Computational Physics (PHYS6350)

Lecture 21: Problems in quantum mechanics

- Matrix method for eigenenergies and eigenstates
- Time-dependent Schroedinger equation
- Variational method

Instructor: Volodymyr Vovchenko (vvovchenko@uh.edu)

Course materials: https://github.com/vlvovch/PHYS6350-ComputationalPhysics
Online textbook: https://vovchenko.net/computational-physics/

Finding the eigenenergies

Time-independent Schroedinger equation reads

$$\left[-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}+V(x)\right]\psi(x)=E\psi(x).$$

e.g. in a box of length L with boundary conditions $\psi(-L/2)=\psi(L/2)=0.$

In the case of (an)harmonic oscillator we learned how to use the shooting method to find the eigenenergies by combining a root finder (bisection or secant method) with an ODE integrator (such as RK4).

This involved discretizing the space on a grid.

The problem can also be tackled efficiently by linear algebra methods.

Matrix method for eigenenergies and eigenstates

By discretizing the space into N intervals, we can represent the wave function $\psi(x)$ as an N+1-dimensional vector $\psi=(\psi_0,\dots,\psi_{N+1})$ such that

$$\psi_k = \psi(x_k), \qquad x_k = -L/2 + kdx, \qquad dx = L/N.$$

Due to the boundary conditions we have $\psi_0=\psi_{N+1}=0$, thus we effectively deal with a N−1-dimensional space.

Each operator becomes a $(N-1)\times(N-1)$ matrix. By discretizing d^{2}/dx^{2} by the central difference we get

$$\frac{d^2}{dx^2}\psi_n \approx \frac{\psi_{n+1} - 2\psi_n + \psi_{n-1}}{dx^2}.$$

Therefore, the Hamiltonian has the following matrix representation:

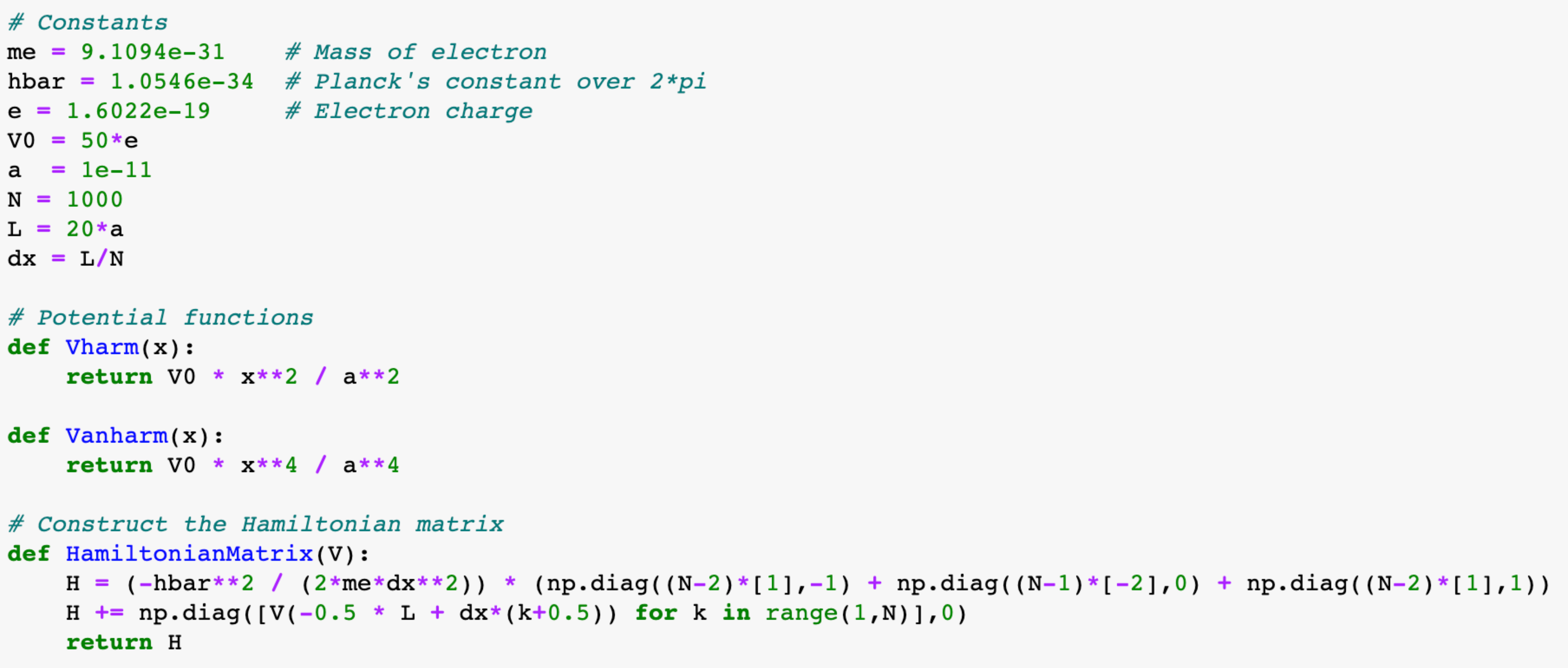
$$H_{nm} = -\frac{\hbar^2}{2m}\left[\delta_{m,n+1}\psi_{n+1} - 2\delta_{m,n}\psi_n + \delta_{m,n-1}\psi_{n-1}\right] + \delta_{m,n}V(x_n),$$

i.e., H is a tridiagonal symmetric matrix.

Therefore, finding the energies and wave function of the system corresponds to the matrix eigenvalue problem for the matrix H.

Let us apply the method to (an)harmonic oscillator we had before.

Matrix method



```python
# Constants
me = 9.1094e-31    # Mass of electron
hbar = 1.0546e-34  # Planck's constant over 2*pi
e = 1.6022e-19     # Electron charge
V0 = 50*e
a  = 1e-11
N = 1000
L = 20*a
dx = L/N

# Potential functions
def Vharm(x):
    return V0 * x**2 / a**2

def Vanharm(x):
    return V0 * x**4 / a**4

# Construct the Hamiltonian matrix
def HamiltonianMatrix(V):
    H = (-hbar**2 / (2*me*dx**2)) * (np.diag((N-2)*[1],-1) + np.diag((N-1)*[-2],0) + np.diag((N-2)*[1],1))
    H += np.diag([V(-0.5 * L + dx*(k+0.5)) for k in range(1,N)],0)
    return H
```

