Hartree-Fock implementation in Python

Timo Horstschäfer, Marcel Langer

***Abstract*—This report will present an approach to numerically solve the Hartree-Fock equation for simple cases and present the results of this approach for the ^{4}He nucleus.**

I. Hartree-Fock approach

The Hartree-Fock approach attempts to simplify the full nuclear Hamiltonian, which includes the two-body interaction, to a single-particle Hamiltonian by applying the variational principle. That way, wave functions and energies for nuclei can be calculated.

A. General case

The general nuclear Hamiltonian in coordinate notation is given by [1] as

$$\hat{H} = \sum_i \frac{-\hbar^2}{2m}\Delta_i + \frac{1}{2}\sum_{i\neq j} V(\boldsymbol{r}_i, \boldsymbol{r}_j).$$

The Hartree-Fock approximation now assumes that the nuclear wave function Φ can be written as a Slater determinant of single-particle wave functions ϕ_i. Calculating $\langle\Phi|\hat{H}|\Phi\rangle$ with this wave function yields

$$\langle\Phi|\hat{H}|\Phi\rangle = \sum_i \langle i|t_i|i\rangle + \frac{1}{2}\sum_{i,j}\langle ij|V|\left(|ij\rangle - |ji\rangle\right).$$

Minimizing this expression while enforcing normalisation for ϕ_i and assuming that V is independent of the wave functions results in the Hartree-Fock equations

$$\begin{aligned}\varepsilon_i\phi_i(\boldsymbol{r}_i) &= \frac{-\hbar^2}{2m}\Delta_i\phi_i(\boldsymbol{r}) \\ &+\sum_j \int \mathrm{d}\boldsymbol{r}' V(\boldsymbol{r},\boldsymbol{r}')\phi_j^*(\boldsymbol{r}')\left[\phi_j(\boldsymbol{r}')\phi_i(\boldsymbol{r}) - \phi_j(\boldsymbol{r})\phi_i(\boldsymbol{r}')\right].\end{aligned} \tag{1}$$

Introducing the densities

$$\rho(\boldsymbol{r}) = \sum_i \phi_i^*(\boldsymbol{r})\phi_i(\boldsymbol{r})$$
$$\rho(\boldsymbol{r},\boldsymbol{r}') = \sum_i \phi_i^*(\boldsymbol{r})\phi_i(\boldsymbol{r}')$$

and the potentials

$$\Gamma_H(\boldsymbol{r}) = \int \mathrm{d}\boldsymbol{r}' V(\boldsymbol{r},\boldsymbol{r}')\rho(\boldsymbol{r}')$$
$$\Gamma_{Ex}(\boldsymbol{r},\boldsymbol{r}') = -V(\boldsymbol{r},\boldsymbol{r}')\rho(\boldsymbol{r},\boldsymbol{r}')$$

lets us rewrite eq. 1 as a non-local Schrödinger equation[2]:

$$\left[\frac{-\hbar^2}{2m}\Delta_i + \Gamma_H(\boldsymbol{r})\right]\phi_i(\boldsymbol{r}) + \int \mathrm{d}\boldsymbol{r}'\Gamma_{Ex}(\boldsymbol{r},\boldsymbol{r}')\phi_i(\boldsymbol{r}') = \varepsilon_i\phi_i(\boldsymbol{r}). \tag{2}$$

B. Solutions for a density-dependent force

In order to find solutions to eq. 2, we introduce an effective potential of the shape

$$V(\boldsymbol{r},\boldsymbol{r}') = a\delta(\boldsymbol{r}-\boldsymbol{r}') + b\rho\left(\frac{\boldsymbol{r}+\boldsymbol{r}'}{2}\right)\delta(\boldsymbol{r}-\boldsymbol{r}'). \tag{3}$$

