```
# Ver 3.0
import pandas as pd
import numpy as np
from numpy import *
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.pyplot import *
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.interpolate import PchipInterpolator
from scipy.signal import savgol_filter
from prettytable import PrettyTable
from collections import namedtuple
import pandas_profiling
import cufflinks as cf
import plotly.offline

# Error elimination, since it does not affect the values obtained
# The graph is rendered in 3D, the package for this type of charts uses the square root
# The presence of a pair of negative numbers excludes their visualization
import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)

comments = [[0, 'All data presented in the SI system',
             'https://en.wikipedia.org/wiki/International_System_of_Units \n'],
            [1, 'Proton and neutron consist of a core \n and two shells around them \n',
             'Robert Hofstadter the Nobel laureate \n'],
            [2, 'The proton consists of two quarks \n "u" and a quark "d" \n',
             'Murray Gell-Mann the Nobel laureate, \n and George Zweig \n'],
            [3, 'The neutron consists of two quarks \n "d" and a quark "u" \n',
             'Murray Gell-Mann the Nobel laureate, \n and George Zweig \n'],
            [4, '"Conditional quark" consists of a core and \n two shells \n',
             'The assumption of the author \n'],
            [5, 'Quark radius \n  "– (0.47 · 10E–16 cm)2 < RE2 < (0.43 · 10E–16 cm)2" \n',
             'https://arxiv.org/pdf/1604.01280.pdf \n'],
            [6, 'Proton, a neutron can be represented \n as the sum of three matrices \n',
             'The mathematical derivation of the author \n'],
            [7, '{x1, x2, x3, 0, 0} + {0, y1, y2, y3, 0} \n + {0, 0, x1, x2, x3} \n',
             'View of three matrices for obtaining \n a proton, neutron \n'],
            [8, 'x1, y1 - quark cores \n', "Usually, quarks proper in today's a view \n"],
            [9, '{x1, x2+y1, x3+y2+x1, y3+x2, x3} \n',
             'A schematic view of the matrix \n for a proton, neutron \n'],
            [10, '{x1, x2+y1, x3+y2+x1} - quark core \n', 'x1, y1 - quark cores \n'],
            [11, '{y3+x2, x3} - quark shells \n', 'x1, y1 - absent \n'],
            [12, 'The proposed approach allows one to obtain many \n different particles',
             'Calculation:quarks "u", "d", \n proton, neutron, pseudo proton, pseudo neutron
\n'],
            [13, 'π = 3.14159265358979', 'https://en.wikipedia.org/wiki/Pi \n'],
            [14, "Planck's constant, h = 6.62607015E−34",
             'https://en.wikipedia.org/wiki/Planck_constant \n'],
           [15, 'Compton wavelength, λ = h/mc',
            'https://en.wikipedia.org/wiki/Compton_wavelength \n'],
           [16, 'Speed of light in a vacuum, c = 299792458',
            'https://en.wikipedia.org/wiki/Speed_of_light \n'],
           [17, 'Electrical constant, ε0 = 8.85418781762039E−12',
            'https://en.wikipedia.org/wiki/Vacuum_permittivity'],
           [18, 'Gravitational constant, G = 6.67448478E-11',
            'newton per square meter per kilogram for the AAF method']]

table1 = PrettyTable(['#', 'Description', 'Link to source/ comments'])

for rec in comments:
    table1.add_row(rec)
```

```
class Preliminary():
# volume of proton + neutron
# Quark condensate provides about 9 percent of the proton's mass
# Physical Review Letters, 2018, website arXiv.org

# V = 4/3πR**3
    π = 3.14159265358979

    Vy = 4/3 * π * (2.5E-16)**3
    Vs1 = 4/3 * π * (1.4E-15)**3
    Vs2 = 4/3 * π * (2.5E-15)**3
    Vs11 = Vs1 - Vy
    Vs21 = Vs2 -Vs11

# https://physics.nist.gov/cgi-bin/cuu/Value?mp - 1.67262192369E-27 kg.

    mps2 = 0.09 * 1.67262192369E-27/(Vs11/Vs21 +1)