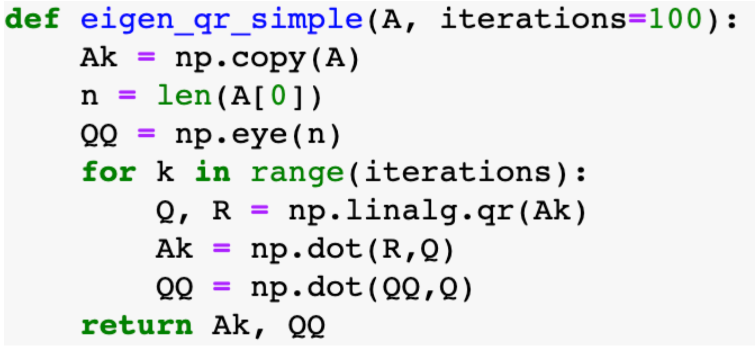
Matrix method for harmonic oscillator

We can use QR decomposition to solve the eigenvalue problem

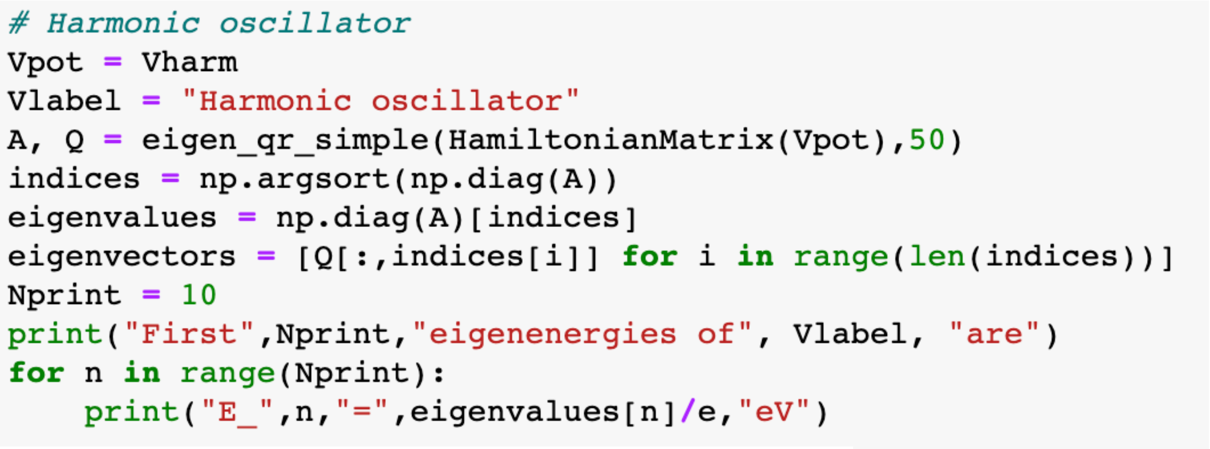
Since our matrix is real symmetric, we can use a straightforward implementation of the QR algorithm. We thus expect to obtain a representation of **H** in the form

$$H = QT\,A\,Q$$

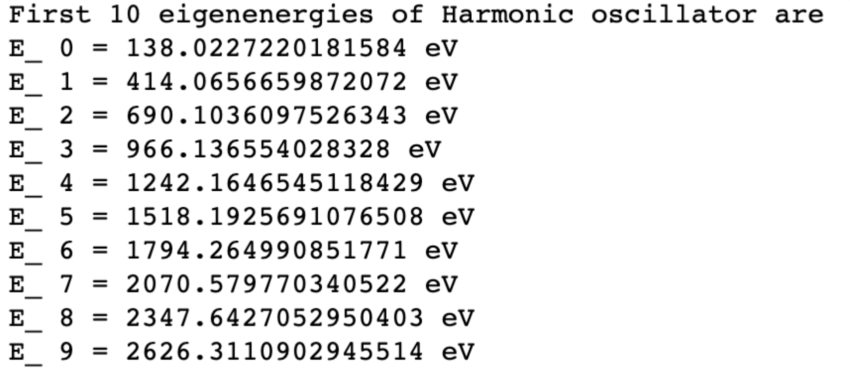
where **A** is diagonal and contains the energies, while **Q** is orthogonal and has eigenvectors (wave functions) in its columns.



```python
def eigen_qr_simple(A, iterations=100):
    Ak = np.copy(A)
    n = len(A[0])
    QQ = np.eye(n)
    for k in range(iterations):
        Q, R = np.linalg.qr(Ak)
        Ak = np.dot(R,Q)
        QQ = np.dot(QQ,Q)
    return Ak, QQ
```



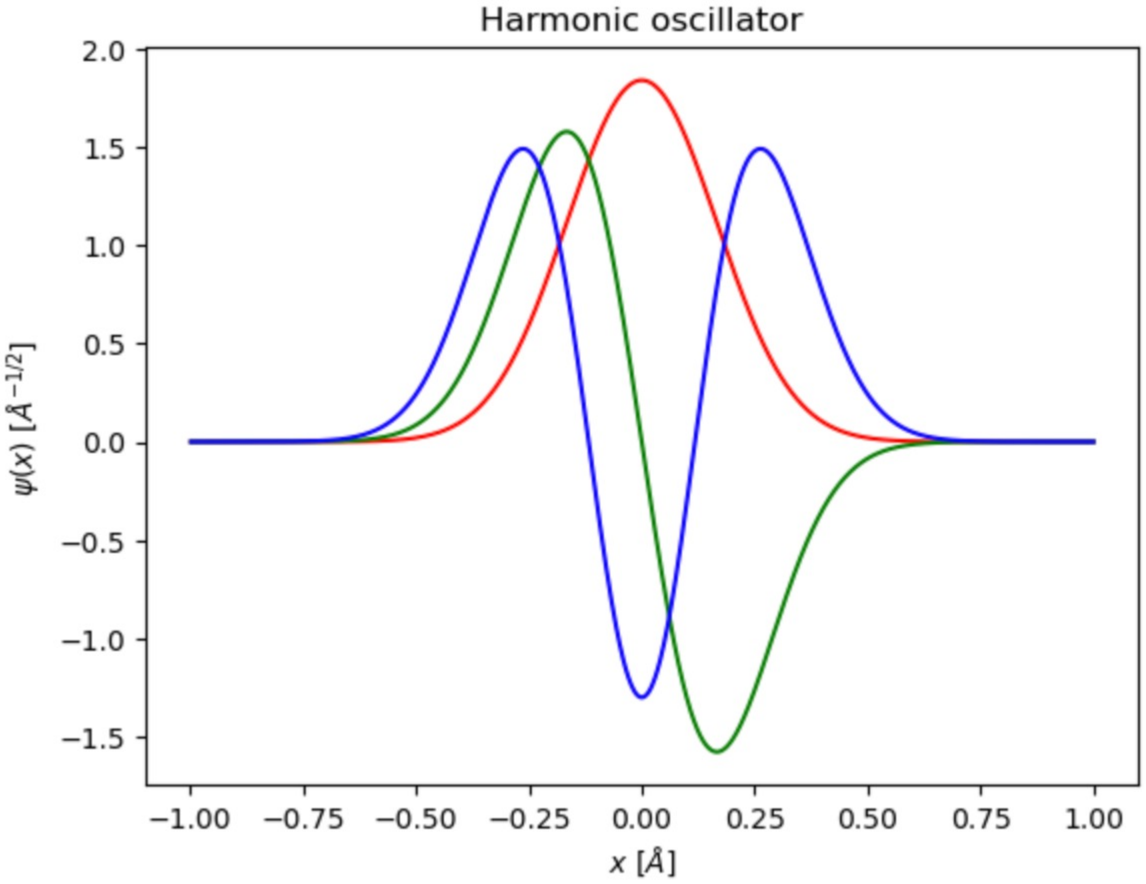
```python
# Harmonic oscillator
Vpot = Vharm
Vlabel = "Harmonic oscillator"
A, Q = eigen_qr_simple(HamiltonianMatrix(Vpot),50)
indices = np.argsort(np.diag(A))
eigenvalues = np.diag(A)[indices]
eigenvectors = [Q[:,indices[i]] for i in range(len(indices))]
Nprint = 10
print("First",Nprint,"eigenenergies of", Vlabel, "are")
for n in range(Nprint):
    print("E_",n,"=",eigenvalues[n]/e,"eV")
```



