independent Schrödinger equation in 1D

For the time-dependent Schrödinger equation

$$i\hbar\frac{\partial}{\partial t}|\psi(t,x)\rangle = \hat{H}|\psi(t,x)\rangle$$

can be decomposed by the method of separation of variables when the potential is time-independent $V = V(x)$, that is

$$\hat{H}|\psi(x)\rangle = E|\psi(x)\rangle,$$

where the hamiltonian is given by

$$\hat{H} = -\frac{\hbar^2}{2m}\nabla^2 + V(x).$$

Problem

The goal of this project is to solve time-independent Schrödinger equation in 1-dimension called "Eigen Probelm", that is

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

Discrete Schrödinger equation

The time-independent Schrödinger equation is

$$-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}|\psi\rangle + V|\psi\rangle = E|\psi\rangle,$$

where is the column vector

$$|\psi\rangle = \psi_i, \quad i = 0, 1, \ldots, n-1,$$

potential V is a diagonal matrix

$$V = V_i\,\delta_{i,j}, \quad i, j = 0, 1, \ldots, n-1,$$

and the 1-dimension laplacian operator ∇^2 approximate to a $n \times n$ matrix, since

$$\frac{d^2 f(x)}{dx^2} \approx \frac{f(x-\delta_x) - 2f(x) + f(x+\delta_x)}{\delta_x^2},$$

we define

$$\frac{d^2}{dx^2}f_i = \frac{1}{\delta_x}(f_{i-1} - 2f_i + f_{i+1}), \quad i = 0, 1, \ldots, n-1,$$

where $f_{-1} = f_n = 0$, plugging we have

$$-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}\psi_i + V_i\,\delta_{i,j}\psi_i = E\psi_i,$$

or

$$-\frac{\hbar^2}{2m}\frac{1}{\delta x^2}\begin{pmatrix} -2 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & -2 \end{pmatrix}\begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \psi_3 \\ \vdots \\ \psi_{n-3} \\ \psi_{n-2} \\ \psi_{n-1} \end{pmatrix} + \begin{pmatrix} V_0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & V_1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & V_2 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & V_3 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & V_{n-3} & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & V_{n-2} & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & V_{n-1} \end{pmatrix}\begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \psi_3 \\ \vdots \\ \psi_{n-3} \\ \psi_{n-2} \\ \psi_{n-1} \end{pmatrix} = E\begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \psi_3 \\ \vdots \\ \psi_{n-3} \\ \psi_{n-2} \\ \psi_{n-1} \end{pmatrix},$$

So, now it becomes solving this eigen problem for a large system

$$\begin{pmatrix} \frac{\hbar^2}{\delta_x^2 m} + V_0 & -\frac{\hbar^2}{2\delta_x^2 m} & 0 & 0 & \cdots & 0 & 0 & 0 \\ -\frac{\hbar^2}{2\delta_x^2 m} & \frac{\hbar^2}{\delta_x^2 m} + V_1 & -\frac{\hbar^2}{2\delta_x^2 m} & 0 & \cdots & 0 & 0 & 0 \\ 0 & -\frac{\hbar^2}{2\delta_x^2 m} & \frac{\hbar^2}{\delta_x^2 m} + V_2 & -\frac{\hbar^2}{2\delta_x^2 m} & \cdots & 0 & 0 & 0 \\ 0 & 0 & -\frac{\hbar^2}{2\delta_x^2 m} & \frac{\hbar^2}{\delta_x^2 m} + V_3 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \frac{\hbar^2}{\delta_x^2 m} + V_{n-3} & -\frac{\hbar^2}{2\delta_x^2 m} & 0 \\ 0 & 0 & 0 & 0 & \cdots & -\frac{\hbar^2}{2\delta_x^2 m} & \frac{\hbar^2}{\delta_x^2 m} + V_{n-2} & -\frac{\hbar^2}{2\delta_x^2 m} \\ 0 & 0 & 0 & 0 & \cdots & 0 & -\frac{\hbar^2}{2\delta_x^2 m} & \frac{\hbar^2}{\delta_x^2 m} + V_{n-1} \end{pmatrix}\begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \psi_3 \\ \vdots \\ \psi_{n-3} \\ \psi_{n-2} \\ \psi_{n-1} \end{pmatrix} = E\begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \psi_3 \\ \vdots \\ \psi_{n-3} \\ \psi_{n-2} \\ \psi_{n-1} \end{pmatrix}$$