# https://physics.nist.gov/cgi-bin/cuu/Value?mn - 1.67492749804E-27 kg.

    mns2 = 0.09 * 1.67492749804E-27/(Vs11/Vs21 +1)
    mps1 = 0.09 * 1.67262192369E-27 - mps2
    mns1 = 0.09 * 1.67492749804E-27 - mns2

class Proton():
# The magnitude of the charge of the core, shells in the proton, neutron, respectively
#Robert Hofstadter the Nobel laureate

    SHELLSP1 = 0.35
    SHELLSP2 = 0.5
    SHELLSP3 = 0.15
    SHELLSN1 = 0.35
    SHELLSN2 = -0.5
    SHELLSN3 = 0.15

# The mass of the core, shells in the proton, neutron, respectively

    shellsmp1 = 1.67262192369E-27 * 0.91
    shellsmp2 = Preliminary.mps1
    shellsmp3 = Preliminary.mps2

    shellsmn1 = 1.67492749804E-27 * 0.91
    shellsmn2 = Preliminary.mns1
    shellsmn3 = Preliminary.mns2

    def __init__ (self, array):
        self.array = array

# Array input according to the matrix proposed by the author

a1 = array ([[2.0 , 1.0, 1.0, 1.0, 1.0, 0.0], [0.0, 1.0, 0.0, 0.0, 0.0, 0.0],
            [0.0, 0.0, 1.0, 0.0, 0.0, 0.0], [1.0, 1.0, 0.0, 2.0, 1.0, 1.0],
            [0.0, 0.0, 0.0, 0.0, 1.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 1.0]])


unit = Proton(a1)
unit.array

b1 = array ([Proton.SHELLSP1, Proton.SHELLSP2, Proton.SHELLSP3, Proton.SHELLSN1,
Proton.SHELLSN2,
             Proton.SHELLSN3])

# The calculation of electric charges of quark "u" and "d" for each shells in electron charges

x1 = linalg.solve (unit.array, b1)
```

```
qvark = list(x1)

data1 = {'index': ['uq1', 'uq2', 'uq3', 'dq1', 'dq2', 'dq3'],
        'Qe': [ qvark[0], qvark[1],  qvark[2],  qvark[3],  qvark[4],  qvark[5]]}

uq1 = qvark[0]
uq2 = qvark[1]
uq3 = qvark[2]
dq1 = qvark[3]
dq2 = qvark[4]
dq3 = qvark[5]

shell = [[1, 'uq1', uq1], [2, 'uq2', uq2], [3, 'uq3', uq3], [4, 'dq1', dq1],
     [5, 'dq2', dq2], [6, 'dq3', dq3]]

table = PrettyTable(['#', 'Index', 'Charge in the Qe'])

for rec in shell:
    table.add_row(rec)

# Calculation of the amount of charge on the shells, charge of an electron is taken modulo

Qe = 1.602176620898e-19
uq11 = Qe * qvark[0]
uq21 = Qe * qvark[1]
uq31 = Qe * qvark[2]
dq11 = Qe * qvark[3]
dq21 = Qe * qvark[4]
dq31 = Qe * qvark[5]

# The calculation of mass of quark "u" and "d" for each shells

b2 = array ([Proton.shellsmp1, Proton.shellsmp2, Proton.shellsmp3, Proton.shellsmn1,
             Proton.shellsmn2, Proton.shellsmn3])

x2 = linalg . solve (unit.array, b2)

qvarkm = list(x2)

data2 = {'index': ['um1', 'um2', 'um3', 'dm1', 'dm2', 'dm3'],
        'mass': [ qvarkm[0], qvarkm[1], qvarkm[2], qvarkm[3], qvarkm[4], qvarkm[5]]}

um1 = qvarkm[0]
um2 = qvarkm[1]
um3 = qvarkm[2]
dm1 = qvarkm[3]
dm2 = qvarkm[4]
dm3 = qvarkm[5]

# The calculation of volume of quark shells "u" and "d"

b3 = ([Preliminary.Vy, Preliminary.Vs11, Preliminary.Vs21, Preliminary.Vy,
       Preliminary.Vs11, Preliminary.Vs21])

x3 = linalg . solve (unit.array, b3)

qvarkv = list(x3)

data3 = {'index': ['uv1', 'uv2', 'uv3', 'dv1', 'dv2', 'dv3'],
        'mass': [ qvarkv[0], qvarkv[1], qvarkv[2], qvarkv[3], qvarkv[4], qvarkv[5]]}

uv1 = qvarkv[0]
uv2 = qvarkv[1]
uv3 = qvarkv[2]
```

```
dv1 = qvarkv[3]
dv2 = qvarkv[4]
dv3 = qvarkv[5]

# Data entry for quarks "u" and "d"

data = {'Index "u"': ['uq11', 'um1', 'uv1', 'uq21', 'um2', 'uv2', 'uq31', 'um3', 'uv3'],
        'Value "u"': [ uq11,   um1,   uv1,   uq21,   um2,   uv2,   uq31,   um3,   uv3],
       'Index "d"': ['dq11', 'dm1', 'dv1', 'dq21', 'dm2', 'dv2','dq31', 'dm3', 'dv3'],
       'Value "d"': [ dq11,   dm1,   dv1,   dq21,   dm2,   dv2,  dq31,   dm3,   dv3]}


quarku = [[1, 'uq11', uq11, 'um1', um1, 'uv1', uv1],
          [2, 'uq21', uq21, 'um2', um2, 'uv2', uv2],
          [3, 'uq31', uq31, 'um3', um3, 'uv3', uv3]]

table2 = PrettyTable(['#', 'Charge sym.', 'Charge in Cl', 'Mass sym.',
                      'Mass in kg.', 'Volume sym.', 'Volume in cbm'])

for rec in quarku:
    table2.add_row(rec)

quarkd = [[1, 'dq11', dq11, 'dm1', dm1, 'dv1', dv1],
          [2, 'dq21', dq21, 'dm2', dm2, 'dv2', dv2],