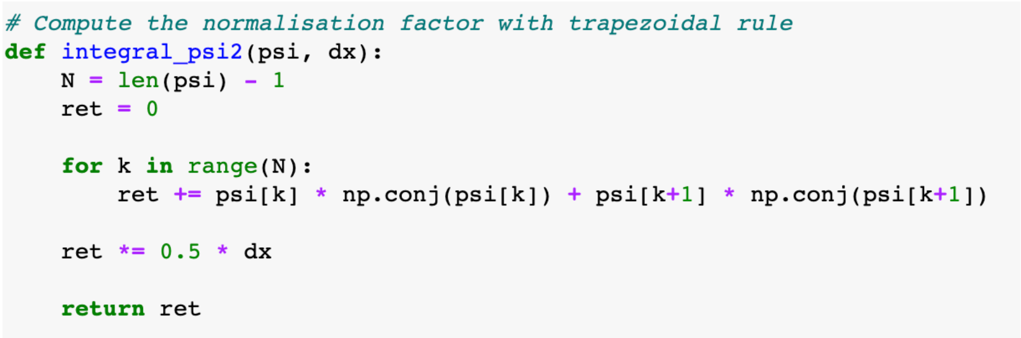
```
First 10 eigenenergies of Harmonic oscillator are
E_ 0 = 138.0227220181584 eV
E_ 1 = 414.0656659872072 eV
E_ 2 = 690.1036097526343 eV
E_ 3 = 966.136554028328 eV
E_ 4 = 1242.1646545118429 eV
E_ 5 = 1518.1925691076508 eV
E_ 6 = 1794.264990851771 eV
E_ 7 = 2070.579770340522 eV
E_ 8 = 2347.6427052950403 eV
E_ 9 = 2626.3110902945514 eV
```

Matrix method for harmonic oscillator

Eigenstates are encoded in the columns of matrix Q

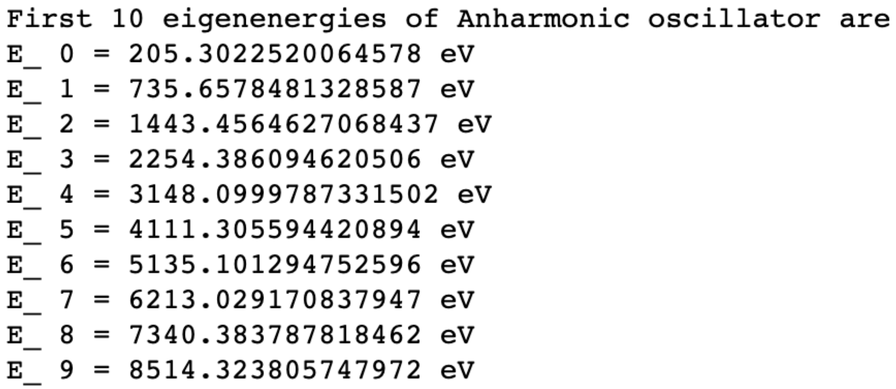


Normalization

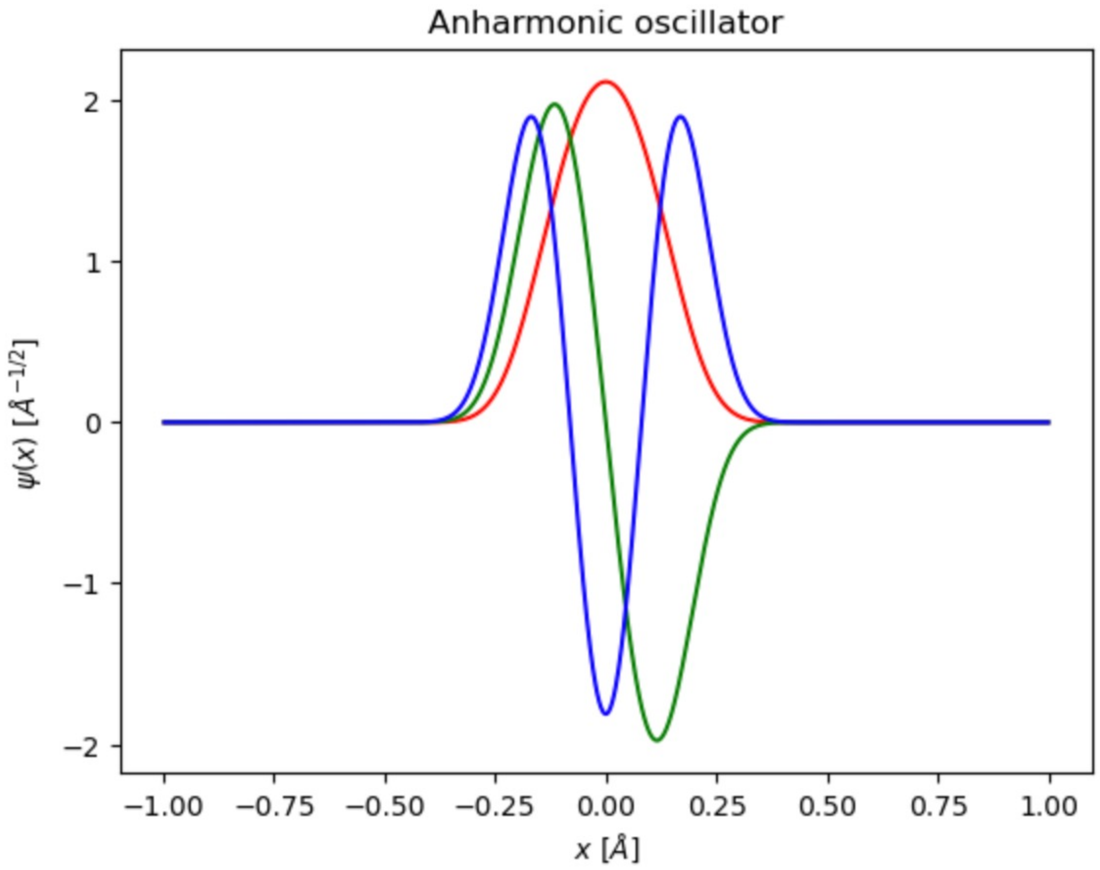


```python
# Compute the normalisation factor with trapezoidal rule
def integral_psi2(psi, dx):
    N = len(psi) - 1
    ret = 0

    for k in range(N):
        ret += psi[k] * np.conj(psi[k]) + psi[k+1] * np.conj(psi[k+1])

    ret *= 0.5 * dx

    return ret
```

Matrix method for anharmonic oscillator

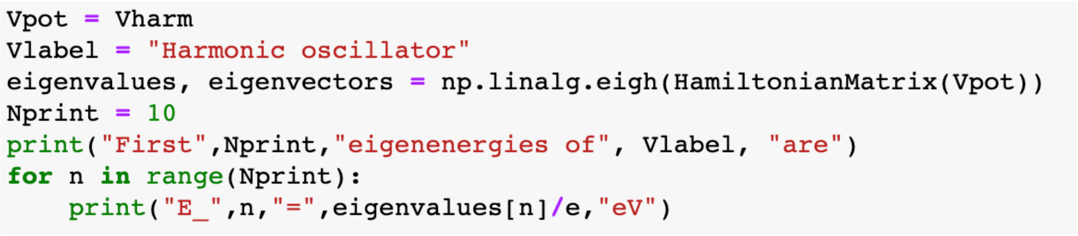


```
First 10 eigenenergies of Anharmonic oscillator are
E_ 0 = 205.3022520064578 eV
E_ 1 = 735.6578481328587 eV
E_ 2 = 1443.4564627068437 eV
E_ 3 = 2254.386094620506 eV
E_ 4 = 3148.0999787331502 eV
E_ 5 = 4111.305594420894 eV
E_ 6 = 5135.101294752596 eV
E_ 7 = 6213.029170837947 eV
E_ 8 = 7340.383787818462 eV
E_ 9 = 8514.323805747972 eV
```