Due to the δ functions, this potential reduces the non-local term to a local one, which is similar to $\Gamma_H(\boldsymbol{r})$ and will be neglected from now on. Substituting V in eq. 2 and performing the variation yields the final one-particle equation

$$\begin{aligned}&\left[\frac{-\hbar^2}{2m}\Delta_i + 2a\rho(\boldsymbol{r}) + 3b\rho(\boldsymbol{r})^2\right]\phi_i(\boldsymbol{r}) \\ &= \left[\frac{-\hbar^2}{2m}\Delta_i + u(\boldsymbol{r})\right]\phi_i(\boldsymbol{r}) \\ &= \hat{H}\phi_i(\boldsymbol{r}) \\ &= \varepsilon_i\phi_i(\boldsymbol{r}).\end{aligned} \tag{4}$$

Since $u(\boldsymbol{r})$ depends on the wave function ϕ_i, this equation poses a self-consistency problem. It can be solved with an iterative approach.

C. Iterative solutions

Let $\{\psi_i\}$ be a complete set of (phenomenological) single-particle wave functions. They can be expanded in the basis of the (unknown) eigenfunctions of $\hat{H}$:

$$\psi_i = \sum c_{i,j}\phi_i.$$

Defining the operator $I := e^{-\delta(\hat{H}-\varepsilon_i)}$ and letting it act on ψ_i yields

$$e^{-\delta(\hat{H}-\varepsilon_i)}\psi_i(\boldsymbol{r}) = \sum_j c_{i,j}e^{-\delta(\varepsilon_j-\varepsilon_i)}\phi_j(\boldsymbol{r}).$$

Since the exponential suppresses high-energy ϕ_j, ψ_i is nudged towards the lowest-energy solution. Repeated applications of E will therefore yield arbitrarily close approximations for the lowest energy ϕ_j. Practically, a number of approximations have to be used:

- Since the eigenfunctions of $\hat{H}$ are unknown, the current guesses ψ_i are used to calculate $\hat{H}$ and ε_i at each iteration.
- δ is regarded as infinitesimal which results in $I \approx 1 - \delta(\hat{H}-\varepsilon_i)$.

The algorithm for solving the Hartree-Fock equations therefore becomes:

1) Guess initial wave functions ψ_i,
2) Calculate ρ, $\hat{H}$ and $\langle\psi_i|\hat{H}|\psi_i\rangle = \varepsilon_i$,
3) Let $\psi_i' = \Big(1-\delta(\hat{H}-\varepsilon_i)\Big)\psi_i$,
4) Repeat from step 2 with $\psi = \psi'$ until the changes are sufficiently small.