          [3, 'dq31', dq31, 'dm3', dm3, 'dv3', dv3]]

table3 = PrettyTable(['#', 'Charge sym.', 'Charge in Cl', 'Mass sym.',
                      'Mass in kg.', 'Volume sym.', 'Volume in cbm'])

for rec in quarkd:
    table3.add_row(rec)


# Description for proton, neutron, by shells
# The top line - the center, the bottom line - the upper shell
# The presented interactions in date4 are the author's approach


proton = [[1, 'pq1', uq11, 'pm1', um1, 'pv1', uv1],
          [2, 'pq2', uq21, 'pm2', um2, 'pv2', uv2],
          [3, 'pq3', dq11, 'pm3', dm1, 'pv3', dv1],
          [4, 'pq4', uq31, 'pm4', um3, 'pv4', uv3],
          [5, 'pq5', uq11, 'pm5', um1, 'pv5', uv1],
          [6, 'pq6', dq21, 'pm6', dm2, 'pv6', dv2],
          [7, 'pq7', dq31, 'pm7', dm3, 'pv7', dv3],
          [8, 'pq8', uq21, 'pm8', um2, 'pv8', uv2],
          [9, 'pq9', uq31, 'pm9', um3, 'pv9', uv3]]

table4 = PrettyTable(['#', 'Charge sym.', 'Charge in Cl', 'Mass sym.',
                      'Mass in kg.', 'Volume sym.', 'Volume in cbm'])

for rec in proton:
    table4.add_row(rec)

Pproton = namedtuple('Pproton', 'name1 charge name2 mass name3 volume')
protons = [Pproton('pq1', uq11, 'pm1', um1, 'pv1', uv1),
           Pproton('pq2', uq21, 'pm2', um2, 'pv2', uv2),
           Pproton('pq3', dq11, 'pm3', dm1, 'pv3', dv1),
           Pproton('pq4', uq31, 'pm4', um3, 'pv4', uv3),
           Pproton('pq5', uq11, 'pm5', um1, 'pv5', uv1),
           Pproton('pq6', dq21, 'pm6', dm2, 'pv6', dv2),
           Pproton('pq7', dq31, 'pm7', dm3, 'pv7', dv3),
           Pproton('pq8', uq21, 'pm8', um2, 'pv8', uv2),
           Pproton('pq9', uq31, 'pm9', um3, 'pv9', uv3)]
```

```
neutron = [[1, 'nq1', dq11, 'nm1', dm1, 'nv1', dv1],
           [2, 'nq2', dq21, 'nm2', dm2, 'nv2', dv2],
           [3, 'nq3', uq11, 'nm3', um1, 'nv3', uv1],
           [4, 'nq4', dq31, 'nm4', dm3, 'nv4', dv3],
           [5, 'nq5', dq11, 'nm5', dm1, 'nv5', dv1],
           [6, 'nq6', uq21, 'nm6', um2, 'nv6', uv2],
           [7, 'nq7', uq31, 'nm7', um3, 'nv7', uv3],
           [8, 'nq8', dq21, 'nm8', dm2, 'nv8', dv2],
           [9, 'nq9', dq31, 'nm9', dm3, 'nv9', dv3]]

table5 = PrettyTable(['#', 'Charge sym.', 'Charge in Cl', 'Mass sym.',
                      'Mass in kg.', 'Volume sym.', 'Volume in cbm'])

for rec in neutron:
    table5.add_row(rec)

Nneutron = namedtuple('Nneutron', 'name1 charge name2 mass name3 volume')
neutrons = [Nneutron('nq1', dq11, 'nm1', dm1, 'nv1', dv1),
            Nneutron('nq2', dq21, 'nm2', dm2, 'nv2', dv2),
            Nneutron('nq3', uq11, 'nm3', um1, 'nv3', uv1),
            Nneutron('nq4', dq31, 'nm4', dm3, 'nv4', dv3),
            Nneutron('nq5', dq11, 'nm5', dm1, 'nv5', dv1),
            Nneutron('nq6', uq21, 'nm6', um2, 'nv6', uv2),
            Nneutron('nq7', uq31, 'nm7', um3, 'nv7', uv3),
            Nneutron('nq8', dq21, 'nm8', dm2, 'nv8', dv2),
            Nneutron('nq9', dq31, 'nm9', dm3, 'nv9', dv3)]

class Pseudoneutron():
# the difference from the class proton in the matrix

    def __init__ (self, arr):
        self.arr = arr

# Array input according to the matrix proposed by the author

a2 = array ([[2.0 , 2.0, 1.0, 1.0, 0.0, 0.0], [0.0, 0.0, 1.0, 0.0, 1.0, 0.0],
             [0.0, 0.0, 0.0, 0.0, 0.0, 1.0], [1.0, 0.0, 0.0, 2.0, 2.0, 1.0],
             [0.0, 1.0, 0.0, 0.0, 0.0, 1.0], [0.0, 0.0, 1.0, 0.0, 0.0, 0.0]])


uni = Pseudoneutron(a2)
uni.arr

x4 = linalg.solve(uni.arr, b1)
x5 = linalg.solve(uni.arr, b2)
x6 = linalg.solve(uni.arr, b3)

psqvark = list(x4)

psdata1 = {'index': ['psuq1', 'psuq2', 'psuq3', 'psdq1', 'psdq2', 'psdq3'],
           'Qe': [ psqvark[0], psqvark[1],  psqvark[2],  psqvark[3],  psqvark[4],
                  psqvark[5]]}

psuq1 = psqvark[0]
psuq2 = psqvark[1]
psuq3 = psqvark[2]
psdq1 = psqvark[3]
psdq2 = psqvark[4]
psdq3 = psqvark[5]

psshell = [[1, 'psuq1', psuq1], [2, 'psuq2', psuq2], [3, 'psuq3', uq3], [4, 'psdq1', psdq1],
           [5, 'psdq2', psdq2], [6, 'psdq3', psdq3]]

pstable = PrettyTable(['#', 'Index', 'Charge in the Qe'])