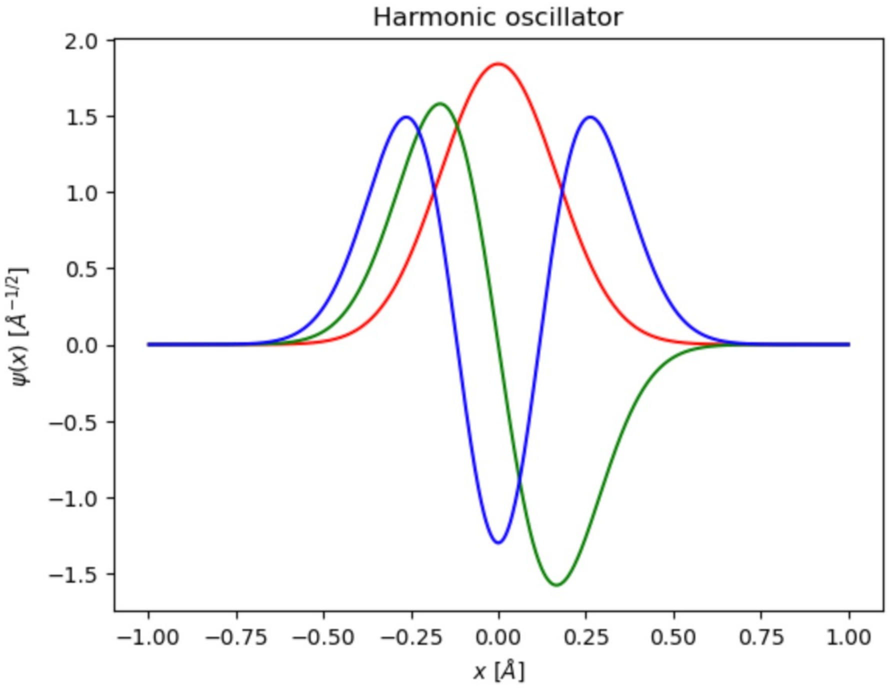


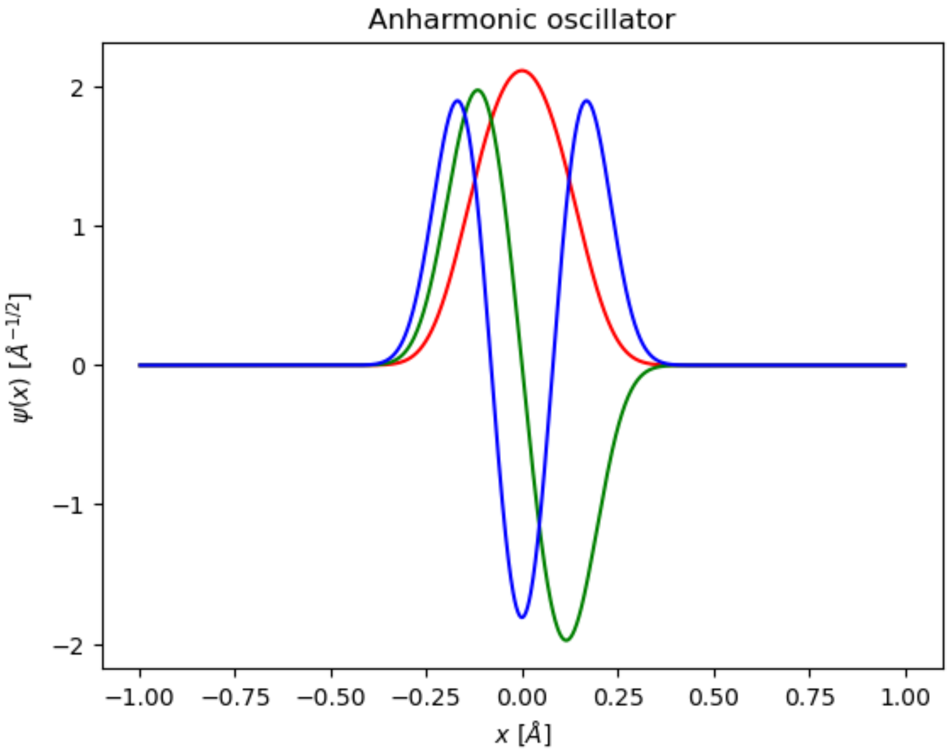
Matrix method

One can also use efficient implementations of the eigenvalue problem in numpy



```
Vpot = Vharm
Vlabel = "Harmonic oscillator"
eigenvalues, eigenvectors = np.linalg.eigh(HamiltonianMatrix(Vpot))
Nprint = 10
print("First",Nprint,"eigenenergies of", Vlabel, "are")
for n in range(Nprint):
    print("E_",n,"=",eigenvalues[n]/e,"eV")
```





Time-dependent Schroedinger equation

The time-dependent Schrödinger equation reads

$$\hat{H}\psi = i\hbar\frac{\partial\psi}{\partial t} \, .$$

e.g., for a free particle it reads

$$-\frac{\hbar^2}{2m}\frac{\partial^2\psi}{\partial x^2} = i\hbar\frac{\partial\psi}{\partial t} \, .$$

Formal solution can be written as

$$\psi(t) = e^{-\frac{it}{\hbar}\hat{H}}\psi(0),$$

with

$$\hat{U}(t) = e^{-\frac{it}{\hbar}\hat{H}}$$

being the time evolution operator. It is a unitary operator:

$$U^{\dagger}U = \hat{I}$$

Therefore, the norm of the wave function is conserved

$$|\psi|^2 = const.$$

Time integration

We can study time evolution by succesively applying approximate $\hat{U}(\Delta t)$ over small time intervals.

- FTCS scheme

  $$\hat{U}(\Delta t) = e^{-\frac{i\Delta t}{\hbar}\hat{H}} \approx 1 - \frac{i\Delta t}{\hbar}\hat{H}$$

  Such an operator is not unitary since

  $$U^{\dagger}(\Delta t) = 1 + \frac{i\Delta t}{\hbar}\hat{H} \neq \hat{U}(\Delta t).$$

  If U is applied to an energy eigenstate ψ_l, such that $\hat{H}\psi_l = E_k\psi_l$, we get

  $$\psi_l(N\Delta t) = \lambda_l^N \psi_l(0),$$

  where $\lambda_l = \left[1 - \frac{i\Delta t E_l}{\hbar}\right]$. Since $|\lambda_l| > 1$, the method is unstable.
- Implicit scheme

We apply the approximation of the FTCS scheme to the inverse of $\hat{U}(\Delta t)$. This implies

$$\hat{U}(\Delta t) = e^{-\frac{i\Delta t}{\hbar}\hat{H}} \approx \frac{1}{1 + \frac{i\Delta t}{\hbar}\hat{H}}.$$

The operator is also non-unitary. The method is stable because

$$|\lambda_l| = \left|\frac{1}{1 + \frac{i\Delta t}{\hbar}E_L}\right| < 1,$$

but the method does not conserve the norm of the wave function.

Time integration: Crank-Nicholson scheme

explicit

$$\hat{U}(\Delta t) = e^{-\frac{i\Delta t}{\hbar}\hat{H}} \approx 1 - \frac{i\Delta t}{\hbar}\hat{H}$$

implicit

$$\hat{U}(\Delta t) = e^{-\frac{i\Delta t}{\hbar}\hat{H}} \approx \frac{1}{1 + \frac{i\Delta t}{\hbar}\hat{H}}.$$

- Crank–Nicholson scheme

Crank–Nicholson scheme takes the combination of FTCS and implicit schemes.
This corresponds to a rational approximation of $\widehat{U}(\Delta t)$:

$$\hat{U}(\Delta t) = e^{-\frac{i\Delta t}{\hbar}\hat{H}} \approx \frac{1 - \frac{i\Delta t}{\hbar}\hat{H}}{1 + \frac{i\Delta t}{\hbar}\hat{H}}.$$

This operator is unitary (for a Hermitian Ĥ):

$$U^{\dagger}U = \hat{I}$$

and conserves the norm of the wave function.

