Cubic Perturbation

March 3, 2021

0.1 Cubic perturbation in a harmonic potential

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate
import scipy.optimize
```

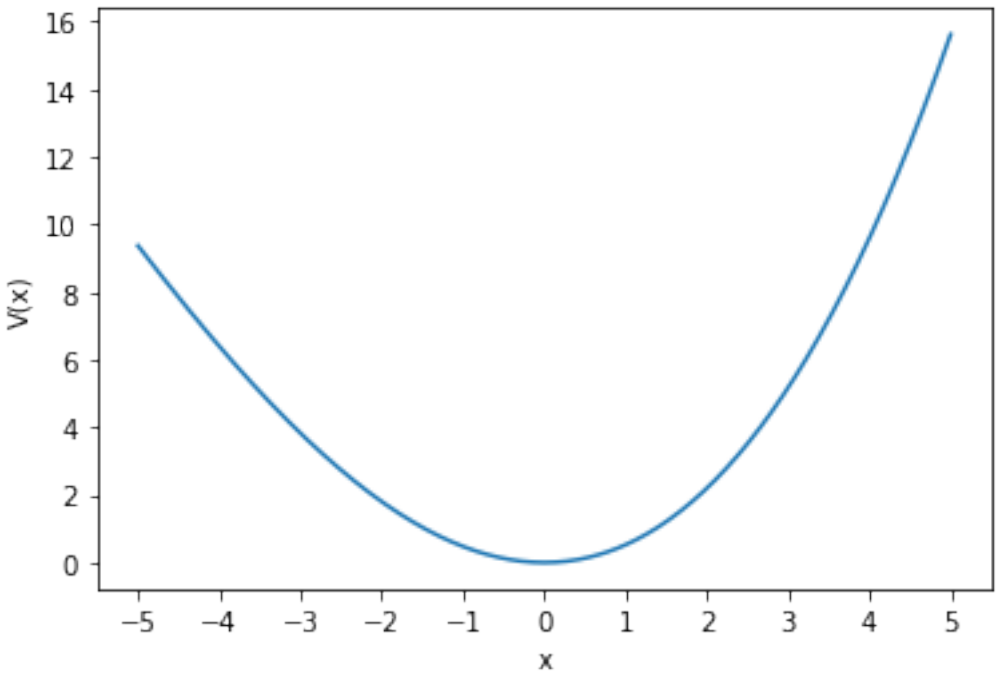
[2]:
```
class Schroedinger :

    def __init__(self) :
        self.hbar = 1.0                  # Planck's constant / 2pi
        self.m = 1.0                     # particle mass
        self.omega = 1.0                 # oscillator frequency
        self.E = 0.0                     # current energy in search
        self.N = 500                     # number of lattice points = N+1
        self.x_left = -5.0               # left boundary
        self.x_right = 5.0               # right boundary
        self.h = (self.x_right - self.x_left) / self.N  # grid spacing
        self.xvals = np.linspace(self.x_left,self.x_right,self.N+1) # x value␣
↪for phi
        self.phi_left = np.zeros(self.N+1)  # wave function integrating from␣
↪left
        self.phi_right = np.zeros(self.N+1) # wave function integrating from␣
↪right
        self.phi = np.zeros(self.N+1)       # whole wave function
        self.i_match = 0                 # Index of turning point
        self.x_match = 0.                # Turning point x value
        self.sign = 1                    # current sign used to make F(E)␣
↪continuous
        self.nodes = 0                   # current number of nodes in␣
↪wavefunction


    def V(self, x):                    # harmonic oscillator with a cubic␣
↪perturbation
        return 0.5 * self.m * self.omega**2 * (x**2 + (1/20) * x**3)
```