```

```
for rec in psshell:
    pstable.add_row(rec)

# Calculation of the amount of charge on the shells, charge of an electron is taken modulo

Qe = 1.602176620898e-19
psuq11 = Qe * psqvark[0]
psuq21 = Qe * psqvark[1]
psuq31 = Qe * psqvark[2]
psdq11 = Qe * psqvark[3]
psdq21 = Qe * psqvark[4]
psdq31 = Qe * psqvark[5]

x4 = linalg.solve(uni.arr, b2)

psqvarkm = list(x4)

psdata2 = {'index': ['psum1', 'psum2', 'psum3', 'psdm1', 'psdm2', 'psdm3'],
           'mass': [ psqvarkm[0], psqvarkm[1], psqvarkm[2], psqvarkm[3],
                     psqvarkm[4], psqvarkm[5]]}

psum1 = psqvarkm[0]
psum2 = psqvarkm[1]
psum3 = psqvarkm[2]
psdm1 = psqvarkm[3]
psdm2 = psqvarkm[4]
psdm3 = psqvarkm[5]

x5 = linalg.solve(uni.arr, b3)

psqvarkv = list(x5)

psdata3 = {'index': ['psuv1', 'psuv2', 'psuv3', 'psdv1', 'psdv2', 'psdv3'],
           'mass': [ psqvarkv[0], psqvarkv[1], psqvarkv[2], psqvarkv[3],
                     psqvarkv[4], psqvarkv[5]]}

psuv1 = psqvarkv[0]
psuv2 = psqvarkv[1]
psuv3 = psqvarkv[2]
psdv1 = psqvarkv[3]
psdv2 = psqvarkv[4]
psdv3 = psqvarkv[5]

# Data entry for quarks "u" and "d"

psdata = {'Index "u"': ['psuq11', 'psum1', 'psuv1', 'psuq21',
                        'psum2', 'psuv2', 'psuq31', 'psum3', 'psuv3'],
        'Value "u"': [ psuq11,   psum1,   psuv1,   psuq21,   psum2,   psuv2,
                       psuq31,   psum3,   psuv3],
       'Index "d"': ['psdq11', 'psdm1', 'psdv1', 'psdq21', 'psdm2', 'psdv2',
                      'psdq31', 'psdm3', 'psdv3'],
       'Value "d"': [ psdq11,   psdm1,   psdv1,   psdq21,   psdm2,   psdv2,
                      psdq31,   psdm3,   psdv3]}


psquarku = [[1, 'psuq11', psuq11, 'psum1', psum1, 'psuv1', psuv1],
            [2, 'psuq21', psuq21, 'psum2', psum2, 'psuv2', psuv2],
            [3, 'psuq31', psuq31, 'psum3', psum3, 'psuv3', psuv3]]

pstable2 = PrettyTable(['#', 'Charge sym.', 'Charge in Cl', 'Mass sym.',
                        'Mass in kg.', 'Volume sym.', 'Volume in cbm'])

for rec in psquarku:
    pstable2.add_row(rec)
```

```
psquarkd = [[1, 'psdq11', psdq11, 'psdm1', psdm1, 'psdv1', psdv1],
            [2, 'psdq21', psdq21, 'psdm2', psdm2, 'psdv2', psdv2],
            [3, 'psdq31', psdq31, 'psdm3', psdm3, 'psdv3', psdv3]]

pstable3 = PrettyTable(['#', 'Charge sym.', 'Charge in Cl', 'Mass sym.',
                        'Mass in kg.', 'Volume sym.', 'Volume in cbm'])

for rec in psquarkd:
    pstable3.add_row(rec)


# Description for psproton, psneutron, by shells
# The top line - the center, the bottom line - the upper shell
# The presented interactions in date4 are the author's approach


psproton = [[1, 'pspq1', psuq11, 'pspm1', psum1, 'pspv1', psuv1],
            [2, 'pspq2', psuq21, 'pspm2', psum2, 'pspv2', psuv2],
            [3, 'pspq3', psdq11, 'pspm3', psdm1, 'pspv3', psdv1],
            [4, 'pspq4', psuq31, 'pspm4', psum3, 'pspv4', psuv3],
            [5, 'pspq5', psuq11, 'pspm5', psum1, 'pspv5', psuv1],
            [6, 'pspq6', psdq21, 'pspm6', psdm2, 'pspv6', psdv2],
            [7, 'pspq7', psdq31, 'pspm7', psdm3, 'pspv7', psdv3],
            [8, 'pspq8', psuq21, 'pspm8', psum2, 'pspv8', psuv2],
            [9, 'pspq9', psuq31, 'pspm9', psum3, 'pspv9', psuv3]]

pstable4 = PrettyTable(['#', 'Charge sym.', 'Charge in Cl', 'Mass sym.',
                        'Mass in kg.', 'Volume sym.', 'Volume in cbm'])

for rec in psproton:
    pstable4.add_row(rec)

Psproton = namedtuple('Psproton', 'name1 charge name2 mass name3 volume')
psprotons = [Psproton('pspq1', psuq11, 'pspm1', psum1, 'pspv1', psuv1),
             Psproton('pspq2', psuq21, 'pspm2', psum2, 'pspv2', psuv2),
             Psproton('pspq3', psdq11, 'pspm3', psdm1, 'pspv3', psdv1),
             Psproton('pspq4', psuq31, 'pspm4', psum3, 'pspv4', psuv3),
             Psproton('pspq5', psuq11, 'pspm5', psum1, 'pspv5', psuv1),
             Psproton('pspq6', psdq21, 'pspm6', psdm2, 'pspv6', psdv2),
             Psproton('pspq7', psdq31, 'pspm7', psdm3, 'pspv7', psdv3),