Free particle in a box

Let us consider the particle in a box of length L. We thus have boundary conditions: $\psi(0) = \psi(L) = 0.$

Given some initial wave function ψ(x), we can numerically integrate the Schrödinger equation to study the time evolution of the wave function. Let us write the equation in the form:

$$\frac{\partial \psi}{\partial t} = \frac{i\hbar}{2m}\frac{\partial^2 \psi}{\partial x^2}.$$

We can now use the central difference approximation for the spatial derivative:

This gives us discretized wave function in coordinate space, in full analogy to the heat equation that we studied before. The only difference is that now we are dealing with complex-valued functions.

- FTCS scheme

$$\psi_k^{n+1} = \psi_k^n + h\frac{i\hbar}{2ma^2}(\psi_{k+1}^n - 2\psi_k^n + \psi_{k-1}^n).$$

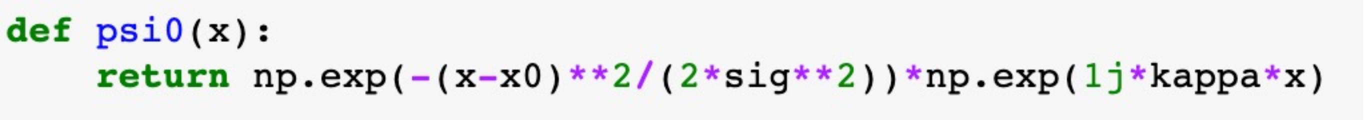
- Implicit scheme

$$\psi_k^{n+1} = \psi_k^n + h\frac{i\hbar}{2ma^2}(\psi_{k+1}^{n+1} - 2\psi_k^{n+1} + \psi_{k-1}^{n+1}).$$

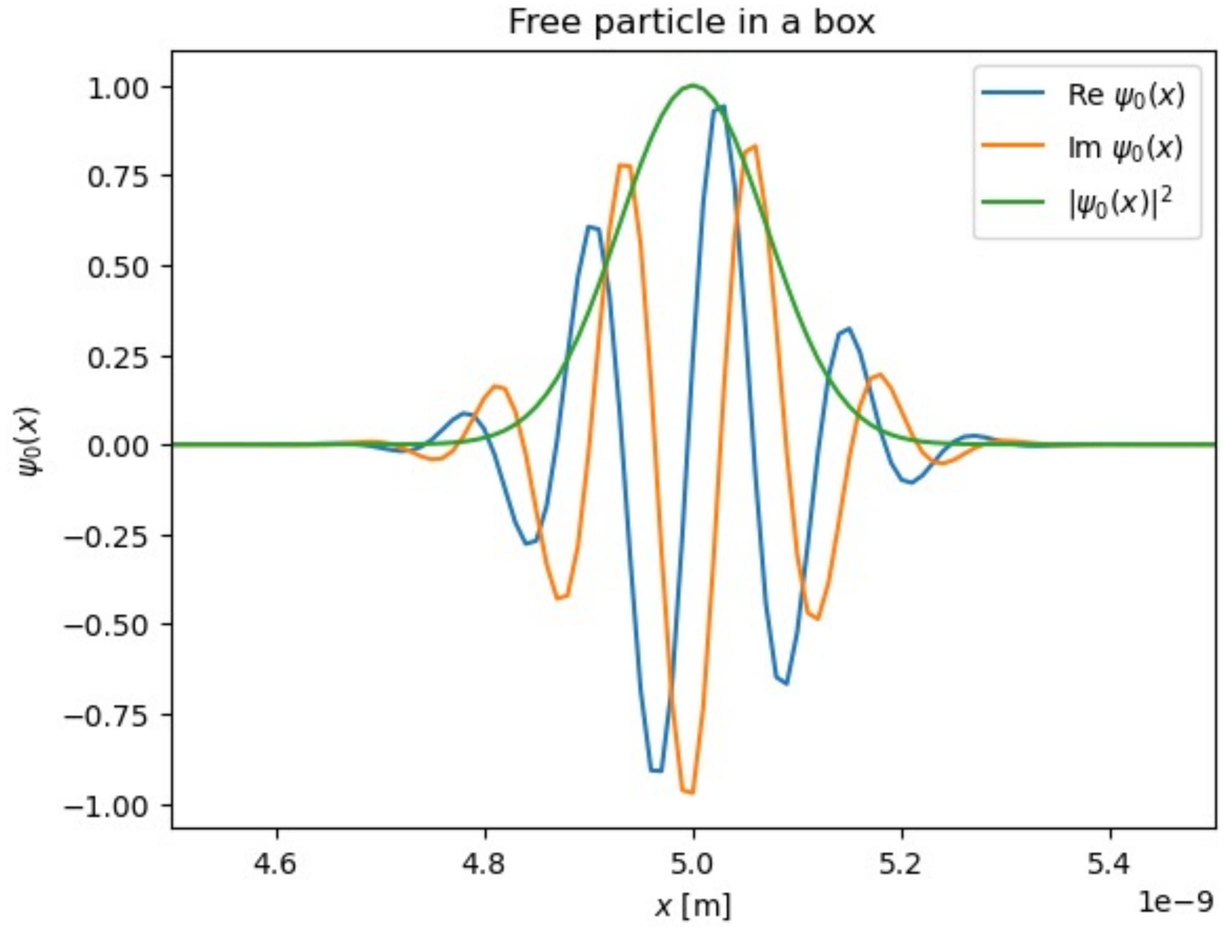
- Crank-Nicholson scheme

$$\psi_k^{n+1} = \psi_k^n + \frac{h}{2}\frac{i\hbar}{2ma^2}(\psi_{k+1}^n - 2\psi_k^n + \psi_{k-1}^n) + \frac{h}{2}\frac{i\hbar}{2ma^2}(\psi_{k+1}^{n+1} - 2\psi_k^{n+1} + \psi_{k-1}^{n+1}).$$

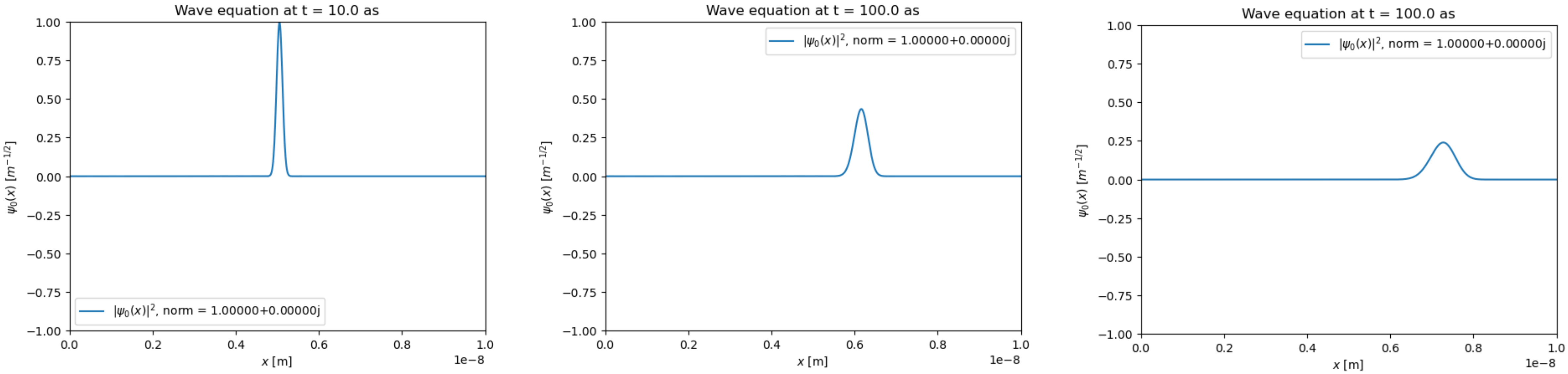
Initial wave function: Gaussian wave packet



```
def psi0(x):
    return np.exp(-(x-x0)**2/(2*sig**2))*np.exp(1j*kappa*x)
```



Time evolution



Expanding wave packet (towards plane wave), norm is conserved

Variational methods

We are often interested in the ground-state energy of the system

This is the lowest energy solution of the time-independent Schroedinger equation:

$$\widehat{H}\psi(\boldsymbol{r}) = E\psi(\boldsymbol{r})$$

For many systems it is challenging to solve this problem explicitly.

