

Harmonic Oscillator

January 21, 2019

1 Computational Exercise 1: A Vary Great, Very Harmonic, Wavefunction

We will apply the Variational Method to the Harmonic Oscillator. In particular, we will optimize the trial wavefunction given by

$$\Phi(x) = \exp(-\alpha x^2), \quad (1)$$

where α is an adjustable parameter that will be variationally optimized. In particular, the goal will be to minimize the energy functional given by

$$E[\Phi(x)] = \frac{\int_0^\infty \Phi^*(x) \hat{H} \Phi(x) dx}{\int_0^\infty \Phi^*(x) \Phi(x) dx} \quad (2)$$

by modulating the trial wavefunction through the adjustable parameter α .

The Hamiltonian operator for Harmonic Oscillator is given by

$$\hat{H} = -\frac{\hbar^2}{2m} \frac{d^2}{dx^2} + \frac{1}{2} k x^2. \quad (3)$$

We will use a system of natural units where $\hbar = 1$, the mass of the particle $m = 1$, and the force constant of the potential $k = 1$.

1.0.1 Questions

1. What is the optimal value of $E[\Phi(x)]$ in natural units? How does this value compare to the true ground state energy of the Harmonic Oscillator?
2. What is the optimal value of α in natural units? How does $\Phi(x)$ with this value of α compare to the true ground state wavefunction?
3. What is the kinetic energy expectation value of the optimized trial wavefunction in natural units?
4. What is the potential energy expectation value of the optimized trial wavefunction in natural units?

1.0.2 (Hopefully) Helpful Hints

You can represent your trial wavefunction numerically on the computer by creating an array of x-values, evaluating the formula for your trial wavefunction at each of those x-values, and then storing each of those function values in another array. The following lines of code will do this using *numpy* arrays and the *numpy.exp* function.

```
In [ ]: import numpy as np
        ### create an array of 1000 x-values between x=-20 and x=20 atomic units
        x_array = np.linspace(-20,20,1000)
        ### pick a trial value of alpha... might as well be 1 atomic unit!
        alpha = 1.0
        ### evaluate your trial wavefunction at each of the x-values stored in the array called
        Phi = np.exp(-alpha * x_array**2)
```

To evaluate the kinetic energy of a trial wavefunction, we will need to take the second derivative of the trial wavefunction. One way to take derivatives on a computer is to use the method of finite differences. In particular, the forward finite difference approximation to the first derivative of your trial wavefunction $\Phi(x)$ at the point x_0 can be written symbolically as

$$\frac{d\Phi(x_0)}{dx} = \frac{\Phi(x_0 + dx) - \Phi(x_0)}{dx} \quad (4)$$

where in our code, dx will be determined by the range of x-values and the number of x-values used in defining x_{array} .

Create a function called $dfdx$ that takes an array of x-values and an array of function values evaluated at each x-value and returns the first derivative of the function values with respect to x. Then, show that calling this function in sequence can be used to compute the second derivative of an array of function values.

A skeleton of the function looks like this:

```
In [ ]: def dfdx(fx, x):
        ### loop through values of fx and x and use forward finite-differences
        ### to compute the d/dx of fx at each point and store in an array called 'fxp'
        ### make sure to properly handle
        ### the last point where f(x_0 + dx) is not stored

        ### create a numpy array called fxp that has the same length
        ### and data type as the array of function values you are differentiating
        fxp = np.zeros_like(fx)

        ''' your forward finite difference code goes here!'''

        ### once you have computed all fxp values, return the array!
        return fxp
```

And the calling sequence for $dfdx$ to compute the second derivative of a trial wavefunction we defined previously will look like this:

```
In [ ]: ### Call dfdx on Phi to get the first derivative of Phi
        Phi_p = dfdx(Phi, x_array)

        ### Call dfdx on Phi_p to get the second derivative of Phi
        Phi_pp = dfdx(Phi_p, x_array)
```