             Psproton('pspq8', psuq21, 'pspm8', psum2, 'pspv8', psuv2),
             Psproton('pspq9', psuq31, 'pspm9', psum3, 'pspv9', psuv3)]


psneutron = [[1, 'psnq1', psdq11, 'psnm1', psdm1, 'psnv1', psdv1],
             [2, 'psnq2', psdq21, 'psnm2', psdm2, 'psnv2', psdv2],
             [3, 'psnq3', psuq11, 'psnm3', psum1, 'psnv3', psuv1],
             [4, 'psnq4', psdq31, 'psnm4', psdm3, 'psnv4', psdv3],
             [5, 'psnq5', psdq11, 'psnm5', psdm1, 'psnv5', psdv1],
             [6, 'psnq6', psuq21, 'psnm6', psum2, 'psnv6', psuv2],
             [7, 'psnq7', psuq31, 'psnm7', psum3, 'psnv7', psuv3],
             [8, 'psnq8', psdq21, 'psnm8', psdm2, 'psnv8', psdv2],
             [9, 'psnq9', psdq31, 'psnm9', psdm3, 'psnv9', psdv3]]

pstable5 = PrettyTable(['#', 'Charge sym.', 'Charge in Cl', 'Mass sym.',
                        'Mass in kg.', 'Volume sym.', 'Volume in cbm'])

for rec in psneutron:
    pstable5.add_row(rec)

Psneutron = namedtuple('Psneutron', 'name1 charge name2 mass name3 volume')
psneutrons = [Psneutron('psnq1', psdq11, 'psnm1', psdm1, 'psnv1', psdv1),
              Psneutron('psnq2', psdq21, 'psnm2', psdm2, 'psnv2', psdv2),
              Psneutron('psnq3', psuq11, 'psnm3', psum1, 'psnv3', psuv1),
```

```
              Psneutron('psnq4', psdq31, 'psnm4', psdm3, 'psnv4', psdv3),
              Psneutron('psnq5', psdq11, 'psnm5', psdm1, 'psnv5', psdv1),
              Psneutron('psnq6', psuq21, 'psnm6', psum2, 'psnv6', psuv2),
              Psneutron('psnq7', psuq31, 'psnm7', psum3, 'psnv7', psuv3),
              Psneutron('psnq8', psdq21, 'psnm8', psdm2, 'psnv8', psdv2),
              Psneutron('psnq9', psdq31, 'psnm9', psdm3, 'psnv9', psdv3)]


# Obtaining data for analysis

# calculation of wave parameters

# Compton wavelength
class Wavep():

# Planck's constant
    CONSTANTH = 6.62607015E-34

# The speed of light in a vacuum
    CONSTANTC = 299792458

# The ratio of Planck's constant to the speed of light in a vacuum, D

    D = CONSTANTH/CONSTANTC

    def __init__ (self, comptonlp):
        self.comptonlp = comptonlp


numbers = [1/protons[0].mass, 1/protons[1].mass, 1/protons[2].mass, 1/protons[3].mass,
           1/protons[4].mass, 1/protons[5].mass, 1/protons[6].mass, 1/protons[7].mass,
           1/protons[8].mass]
for i, item in enumerate(numbers):
       numbers[i] *= Wavep.D

unit2 = Wavep(numbers)

class Waven():

    def __init__ (self, comptonln):
        self.comptonln = comptonln


numbersn = [1/neutrons[0].mass, 1/neutrons[1].mass, 1/neutrons[2].mass, 1/neutrons[3].mass,
            1/neutrons[4].mass, 1/neutrons[5].mass, 1/neutrons[6].mass, 1/neutrons[7].mass,
            1/neutrons[8].mass]
for i, item in enumerate(numbersn):
       numbersn[i] *= Wavep.D


unit3 = Waven(numbersn)

class Wavepsn():

    def __init__ (self, comptonlpsn):
        self.comptonlpsn = comptonlpsn


numberspsn = [1/psneutrons[0].mass, 1/psneutrons[1].mass, 1/psneutrons[2].mass,
              1/psneutrons[3].mass, 1/psneutrons[4].mass, 1/psneutrons[5].mass,
              1/psneutrons[6].mass, 1/psneutrons[7].mass, 1/psneutrons[8].mass]
for i, item in enumerate(numberspsn):
       numberspsn[i] *= Wavep.D
```

```
unit4 = Wavepsn(numberspsn)

class Wavepsp():

    def __init__ (self, comptonlpsp):
        self.comptonlpsp = comptonlpsp


numberspsp = [1/psprotons[0].mass, 1/psprotons[1].mass, 1/psprotons[2].mass,
              1/psprotons[3].mass, 1/psprotons[4].mass, 1/psprotons[5].mass,
              1/psprotons[6].mass, 1/psprotons[7].mass, 1/psprotons[8].mass]
for i, item in enumerate(numberspsp):
       numberspsp[i] *= Wavep.D


unit5 = Wavepsp(numberspsp)

#Electromagnetic characteristic of fine structure
class ElectricWavep():

# Planck's constant
    CONSTANTH = 6.62607015E-34

# The speed of light in a vacuum
    CONSTANTC = 299792458

# The electrical constant ε
    CONSTANTE0 = 8.85418781762039E-12

# The ratio to the unit of the doubled product of the electrical constant,
# Planck's constant, the speed of light in vacuum

    constantd = 1/(2 * CONSTANTE0 * CONSTANTH * CONSTANTC)