```
    def q(self, x):                    # Sturm-Liouville q function
        return 2 * self.m / self.hbar**2 * (self.E - self.V(x))


    def F(self, energy):               # eigenvalue at F(E) = 0

        # set energy needed by the q(x) function
        self.E = energy

        # find the right turning point
        self.i_match = self.N


        x = self.x_right               # start at right boundary
        while self.V(x) > self.E:          # in forbidden region
            self.i_match -= 1
            x -= self.h
            if self.i_match < 0:
                raise Exception("can't find right turning point")
        self.x_match = self.xvals[ self.i_match ]

        # integrate self.phi_left using Numerov algorithm
        self.phi_left[0] = 0.0
        self.phi_left[1] = 1.0e-10
        c = self.h**2 / 12.0           # constant in Numerov formula
        for i in range(1, self.i_match+1):
            x = self.x_left + i * self.h
            self.phi_left[i+1]  = 2 * (1 - 5 * c * self.q(x)) * self.phi_left[i]
            self.phi_left[i+1] -= (1 + c * self.q(x - self.h)) * self.
↪phi_left[i-1]
            self.phi_left[i+1] /= 1 + c * self.q(x + self.h)

        # integrate self.phi_right
        self.phi[self.N] = self.phi_right[self.N] = 0.0
        self.phi[self.N-1] = self.phi_right[self.N-1] = 1.0e-10
        for i in range(self.N - 1, self.i_match - 1, -1):
            x = self.x_right - i * self.h
            self.phi_right[i-1]  = 2 * (1 - 5 * c * self.q(x)) * self.
↪phi_right[i]
            self.phi_right[i-1] -= (1 + c * self.q(x + self.h)) * self.
↪phi_right[i+1]
            self.phi_right[i-1] /= 1 + c * self.q(x - self.h)
            self.phi[i-1] = self.phi_right[i-1]

        # rescale self.phi_left
        scale = self.phi_right[self.i_match] / self.phi_left[self.i_match]
```

```
        for i in range(self.i_match + 2):
            self.phi_left[i] *= scale
            self.phi[i] = self.phi_left[i]

        # make F(E) continuous
        # count number of nodes in self.phi_left
        n = 0
        for i in range(1, self.i_match+1):
            if self.phi_left[i-1] * self.phi_left[i] < 0.0:
                n += 1

        # flip its sign when a new node develops

        if n != self.nodes:
            self.nodes = n
            self.sign = -self.sign

        return ( self.sign *
         ( self.phi_right[self.i_match-1] - self.phi_right[self.i_match+1] -
           self.phi_left [self.i_match-1] + self.phi_left[self.i_match+1] ) /
        (2 * self.h * self.phi_right[self.i_match]) )

    def normalize(self):
        norm = 0.0
        norm = np.sqrt( np.sum( self.phi**2 ) / self.N )
        self.phi = self.phi / norm
```

0.2 Plotting the potential

[3]:
```
schroedinger = Schroedinger()