Algorithm

import package

In [ ]:

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy.linalg import eigh_tridiagonal
```

setting plotting default

Matplotlib: how to change default style

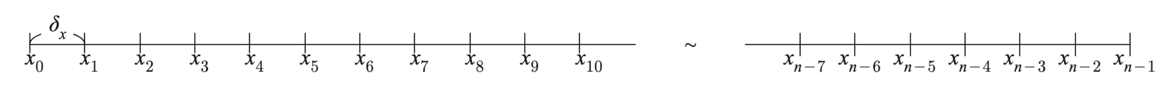
How to change the font size on a matplotlib plot

In [ ]:

```
mpl.rcParams['figure.dpi'] = 400
mpl.rcParams['figure.figsize'] = [12,3]
mpl.rcParams['lines.linewidth'] = 0.5
mpl.rcParams['lines.color'] = 'red'
mpl.rcParams['text.usetex'] = True
mpl.rcParams['font.family'] = 'Times New Roman'
mpl.rcParams['font.size'] = 8
mpl.rcParams['axes.prop_cycle'] = mpl.cycler('color','bgrcmyk')
mpl.rcParams['axes.linewidth'] = 0.2
plt.rcParams["mathtext.fontset"] = "cm"
plt.rc('axes', labelsize=8)
plt.rc('font', size = 12)
plt.rc('xtick', labelsize=5)
plt.rc('ytick', labelsize=5)
```

seting the position space

$$x = [x_i, x_i + dx, \ldots, x_f]$$



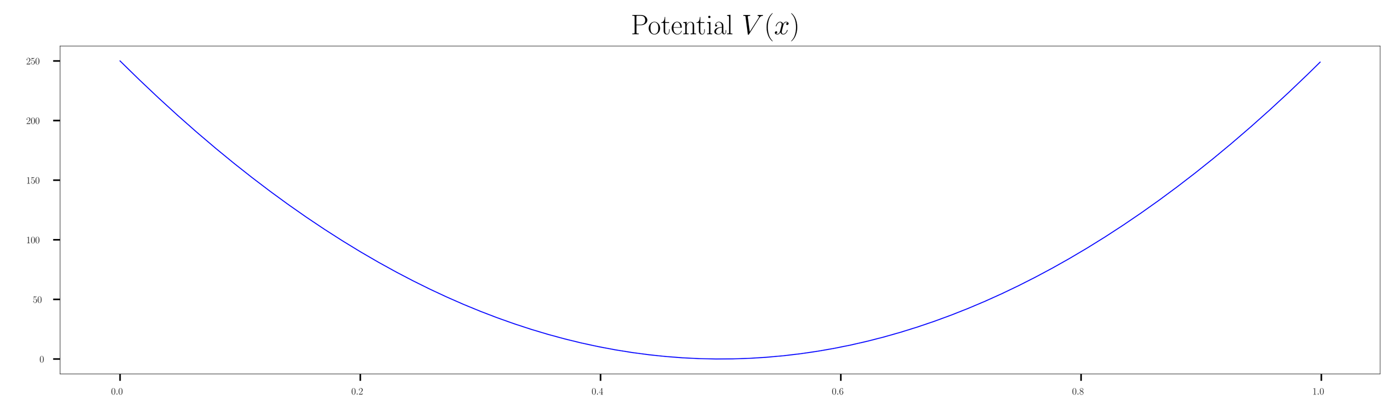
In [ ]:

```
xi = 0      # initial point
xf = 1      # final point
dx = 0.001   # step (delta_x)
x = np.arange(xi,xf,dx)
```

define the potential function

In [ ]:

```
f=plt.figure()
def V(x):
    v = 1000 *(x-0.5)**2
    #v[x<0.5]=0
    return v
plt.plot(x,V(x))
plt.title("Potential $V(x)$")
plt.show()
f.savefig('assets/Potential.pdf')
```



Solving the eigen problem

Since the discrete Hamiltonian is an tridiagonal matrix in the form

$$\hat{H} = \begin{pmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{pmatrix},$$

we can use tridiagonal matrix algorithm to solve it, which only need $\mathcal{O}(n)$ operations rather than $\mathcal{O}(n^3)$ operations by Gaussian elimination.

Here using `eigh_tridiagonal(d,e)` function, built up by scipy.

In [ ]:

```
d = 1/dx**2 + V(x)
e = -1/(2*dx**2) * np.ones(len(x)-1)
w,v = eigh_tridiagonal(d,e)
```

Define the Eigenfunction ψ_i and Eigenvalue E_i

- Eigenfunction ψ_i, $i = 0, 1, \ldots, \text{length}\,(x)$
- Eigenvalue E_i, $i = 0, 1, \ldots, \text{length}\,(x)$

In [ ]:

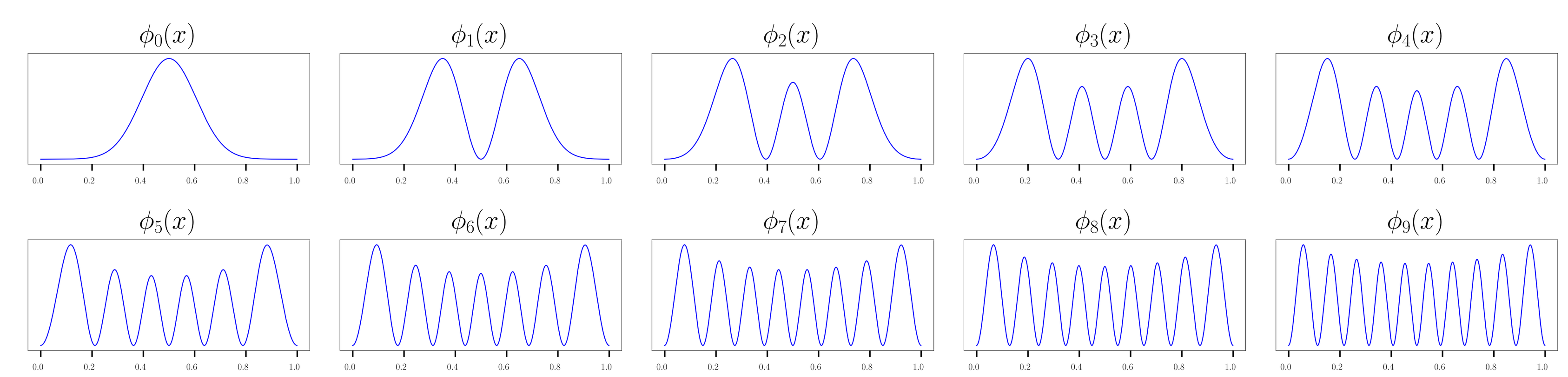
```
phi = v.T
E = w
```

Plot the Eigenfunction

$$|\phi_i(x)|^2, \quad i = 0, 1, \ldots, 9$$

In [ ]:

```
row = 2
col = 5
f=plt.figure()
for i in range(row*col):
    plt.subplot(row,col,i+1)
    plt.plot(x,phi[i]**2)
    plt.title("$\phi_{"+str(i)+"}(x)$")
    plt.yticks([])
f.tight_layout()
f.savefig('assets/Eigenfunction.pdf')
```



In [ ]:

```
E_N = 100
f, (a0,a1) = plt.subplots(1, 2, gridspec_kw={'width_ratios': [1, 10]})
for i in range(E_N):
    a0.axhline(y = E[i])
a1.plot(E[0:E_N],'.',markersize=2)
a0.set_yticks([])
a0.set_xticks([])
a1.set_yticks([])
a1.set_xlabel('$n$')
a1.set_ylabel('Energy')
f.suptitle('Eigenvalue $E_n$')
f.savefig('assets/Eigenvalue.pdf')
```