    def __init__ (self, elektromagnetikp):
        self.elektromagnetikp = elektromagnetikp


numbers = [protons[0].charge **2, protons[1].charge **2, protons[2].charge **2,
           protons[3].charge **2, protons[4].charge **2, protons[5].charge **2,
           protons[6].charge **2, protons[7].charge **2, protons[8].charge **2]

for i, item in enumerate(numbers):
       numbers[i] *= ElectricWavep.constantd

unit6 = ElectricWavep(numbers)

class ElectricWaven():

# Planck's constant
    CONSTANTH = 6.62607015E-34

# The speed of light in a vacuum
    CONSTANTC = 299792458

# The electrical constant ε
    CONSTANTE0 = 8.85418781762039E-12

# The ratio to the unit of the doubled product of the electrical constant,
# Planck's constant, the speed of light in vacuum

    constantd = 1/(2 * CONSTANTE0 * CONSTANTH * CONSTANTC)

    def __init__ (self, elektromagnetikn):
        self.elektromagnetikn = elektromagnetikn
```

```
numbers = [neutrons[0].charge **2, neutrons[1].charge **2, neutrons[2].charge **2,
           neutrons[3].charge **2, neutrons[4].charge **2, neutrons[5].charge **2,
           neutrons[6].charge **2, neutrons[7].charge **2, neutrons[8].charge **2]

for i, item in enumerate(numbers):
       numbers[i] *= ElectricWaven.constantd

unit7 = ElectricWaven(numbers)

class ElectricWavepsn():

# Planck's constant
    CONSTANTH = 6.62607015E-34

# The speed of light in a vacuum
    CONSTANTC = 299792458

# The electrical constant ε
    CONSTANTE0 = 8.85418781762039E-12

# The ratio to the unit of the doubled product of the electrical constant,
# Planck's constant, the speed of light in vacuum

    constantd = 1/(2 * CONSTANTE0 * CONSTANTH * CONSTANTC)

    def __init__ (self, elektromagnetikpsn):
        self.elektromagnetikpsn = elektromagnetikpsn


numbers = [psneutrons[0].charge **2, psneutrons[1].charge **2, psneutrons[2].charge **2,
           psneutrons[3].charge **2, psneutrons[4].charge **2, psneutrons[5].charge **2,
           psneutrons[6].charge **2, psneutrons[7].charge **2, psneutrons[8].charge **2]

for i, item in enumerate(numbers):
       numbers[i] *= ElectricWavepsn.constantd

unit8 = ElectricWavepsn(numbers)

class ElectricWavepsp():

# Planck's constant
    CONSTANTH = 6.62607015E-34

# The speed of light in a vacuum
    CONSTANTC = 299792458

# The electrical constant ε
    CONSTANTE0 = 8.85418781762039E-12

# The ratio to the unit of the doubled product of the electrical constant,
# Planck's constant, the speed of light in vacuum

    constantd = 1/(2 * CONSTANTE0 * CONSTANTH * CONSTANTC)

    def __init__ (self, elektromagnetikpsp):
        self.elektromagnetikpsp = elektromagnetikpsp


numbers = [psprotons[0].charge **2, psprotons[1].charge **2, psprotons[2].charge **2,
           psprotons[3].charge **2, psprotons[4].charge **2, psprotons[5].charge **2,
           psprotons[6].charge **2, psprotons[7].charge **2, psprotons[8].charge **2]

for i, item in enumerate(numbers):
```

```
        numbers[i] *= ElectricWavepsp.constantd

unit9 = ElectricWavepsp(numbers)

# Gravity
class GravityWavep():

# Planck's constant
    CONSTANTH = 6.62607015E-34

# The speed of light in a vacuum
    CONSTANTC = 299792458

# The Gravitational constant
    CONSTANTEG = 6.67448478E-11

    π = 3.14159265358979

# The ratio of the doubled product of pi and the gravitational constant to
# Planck's constant, the speed of light in vacuum

    constantg = 2 * π * CONSTANTEG/(CONSTANTH * CONSTANTC)

    def __init__ (self, gravp):
        self.gravp = gravp


numbers = [protons[0].mass **2, protons[1].mass **2, protons[2].mass **2, protons[3].mass **2,
           protons[4].mass **2, protons[5].mass **2, protons[6].mass **2, protons[7].mass **2,
           protons[8].mass **2]

for i, item in enumerate(numbers):
       numbers[i] *= GravityWavep.constantg

unit10 = GravityWavep(numbers)

class GravityWaven():