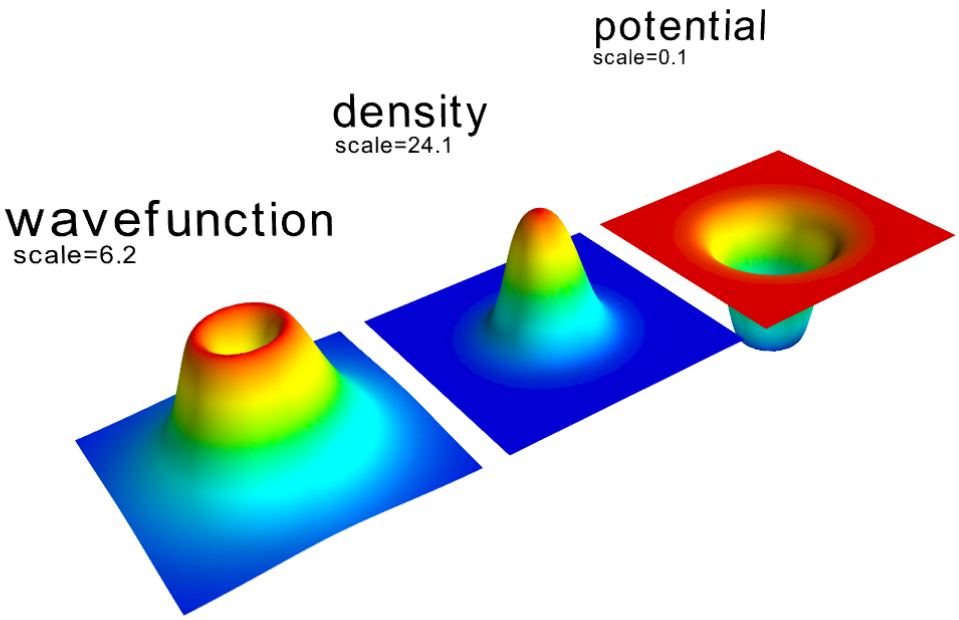

Fig. 1: 3D representation of wavefunction, density and the potential

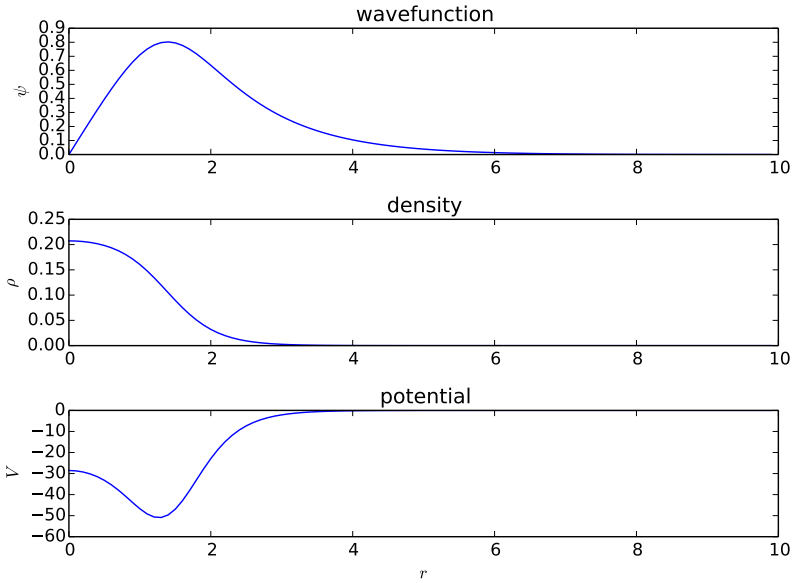

Fig. 2: Same result as 2D graph

II. Implementation

A. How the code works

In this section, we give a short explanation on how the actual implementation to solve the Hartree-Fock equation works.

First, we work in a discrete space, meaning that the wavefunction is a vector of length N. The same holds for the density and the potential. Second, the implementation is done for the ^{4}He nucleus, where we assume that the wave functions of all nucleons are the same. The Coulomb interaction is also neglected. Since ^{4}He is a spherical nucleus, spherical symmetry is assumed, which is the reason why only the radial part of the wave function is examined in detail.

We initialize the wave function with a gaussian of the form

$$f(x, P) = \frac{x}{P} e^{-\left(\frac{x}{P}\right)^2}$$

where P is an arbitrarily chosen starting parameter, which is $P = 1.4$ in our case.

Then, then iteration is started. For simplicity, it will run for a given amount of steps, regardless if the wavefunction becomes static after some iterations.

a) Density: The density is calculated as

$$\rho = 4\frac{\psi^2}{4\pi r^2} = \frac{1}{\pi}\left(\frac{\psi}{r}\right)^2$$

where the factor $4\pi r^2$ is the normalization factor and the factor 4 in the beginning is due to the 4 particles in the ^{4}He.
b) Potential: The potential comes directly from eq. 4 so we have

$$u = 2a\rho + 3b\rho^2$$

where the coefficients are chosen as $a = -408.75$ and $b = 1093.0$, according to the original implementation in [1].
c) Kinetic term: To perform the next step, we first need to calculate the kinetic energy term of the Hamiltonian.
Since it is given as

$$H_{\text{kin}} = -\frac{\hbar^2}{2m}\Delta\psi$$

the main difficulty is to calculate the Laplacian of the wavefunction. The term $\frac{\hbar}{2m}$ can be chosen as a single constant with value $\frac{\hbar}{2m} = 41.4$ in natural units.
The lowest order of the second derivative is the 3-point formula, which reads

$$f''(x_i) = \frac{f(x_{i-1}) - 2f(x_i) + f(x_{i+1})}{h^2}$$

with h referring to the grid spacing.
d) Apply exponential: After the energy is computed, it is easy to act with the exponential

$$e^{-\delta(\hat{H}-\varepsilon)}\psi \approx (1 - \delta(\hat{H} - \varepsilon))\psi$$

