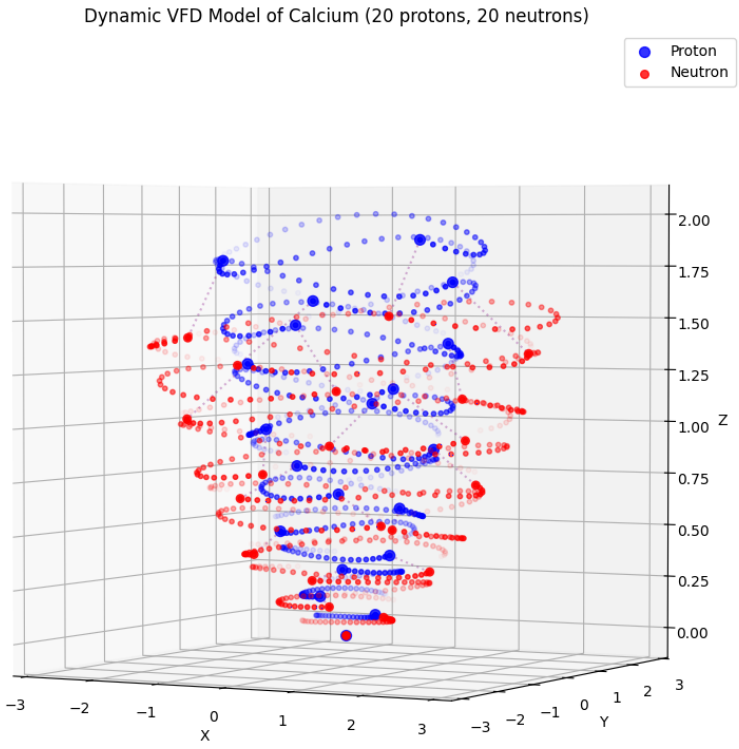
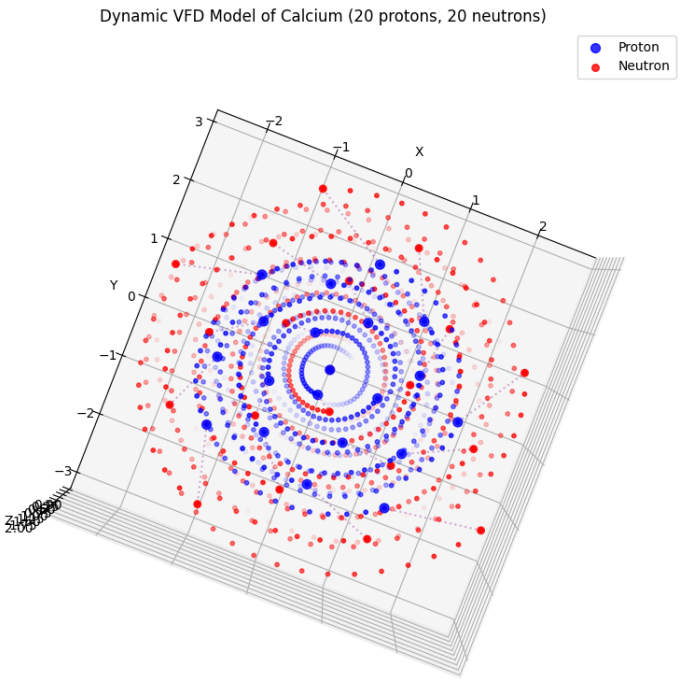
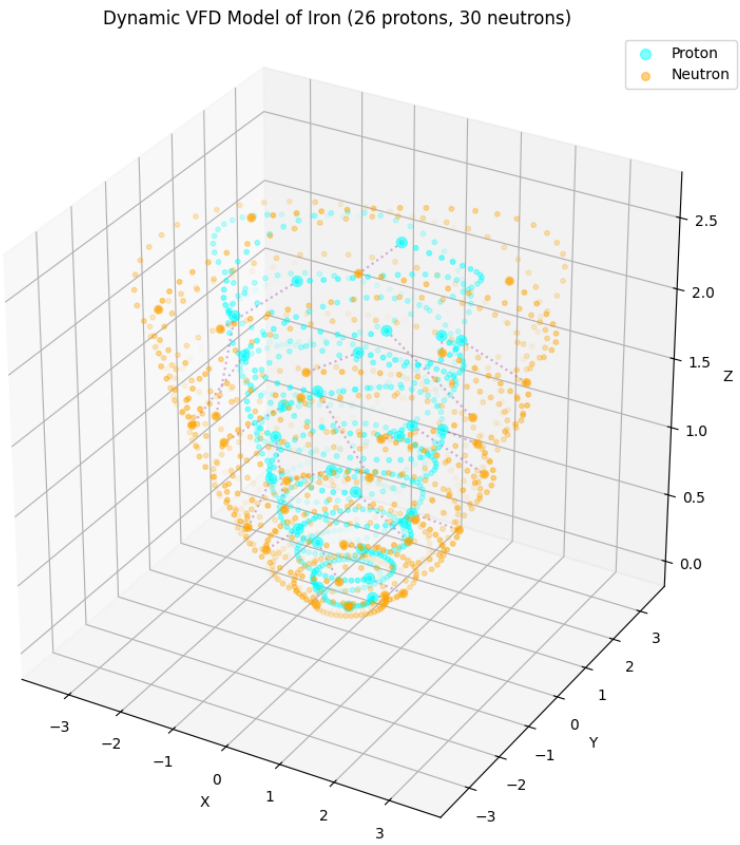
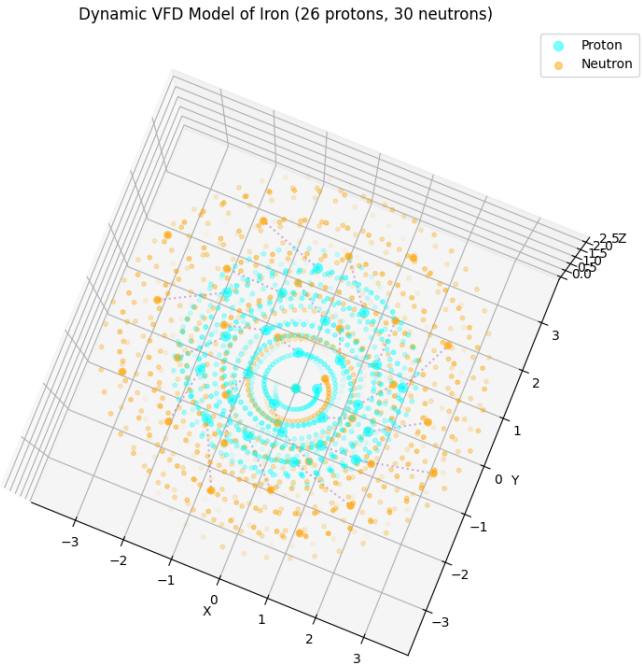
Element Modeling