# Planck's constant
    CONSTANTH = 6.62607015E-34

# The speed of light in a vacuum
    CONSTANTC = 299792458

# The Gravitational constant
    CONSTANTEG = 6.67448478E-11

    π = 3.14159265358979

# The ratio of the doubled product of pi and the gravitational constant to
# Planck's constant, the speed of light in vacuum

    constantg = 2 * π * CONSTANTEG/(CONSTANTH * CONSTANTC)

    def __init__ (self, gravn):
        self.gravn = gravn


numbers = [neutrons[0].mass **2, neutrons[1].mass **2, neutrons[2].mass **2,
           neutrons[3].mass **2, neutrons[4].mass **2, neutrons[5].mass **2,
           neutrons[6].mass **2, neutrons[7].mass **2, neutrons[8].mass **2]

for i, item in enumerate(numbers):
       numbers[i] *= GravityWaven.constantg
```

```
unit11 = GravityWaven(numbers)

class GravityWavepsn():

# Planck's constant
    CONSTANTH = 6.62607015E-34

# The speed of light in a vacuum
    CONSTANTC = 299792458

# The Gravitational constant
    CONSTANTEG = 6.67448478E-11

    π = 3.14159265358979

# The ratio of the doubled product of pi and the gravitational constant to
# Planck's constant, the speed of light in vacuum

    constantg = 2 * π * CONSTANTEG/(CONSTANTH * CONSTANTC)

    def __init__ (self, gravpsn):
        self.gravpsn = gravpsn


numbers = [psneutrons[0].mass **2, psneutrons[1].mass **2, psneutrons[2].mass **2,
           psneutrons[3].mass **2, psneutrons[4].mass **2, psneutrons[5].mass **2,
           psneutrons[6].mass **2, psneutrons[7].mass **2, psneutrons[8].mass **2]

for i, item in enumerate(numbers):
       numbers[i] *= GravityWavepsn.constantg

unit12 = GravityWavepsn(numbers)

class GravityWavepsp():

# Planck's constant
    CONSTANTH = 6.62607015E-34

# The speed of light in a vacuum
    CONSTANTC = 299792458

# The Gravitational constant
    CONSTANTEG = 6.67448478E-11

    π = 3.14159265358979

# The ratio of the doubled product of pi and the gravitational constant to
# Planck's constant, the speed of light in vacuum

    constantg = 2 * π * CONSTANTEG/(CONSTANTH * CONSTANTC)

    def __init__ (self, gravpsp):
        self.gravpsp = gravpsp


numbers = [psprotons[0].mass **2, psprotons[1].mass **2, psprotons[2].mass **2,
           psprotons[3].mass **2, psprotons[4].mass **2, psprotons[5].mass **2,
           psprotons[6].mass **2, psprotons[7].mass **2, psprotons[8].mass **2]

for i, item in enumerate(numbers):
       numbers[i] *= GravityWavepsp.constantg

unit13 = GravityWavepsp(numbers)

# Obtaining additional data for analysis
```

```
# Gravity
class Frequencyp():

# The speed of light in a vacuum
    CONSTANTC = 299792458


    def __init__ (self, frequencep):
        self.frequencep = frequencep


numbers = [1/unit2.comptonlp[0], 1/unit2.comptonlp[1], 1/unit2.comptonlp[2],
           1/unit2.comptonlp[3], 1/unit2.comptonlp[4], 1/unit2.comptonlp[5],
           1/unit2.comptonlp[6], 1/unit2.comptonlp[7], 1/unit2.comptonlp[8]]

for i, item in enumerate(numbers):
       numbers[i] *= Frequencyp.CONSTANTC

unit14 = Frequencyp(numbers)

class Frequencyn():

# The speed of light in a vacuum
    CONSTANTC = 299792458


    def __init__ (self, frequencen):
        self.frequencen = frequencen


numbers = [1/unit3.comptonln[0], 1/unit3.comptonln[1], 1/unit3.comptonln[2],
           1/unit3.comptonln[3], 1/unit3.comptonln[4], 1/unit3.comptonln[5],
           1/unit3.comptonln[6], 1/unit3.comptonln[7], 1/unit3.comptonln[8]]

for i, item in enumerate(numbers):
       numbers[i] *= Frequencyn.CONSTANTC

unit15 = Frequencyn(numbers)

class Frequencypsn():

# The speed of light in a vacuum
    CONSTANTC = 299792458

    def __init__ (self, frequencepsn):
        self.frequencepsn = frequencepsn


numbers = [1/unit4.comptonlpsn[0], 1/unit4.comptonlpsn[1],
           1/unit4.comptonlpsn[2], 1/unit4.comptonlpsn[3],
           1/unit4.comptonlpsn[4], 1/unit4.comptonlpsn[5],
           1/unit4.comptonlpsn[6], 1/unit4.comptonlpsn[7],
           1/unit4.comptonlpsn[8]]

for i, item in enumerate(numbers):