Variational method involves the use of trial wave functions $\psi_{trial}(\boldsymbol{r})$

The average energy computed using the trial wave function sets an upper bound of the ground-state energy

$$\boldsymbol{E}[\psi_{trial}] = \frac{\langle\psi_{trial}|\hat{H}|\psi_{trial}\rangle}{\langle\psi_{trial}|\psi_{trial}\rangle} \geq \boldsymbol{E}_0.$$

This is because we can decompose $\psi_{trial}(\boldsymbol{r})$ into orthogonal basis functions of the Hamiltonian operator

$$\psi_{trial} = \sum_n c_n \psi_n \qquad \langle\psi_n|\psi_m\rangle = \delta_{nm} \qquad \widehat{H}\psi_n = En\psi_n$$

Variational methods: Hydrogen Atom

Let us take the hydrogen atom (in dimensionless form)

$$\hat{H} = -\frac{1}{2}\nabla^2 - \frac{1}{r}$$

Length scale:

$$a_0 = \frac{4\pi\varepsilon_0\hbar^2}{\mu e^2}$$

Bohr radius

Energy scale:

$$E_0 = \frac{\hbar^2}{\mu a_0^2} = -27.2\ \mathrm{eV}$$

Trial wave function (unnormalized)

$$\psi_\alpha(r) = e^{-\alpha r}$$

Kinetic energy term:

$$T = -\frac{1}{2}\nabla^2\psi_\alpha(r) = -\frac{1}{2}\alpha^2 e^{-\alpha r} + \frac{\alpha}{r}e^{-\alpha r}$$

Expectation:

- Exact solution for $\alpha = 1$
- Ground state energy $E_{GS} = -0.5$ (13.6 eV)

Potential energy term:

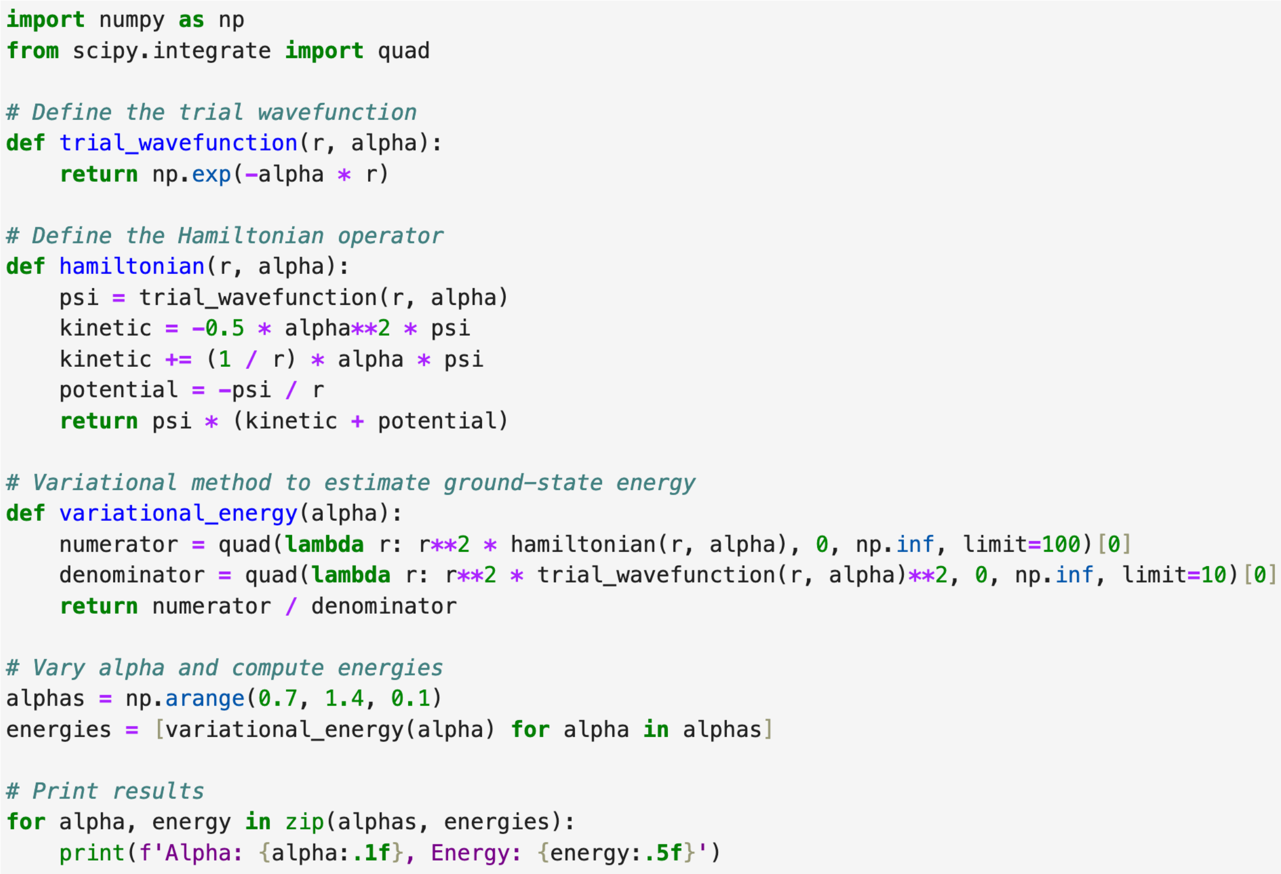
$$V = -\frac{1}{r}\psi_\alpha(r) = -\frac{1}{r}e^{-\alpha r}$$

Average energy:

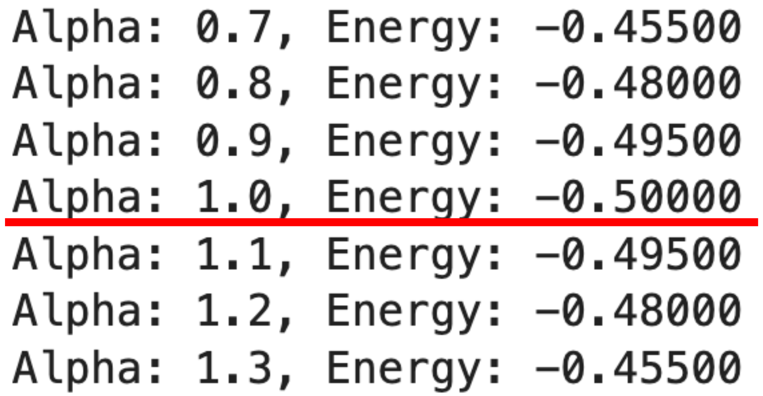
$$\langle E(\alpha)\rangle = \frac{\int_0^\infty r^2\psi_\alpha(r)\widehat{H}\psi_\alpha(r)}{\int_0^\infty r^2\psi_\alpha(r)\,\psi_\alpha(r)}$$