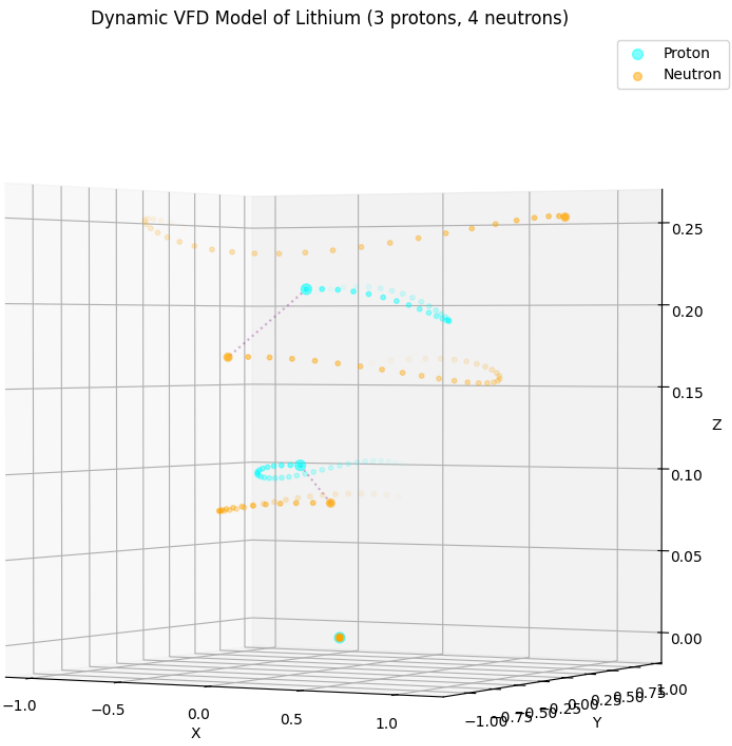
Description

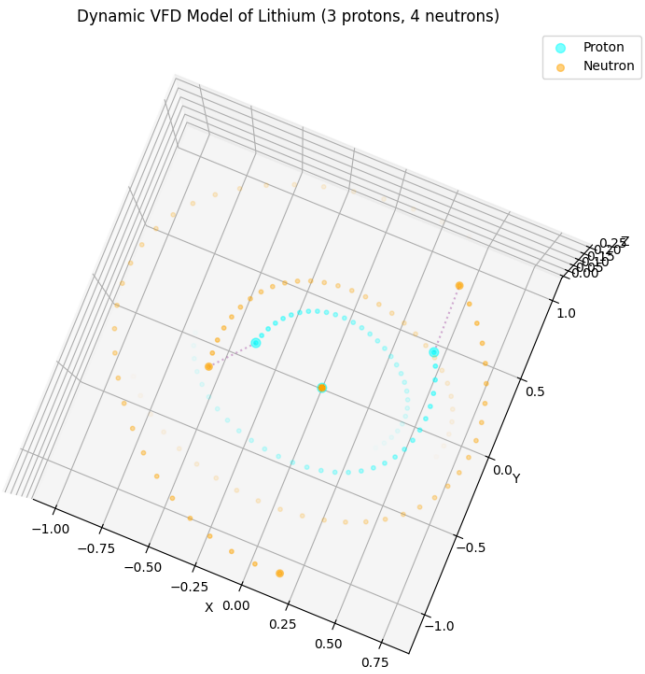



Dynamic VFD Model of Calcium (20 protons, 20 neutrons)
Proton
Neutron
X
Y


Dynamic VFD Model of Iron (26 protons, 30 neutrons)
Proton
Neutron
2.5
2.0
1.5
1.0
0.5
0.0
Z
−3
−2
−1
0
1
2
3
X
Y


Dynamic VFD Model of Iron (26 protons, 30 neutrons)
Proton
Neutron
X
Y
Z


Dynamic VFD Model of Lithium (3 protons, 4 neutrons)
Proton
Neutron
0.25
0.20
0.15
Z
0.10
0.05
0.00
−1.0
−0.5
0.0
0.5
1.0
X
Y



Pythion Code to Simulate the Elements

Copy and Run the code in a Python enviroemt for example c:/ElementSim.py Moscovium

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FuncAnimation
import sys

# Golden ratio constant
golden_ratio = (1 + np.sqrt(5)) / 2

# Define magic numbers for nuclear stability
magic_numbers = [2, 8, 20, 28, 50, 82, 126]

# Lookup table for elements (atomic numbers mapped to proton and neutron count
# Neutron counts are approximate for the most stable isotope of each element.
element_data = {
    "Hydrogen": (1, 0), "Helium": (2, 2), "Lithium": (3, 4), "Beryllium": (4,
    "Carbon": (6, 6), "Nitrogen": (7, 7), "Oxygen": (8, 8), "Fluorine": (9, 10
    "Sodium": (11, 12), "Magnesium": (12, 12), "Aluminum": (13, 14), "Silicon"
    "Sulfur": (16, 16), "Chlorine": (17, 18), "Argon": (18, 22), "Potassium":
    "Scandium": (21, 24), "Titanium": (22, 26), "Vanadium": (23, 28), "Chromiu
```

```
    "Iron": (26, 30), "Cobalt": (27, 32), "Nickel": (28, 31), "Copper": (29, 3