Here, $\delta = 0.0001$ is the damping parameter, to keep the path during iteration smooth.
Acting with the Hamiltonian $\hat{H}$ on the wavefunction yields

$$\hat{H}\psi = \left(-\frac{\hbar^2}{2m}\Delta + u\right)\psi = H_{\text{kin}} + u\psi$$

whereas ε is the expectation value of the energy. Thus, with the discrete wave function, we have

$$\begin{aligned}\varepsilon &= \langle\psi|\hat{H}|\psi\rangle \\ &= \sum_i \psi_i \hat{H}_i \psi_i \; h \\ &= h\sum \psi_i(E_{\text{kin},i} + u_i\psi_i)\end{aligned}$$

with h again referring to the grid spacing.
e) Renormalization: Since the operator $(1-\delta(\hat{H}-\varepsilon))$ changes the norm, we have to renormalize, according the the discrete integral

$$\langle\psi|\psi\rangle = \sum_i \psi_i^2 h$$

giving

$$\psi' = \frac{\psi}{\sqrt{h\sum_i \psi_i^2}}$$

III. Visualization

The implementation outputs just a one-dimensional array for the wavefunction, density and the potential. However, a good visualization is helpful to interpret the results.
To do this, we projected the radial solutions onto a 2D grid, where the center of the surface corresponts to the $r = 0$ position. This grid can then can be easily plotted in 3D space as a deformed surface, with the ascension of the surface representing the value of given function.
Since the points on the 2D grid have different distances from the origin than were actually calculated, the result has to be interpolated. We chose a simple linear interpolation between each of the points.
In addition, since the absolute values of the three result vector are much different, each surface is scaled by the function

$$s = \frac{C}{\max(|v_i|)}$$

to allow plotting all three surfaces in a single image. Here, $\boldsymbol{v}$ denotes the one-dimensional result array of the calculations and the value of the scaling constant is $C = 5$.
The resulting surfaces are shown in figure 1.

References

[1] P. D. Stevenson, *Hartree-Fock Approximation in Nuclear Structure: A Primer*, University of Surrey
http://personal.ph.surrey.ac.uk/~phs3ps/unis-notes-yr2005-stevenson.pdf
[2] P. Ring, P. Schuck *The Nuclear Many-Body Problem*. Springer, 1980.

APPENDIX A
SOURCE CODE

Listing 1: hf.py

```
#!/usr/bin/env python
#     (the first line allows execution
    directly from the linux shell)
#
# --- simple hartree-fock simulation,
    basically a port of http://personal.ph
    .surrey.ac.uk/~phs3ps/simple-hf.html
# Author:          M. Langer, T.
    Horstschaefer
# dependencies:  PYTHON v2.7, numpy
# last modified:

# coding=utf-8

import math
import numpy as np

def gaussian(x,parameter):
  return (x/parameter)*np.exp(-(x/
      parameter)**2/2)

hbar = 41.437

def hartreeFock(N, max_iterations, dr,
    initial_parameter, damping,
    potential_f):
        grid = np.arange(0, N*dr, dr)
        density = np.zeros(N)

        # init wf with gaussian
        wavefunction = gaussian(grid,
            initial_parameter)

        for i in range(max_iterations):
          wavefunction /= np.sqrt(dr*sum(
              wavefunction**2))

          # define density with a new, "
              virtual" wf (wf / grid) that
               is nomalised in spherical
              coordinates
          density[1:] = 4 * (wavefunction
              [1:]/grid[1:])**2 / (4*np.pi
              )
          density[0] = 4*( (4*
              wavefunction[1]/3 -
              wavefunction[2]/6) /dr)**2/
              (4*np.pi)

          potential = potential_f(density
              )

          # calculate laplacian on the
              wave function with three-
              point approach, treating the
               edges with two-point
          kinetic = np.empty(N)
          kinetic[0] = hbar*wavefunction
              [0]/(dr**2)
          kinetic[N-1] = -(hbar/(2*dr**2)
              )*(wavefunction[N-2]-
              wavefunction[N-1])
          for j in xrange(1,N-1):
            kinetic[j] = -(hbar/(2*dr**2)
                )*(wavefunction[j-1]-2*
                wavefunction[j]+
                wavefunction[j+1])

          transformed = kinetic +
              potential*wavefunction
          energy = dr * sum(transformed*
              wavefunction)
          wavefunction -= damping * (
              transformed - energy *
              wavefunction)

        return np.array([ grid,
            wavefunction, density,
            potential ])
```