Variational methods: Hydrogen Atom

Let us vary alpha from 0.7 to 1.3



```python
import numpy as np
from scipy.integrate import quad

# Define the trial wavefunction
def trial_wavefunction(r, alpha):
    return np.exp(-alpha * r)

# Define the Hamiltonian operator
def hamiltonian(r, alpha):
    psi = trial_wavefunction(r, alpha)
    kinetic = -0.5 * alpha**2 * psi
    kinetic += (1 / r) * alpha * psi
    potential = -psi / r
    return psi * (kinetic + potential)

# Variational method to estimate ground-state energy
def variational_energy(alpha):
    numerator = quad(lambda r: r**2 * hamiltonian(r, alpha), 0, np.inf, limit=100)[0]
    denominator = quad(lambda r: r**2 * trial_wavefunction(r, alpha)**2, 0, np.inf, limit=10)[0]
    return numerator / denominator

# Vary alpha and compute energies
alphas = np.arange(0.7, 1.4, 0.1)
energies = [variational_energy(alpha) for alpha in alphas]

# Print results
for alpha, energy in zip(alphas, energies):
    print(f'Alpha: {alpha:.1f}, Energy: {energy:.5f}')
```



```
Alpha: 0.7, Energy: -0.45500
Alpha: 0.8, Energy: -0.48000
Alpha: 0.9, Energy: -0.49500
Alpha: 1.0, Energy: -0.50000
Alpha: 1.1, Energy: -0.49500
Alpha: 1.2, Energy: -0.48000
Alpha: 1.3, Energy: -0.45500
```

In eV: -0.5 * 27.2 eV = -13.6 eV

Variational Monte Carlo

In practice, computing the integral explicitly can be difficult, especially when we deal with multi-particle system

In **Variational Monte Carlo**, the corresponding integrals are computed with Monte Carlo techniques

First, rewrite the expectation value for the energy as

$$E(a) = \frac{\langle \Psi(a)|\mathcal{H}|\Psi(a)\rangle}{\langle \Psi(a)|\Psi(a)\rangle} = \frac{\int |\Psi(X,a)|^2 \frac{\mathcal{H}\Psi(X,a)}{\Psi(X,a)}\, dX}{\int |\Psi(X,a)|^2\, dX}.$$

We can interpret $P(X;\alpha) = \frac{|\Psi(X,a)|^2}{\int |\Psi(X,a)|^2\, dX}$ as probability distribution function

If we can sample X from $P(X;\alpha)$, the expectation value of the energy $E(\alpha)$ is just the mean of the so-called local energy $E_{\text{loc}}(X;\alpha)$

$$E_{\text{loc}}(X;\alpha) = \frac{\mathcal{H}\Psi(X,a)}{\Psi(X,a)}$$

Sampling from $P(X;\alpha)$ can be achieved through a variety of methods, in the most general case through importance sampling and Metropolis-Hastings algorithm

Variational Monte Carlo: Hydrogen Atom

Let us turn back to the Hydrogen Atom. Using the same trial wave function, the local energy reads

$$E_{local}(r) = -\frac{1}{2}\alpha^2 + \frac{\alpha}{r} - \frac{1}{r}.$$

$$\psi_\alpha(r) = e^{-\alpha r}$$

The probability distribution for the radial coordinate reads

$$P_r(r;\alpha) \propto r^2 \exp(-2\alpha r).$$

This is a partial case of the Gamma distribution with $k = 3$ and scale factor $1/(2\alpha)$.

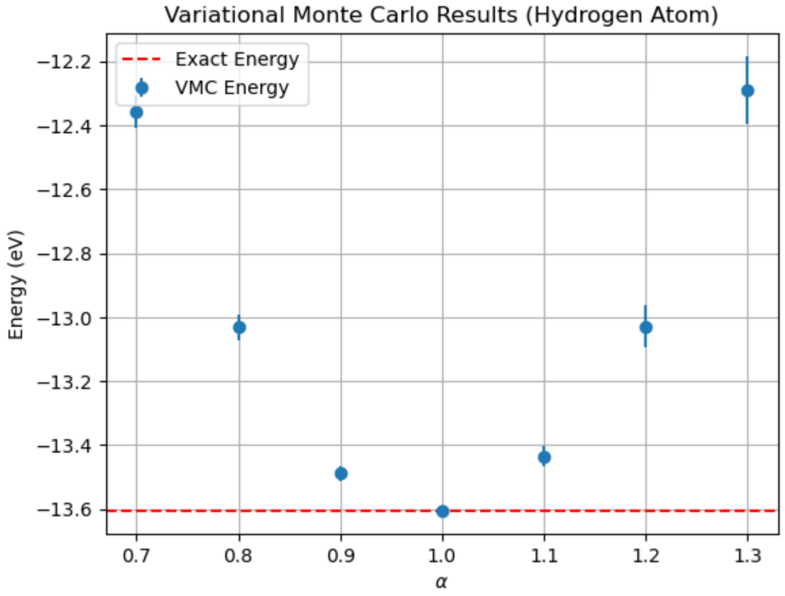
Algorithm:

For each value of alpha:

1. Sample r from the Gamma distribution
2. Compute the value of the local energy E_{local}(alpha)
3. Do it many times and compute the average

Variational Monte Carlo: Hydrogen Atom

```python
# Define the local energy
def local_energy(r, alpha):
    kinetic = -0.5 * alpha**2
    kinetic += (1 / r) * alpha
    potential = -1 / r
    return kinetic + potential

# Variational Monte Carlo
def variational_monte_carlo(alpha, n_samples):
    samples = np.random.gamma(shape=3, scale=1/(2.*alpha), size=n_samples)
    local_energies = [local_energy(r, alpha) for r in samples]
    return np.mean(local_energies), np.var(local_energies), np.std(local_energies) / np.sqrt(n_samples)

# Vary alpha and compute energies
alphas = np.arange(0.7, 1.4, 0.1)
n_samples = 10000

energies = []
errors = []
```



```
Alpha: 0.7, Mean Energy: -0.45414, Variance: 0.03601, Error: 0.00190
Alpha: 0.7999999999999999, Mean Energy: -0.47891, Variance: 0.02227, Error: 0.00149
Alpha: 0.8999999999999999, Mean Energy: -0.49567, Variance: 0.00710, Error: 0.00084
Alpha: 0.9999999999999999, Mean Energy: -0.50000, Variance: 0.00000, Error: 0.00000
Alpha: 1.0999999999999999, Mean Energy: -0.49373, Variance: 0.01236, Error: 0.00111
Alpha: 1.1999999999999997, Mean Energy: -0.47883, Variance: 0.05781, Error: 0.00240
Alpha: 1.2999999999999998, Mean Energy: -0.45162, Variance: 0.15036, Error: 0.00388
```

Variational Monte Carlo: Helium Atom

In the Helium Atom we have two electrons. Their interaction complicates things.

$$\hat{H} = -\frac{1}{2}\nabla_1^2 - \frac{1}{2}\nabla_2^2 - \frac{Z}{r_1} - \frac{Z}{r_2} + \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|},$$

Let us take the trial wave function as a product of Hydrogen Atom wave functions

$$\psi_\alpha(r_1, r_2) = e^{-\alpha(r_1+r_2)}$$

Local energy: $$E_{local}(r_1, r_2) = -\frac{1}{2}\alpha^2(1+1) + \alpha\left(\frac{1}{r_1} + \frac{1}{r_2}\right) - Z\left(\frac{1}{r_1} + \frac{1}{r_2}\right) + \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|}.$$