    "Gallium": (31, 39), "Germanium": (32, 41), "Arsenic": (33, 42), "Selenium
    "Krypton": (36, 48), "Rubidium": (37, 48), "Strontium": (38, 50), "Yttrium
    "Niobium": (41, 52), "Molybdenum": (42, 54), "Technetium": (43, 55), "Ruth
    "Palladium": (46, 60), "Silver": (47, 61), "Cadmium": (48, 64), "Indium":
    "Antimony": (51, 71), "Tellurium": (52, 76), "Iodine": (53, 74), "Xenon":
    "Barium": (56, 81), "Lanthanum": (57, 82), "Cerium": (58, 82), "Praseodymi
    "Promethium": (61, 84), "Samarium": (62, 88), "Europium": (63, 89), "Gadol
    "Dysprosium": (66, 97), "Holmium": (67, 98), "Erbium": (68, 99), "Thulium"
    "Lutetium": (71, 104), "Hafnium": (72, 106), "Tantalum": (73, 108), "Tungs
    "Osmium": (76, 114), "Iridium": (77, 115), "Platinum": (78, 117), "Gold":
    "Thallium": (81, 123), "Lead": (82, 125), "Bismuth": (83, 126), "Polonium"
    "Radon": (86, 136), "Francium": (87, 136), "Radium": (88, 138), "Actinium"
    "Protactinium": (91, 140), "Uranium": (92, 146), "Neptunium": (93, 144), "
    "Americium": (95, 148), "Curium": (96, 151), "Berkelium": (97, 150), "Cali
    "Einsteinium": (99, 153), "Fermium": (100, 157), "Mendelevium": (101, 157)
    "Lawrencium": (103, 159), "Rutherfordium": (104, 157), "Dubnium": (105, 15
    "Bohrium": (107, 157), "Hassium": (108, 161), "Meitnerium": (109, 159), "D
    "Roentgenium": (111, 161), "Copernicium": (112, 173), "Nihonium": (113, 17
    "Moscovium": (115, 173), "Livermorium": (116, 178), "Tennessine": (117, 17
}