Listing 2: plot.py

```
# Must be run with 'ipython --gui=wx --
    pylab=wx' and then 'run plot.py'
import hf

from matplotlib import pyplot as plt
from scipy.interpolate import interp1d
from mayavi import mlab

import numpy as np
import os


"""
# Run fortran code
os.system("gfortran hf.F90 -o hf && ./hf
    ")

# Read results from fortran code
data = np.genfromtxt("fort.15")
grid = data[:,0]
density = data[:,1]

data = np.genfromtxt("fort.16")
potential = data[:,1]
plt.plot(grid, density)
"""
a = -408.75
b =  1093.0

param = {
    'N': 100,
    'max_iterations': 1000,
    'dr' : 0.1,                 # grid
        spacing
    'initial_parameter': 1.4,
    'damping': 0.0001,

    # HF Potential parameters
    'potential_f': lambda density: 2*a*
        density + 3*b*density**2
}

# Solve Hartree-Fock equation
grid, wavefunction, density, potential =
    hf.hartreeFock(**param)

# MATPLOTLIB

# show 2d plots
plt.figure(1)

plt.subplot(311)
plt.title('wavefunction')
plt.ylabel(r'$\psi$')
plt.plot(grid, wavefunction)

plt.subplot(312)
plt.title('density')
plt.ylabel(r'$\rho$')
plt.plot(grid, density)

plt.subplot(313)
plt.title('potential')
plt.ylabel(r'$V$')
plt.xlabel(r'$r$')
plt.plot(grid, potential)

plt.tight_layout()
plt.show()


# MAYAVI

# Make a 2D grid out of a radial 1D
    function
def Make2DGrid(x, y):
        f = interp1d(x, y, fill_value=0.)
        g_max = max(x)
        n = len(x)

        # evaluate result on 2D grid
        s = np.zeros((n, n))
        for x in xrange(0, n):
                for y in xrange(0, n):
                        s[x,y] = f( np.
                            sqrt( (x - n
                            /2)**2 + (y -
                            n/2)**2 ) *
                            param['dr'] )
        return s



# Make grid to shift the objects later
g_max = max(grid)
x, y = np.mgrid[ -g_max:g_max:param['dr'
    ], -g_max:g_max:param['dr'] ]

# Plot objects shifted on x-axis
mlab.figure(bgcolor=(1,1,1), fgcolor
    =(0,0,0))
shift = g_max+1


scale = lambda y: 5./max(abs(y))

mlab.surf(x - shift,y, Make2DGrid(grid,
    potential),     warp_scale = scale(
    potential))
mlab.surf(x,y,          Make2DGrid(grid,
    density),       warp_scale = scale(
    density))
mlab.surf(x + shift,y, Make2DGrid(grid,
    wavefunction), warp_scale = scale(
```

```
    wavefunction))

# Add description
t_z = 8
ts_scale = 0.5
mlab.text3d(-shift,-g_max, t_z, '
    potential')
mlab.text3d(-shift,-g_max, t_z-1, 'scale
    ={:.1f}'.format(scale(potential)),
    scale=ts_scale)

mlab.text3d(      0,-g_max, t_z, 'density'
    )
mlab.text3d(      0,-g_max, t_z-1, 'scale
    ={:.1f}'.format(scale(density)), scale
    =ts_scale)

mlab.text3d( shift,-g_max, t_z, '
    wavefunction')
mlab.text3d( shift,-g_max, t_z-1, 'scale
    ={:.1f}'.format(scale(wavefunction)),
    scale=ts_scale)
```