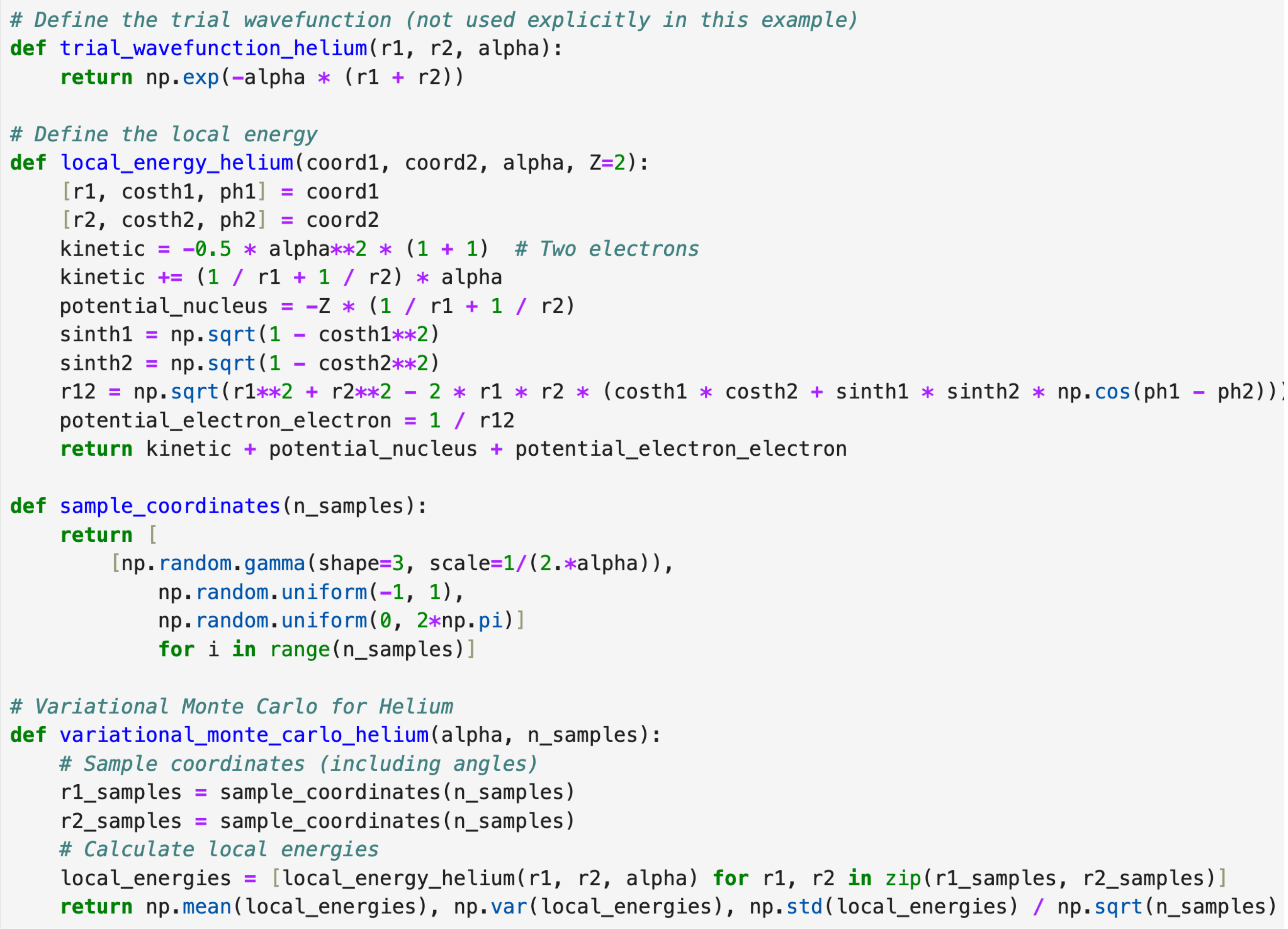
As before, the radial coordinates r_1 and r_2 follow the Gamma distribution with $k = 3$ and scale factor $1/(2\alpha)$.

However, we have an additional factor in the local energy which depends on the spherical angles:

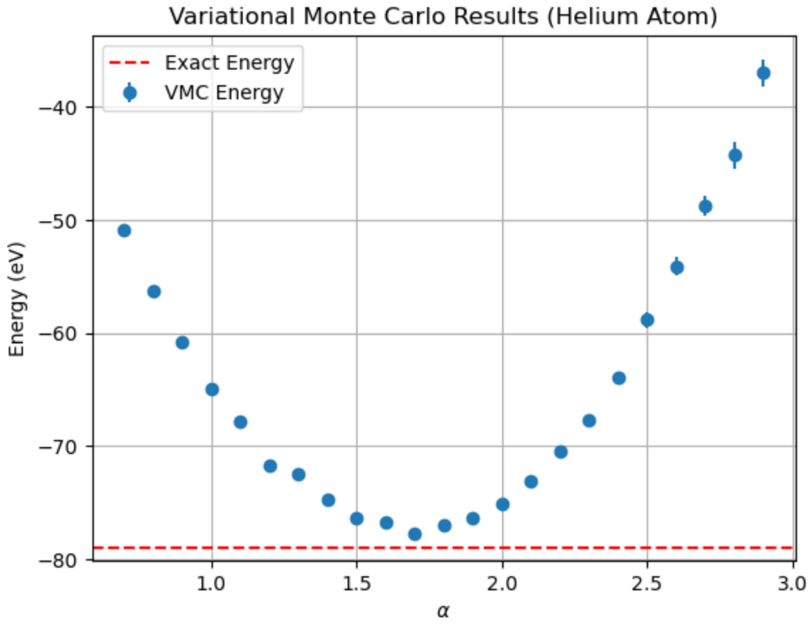
$$|\mathbf{r}_1 - \mathbf{r}_2| = \sqrt{r_1^2 + r_2^2 - 2r_1r_2\left(\cos\theta_1\cos\theta_2 + \sin\theta_1\sin\theta_2\cos(\phi_1 - \phi_2)\right)}.$$

We have to sample $\theta_{1,2}$ and $\phi_{1,2}$ from a unit sphere isotropically

Variational Monte Carlo: Helium Atom



```python
# Define the trial wavefunction (not used explicitly in this example)
def trial_wavefunction_helium(r1, r2, alpha):
    return np.exp(-alpha * (r1 + r2))

# Define the local energy
def local_energy_helium(coord1, coord2, alpha, Z=2):
    [r1, costh1, ph1] = coord1
    [r2, costh2, ph2] = coord2
    kinetic = -0.5 * alpha**2 * (1 + 1)  # Two electrons
    kinetic += (1 / r1 + 1 / r2) * alpha
    potential_nucleus = -Z * (1 / r1 + 1 / r2)
    sinth1 = np.sqrt(1 - costh1**2)
    sinth2 = np.sqrt(1 - costh2**2)
    r12 = np.sqrt(r1**2 + r2**2 - 2 * r1 * r2 * (costh1 * costh2 + sinth1 * sinth2 * np.cos(ph1 - ph2))
    potential_electron_electron = 1 / r12
    return kinetic + potential_nucleus + potential_electron_electron

def sample_coordinates(n_samples):
    return [
        [np.random.gamma(shape=3, scale=1/(2.*alpha)),
           np.random.uniform(-1, 1),
           np.random.uniform(0, 2*np.pi)]
           for i in range(n_samples)]

# Variational Monte Carlo for Helium
def variational_monte_carlo_helium(alpha, n_samples):
    # Sample coordinates (including angles)
    r1_samples = sample_coordinates(n_samples)
    r2_samples = sample_coordinates(n_samples)
    # Calculate local energies
    local_energies = [local_energy_helium(r1, r2, alpha) for r1, r2 in zip(r1_samples, r2_samples)]
    return np.mean(local_energies), np.var(local_energies), np.std(local_energies) / np.sqrt(n_samples)
```



The lowest we get (-77.7 eV) is not too far from the true value (-79.0 eV)!