# Function to check if a number is a magic number
def is_magic_number(n):
    return n in magic_numbers

# Function to generate 3D coordinates for a vibrating spiral path with time-ba
def golden_spiral_oscillating(num_points, scale=1, offset_angle=0, z_scale=0.1
    angles = np.arange(num_points) * (2 * np.pi / golden_ratio) + offset_angle
    radii = scale * np.sqrt(np.arange(num_points))
    x = radii * np.cos(angles) * (1 + 0.1 * np.sin(2 * np.pi * frequency * t))
    y = radii * np.sin(angles) * (1 + 0.1 * np.cos(2 * np.pi * frequency * t))
    z = z_scale * np.arange(num_points) * (1 + 0.05 * np.sin(4 * np.pi * frequ
    return x, y, z

# Function to display usage information
def display_usage():
    print("Usage: python script.py <element_name> [<proton_count> <neutron_cou
    print("Example: python script.py Carbon")
    print("Example: python script.py CustomElement 20 20")
    sys.exit(1)

# Retrieve element name, proton count, and neutron count from command line arg
if len(sys.argv) < 2:
    display_usage()

element_name = sys.argv[1]

# Check if the element is in the lookup table
if element_name in element_data:
    num_protons, num_neutrons = element_data[element_name]
else:
    # If not found, check if proton and neutron counts are provided
    if len(sys.argv) != 4:
        print(f"Error: Element '{element_name}' not found in lookup. Please pr
        display_usage()
```

```
    try:
        num_protons = int(sys.argv[2])
        num_neutrons = int(sys.argv[3])
    except ValueError:
        print("Error: Proton and neutron counts must be integers.")
        display_usage()

# Check if proton or neutron count is a magic number to adjust stability prope
is_proton_magic = is_magic_number(num_protons)
is_neutron_magic = is_magic_number(num_neutrons)

# Animation function for dynamic oscillation with trails and stability colorin
def update(frame, ax, proton_trails, neutron_trails, frequency):
    ax.cla()

    # Stability color and transparency based on magic number status
    proton_color = 'blue' if is_proton_magic else 'cyan'
    neutron_color = 'red' if is_neutron_magic else 'orange'
    stability_alpha = 0.8 if is_proton_magic or is_neutron_magic else 0.5

    # Generate positions with oscillations
    proton_x, proton_y, proton_z = golden_spiral_oscillating(num_protons, scal
    neutron_x, neutron_y, neutron_z = golden_spiral_oscillating(num_neutrons,

    proton_trails.append((proton_x, proton_y, proton_z))
    neutron_trails.append((neutron_x, neutron_y, neutron_z))

    if len(proton_trails) > 30:
        proton_trails.pop(0)
    if len(neutron_trails) > 30:
        neutron_trails.pop(0)

    for idx, (p_x, p_y, p_z) in enumerate(proton_trails):
        ax.scatter(p_x, p_y, p_z, color=proton_color, s=10, alpha=(idx + 1) /
    for idx, (n_x, n_y, n_z) in enumerate(neutron_trails):
        ax.scatter(n_x, n_y, n_z, color=neutron_color, s=10, alpha=(idx + 1) /

    ax.scatter(proton_x, proton_y, proton_z, color=proton_color, s=50, alpha=s
    ax.scatter(neutron_x, neutron_y, neutron_z, color=neutron_color, s=30, alp

    for i in range(min(len(proton_x), len(neutron_x))):
        ax.plot([proton_x[i], neutron_x[i]], [proton_y[i], neutron_y[i]], [pro

    ax.set_title(f"Dynamic VFD Model of {element_name} ({num_protons} protons,
    ax.legend(loc='upper right')
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")

# Visualization setup
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')
ax.set_box_aspect([1, 1, 1])
proton_trails = []
neutron_trails = []
frames = 100
```

```
frequency = 0.02

# Create animation
anim = FuncAnimation(fig, update, frames=frames, fargs=(ax, proton_trails, neu

plt.show()
```

Category

1. Vibrational Field Dynamic

Date
2026/01/29
Date Created
2024/10/28
Author
leesmart