x = schroedinger.x_left + schroedinger.h * np.linspace(0, schroedinger.N,␣
 ↪schroedinger.N+1)
vx = schroedinger.V(x)
plt.plot(x,vx)
plt.xticks(np.linspace(-5,5,11,endpoint=True))
plt.xlabel("x")
plt.ylabel("V(x)")
plt.show()
```

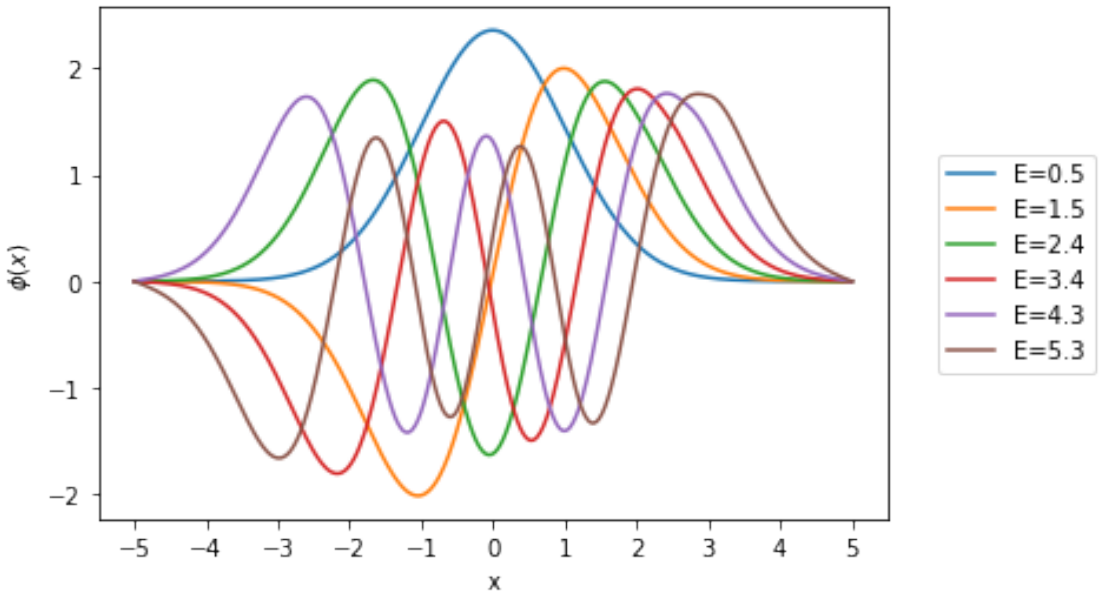


0.3 Plotting the wavefunction for different energy levels

[4]:
```
# find the energy levels
E_max = 5.0
schroedinger.E = 0.1        # guess an E below the ground state
level = 0                   # level number
E_old = 0.0                 # previous energy eigenvalue


while True:                 # loop over levels

    # estimate next E and dE
    dE = 0.5 * (schroedinger.E - E_old)
    E_old = schroedinger.E
    schroedinger.E += dE

    res = scipy.optimize.root_scalar( schroedinger.F,
                                      x0=schroedinger.E,
                                      x1=schroedinger.E+dE )
    schroedinger.E = res.root

    level += 1

    res = scipy.optimize.root_scalar( schroedinger.q,
```

```
                                    x0=schroedinger.x_left,
                                    x1=schroedinger.x_match )
    x = res.root
    swrite = '-: {0:4.2f} {1:4.2f}'.format( x, schroedinger.E )
    print (swrite)

    res = scipy.optimize.root_scalar( schroedinger.q,
                                    x0=schroedinger.x_match,
                                    x1=schroedinger.x_right )
    x = res.root
    swrite = '+: {0:4.2f} {1:4.2f}'.format( x, schroedinger.E )
    print (swrite)


    schroedinger.normalize()
    xvals = schroedinger.x_left + schroedinger.h * np.arange(schroedinger.N+1)

    plt.plot(xvals, schroedinger.phi, label="E=%2.1f" % (schroedinger.E) )

    if schroedinger.E >= E_max:          # we are done
        break
plt.legend(bbox_to_anchor=(1.05, 0.5), loc='center left')
plt.xticks(np.linspace(-5,5,11,endpoint=True))
plt.xlabel("x")
plt.ylabel(r"$\phi(x)$")
plt.show()
```

```
-: 0.97 0.49
+: 0.97 0.49
-: 1.65 1.47
+: 1.65 1.47
-: 2.10 2.44
+: 2.10 2.44
-: 2.46 3.40
+: 2.46 3.40
-: 2.76 4.35
+: 2.76 4.35
-: 3.03 5.28
+: 3.03 5.28
```



0.4 Plotting the root-finding function F(E) as a function of the energy

[5]:
```
%matplotlib notebook
# print the search function
schroedinger.E = 0.1
dE = 0.01
E_data = []
F_data = []
while schroedinger.E < E_max:
    E_data.append( schroedinger.E )
    F_data.append(  schroedinger.F(schroedinger.E) )
    schroedinger.E += dE

plt.plot(E_data,F_data)
plt.ylim(-10,10)
plt.xlabel("E")
plt.ylabel("F(E)")
plt.axhline(linewidth=1, color='k')
plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[ ]:
```

```