       numbers[i] *= Frequencypsn.CONSTANTC

unit16 = Frequencypsn(numbers)

class Frequencypsp():

# The speed of light in a vacuum
    CONSTANTC = 299792458

    def __init__ (self, frequencepsp):
```

```
        self.frequencepsp = frequencepsp


numbers = [1/unit5.comptonlpsp[0], 1/unit5.comptonlpsp[1],
           1/unit5.comptonlpsp[2], 1/unit5.comptonlpsp[3],
           1/unit5.comptonlpsp[4], 1/unit5.comptonlpsp[5],
           1/unit5.comptonlpsp[6], 1/unit5.comptonlpsp[7],
           1/unit5.comptonlpsp[8]]

for i, item in enumerate(numbers):
       numbers[i] *= Frequencypsp.CONSTANTC

unit17 = Frequencypsp(numbers)

# Visualization

# Delta between the masses of the neutron and proton by shells, 2D graph

delta = ([neutrons[0].mass-protons[0].mass, neutrons[1].mass-protons[1].mass,
          neutrons[2].mass-protons[2].mass, neutrons[3].mass-protons[3].mass,
          neutrons[4].mass-protons[4].mass, neutrons[5].mass-protons[5].mass,
          neutrons[6].mass-protons[6].mass, neutrons[7].mass-protons[7].mass,
          neutrons[8].mass-protons[8].mass])
shell = ([1, 2, 3, 4, 5, 6, 7, 8, 9])
plt.figure(figsize=(12,8))
plt.plot(shell, delta, color = "green")
plt.bar(shell, delta, color = "lightgray")

plt.xlabel('Shell number \n \n Delta between the masses of the neutron and proton by shells \n
\n',
           fontsize=18)
plt.ylabel('Weight in kg * 10^-30', fontsize=18)
grid(True)

# Delta between the masses of the psneutron and psproton by shells, 2D graph

delta2 = ([psneutrons[0].mass -psprotons[0].mass, psneutrons[1].mass -psprotons[1].mass,
           psneutrons[2].mass -psprotons[2].mass, psneutrons[3].mass -psprotons[3].mass,
           psneutrons[4].mass -psprotons[4].mass, psneutrons[5].mass -psprotons[5].mass,
           psneutrons[6].mass -psprotons[6].mass, psneutrons[7].mass -psprotons[7].mass,
           psneutrons[8].mass -psprotons[8].mass])
#shell = ([1, 2, 3, 4, 5, 6, 7, 8, 9])
plt.figure(figsize=(12,8))
plt.plot(shell, delta2, color = "green")
plt.bar(shell, delta2, color = "lightgray")

fig2 = plt.xlabel('Shell number \n \n Delta between the masses of the psneutron and psproton by
shells \n \n',
                  fontsize=18)
fig2 = plt.ylabel('Weight in kg * 10^-30', fontsize=18)
grid(True)

# Graphical display of data on mass, volume, charge for proton, neutron, pseudo proton, pseudo
neutron
print('Graphical display of data on mass, volume, charge for proton, neutron, pseudo proton,
pseudo neutron')

data4 = {'Index "p"': ['charge1', 'mass1', 'volume1', 'charge2', 'mass2', 'volume2', 'charge3',
                       'mass3', 'volume3', 'charge4', 'mass4', 'volume4', 'charge5', 'mass5',
                       'volume5', 'charge6', 'mass6', 'volume6', 'charge7', 'mass7', 'volume7',
                       'charge8', 'mass8', 'volume8', 'charge9', 'mass9', 'volume9'],
        'Value "p"': [protons[0].charge, protons[0].mass, protons[0].volume,
                      protons[1].charge, protons[1].mass, protons[1].volume,
                      protons[2].charge, protons[2].mass, protons[2].volume,
                      protons[3].charge, protons[3].mass, protons[3].volume,
```

```
                        protons[4].charge, protons[4].mass, protons[4].volume,
                        protons[5].charge, protons[5].mass, protons[5].volume,
                        protons[6].charge, protons[6].mass, protons[6].volume,
                        protons[7].charge, protons[7].mass, protons[7].volume,
                        protons[8].charge, protons[8].mass, protons[8].volume],

        'Index "n"': ['charge1', 'mass1', 'volume1', 'charge2', 'mass2', 'volume2', 'charge3',
                      'mass3', 'volume3', 'charge4', 'mass4', 'volume4', 'charge5', 'mass5',
                      'volume5', 'charge6', 'mass6', 'volume6', 'charge7', 'mass7', 'volume7',
                      'charge8', 'mass8', 'volume8', 'charge9', 'mass9', 'volume9'],
        'Value "n"': [ neutrons[0].charge, neutrons[0].mass, neutrons[0].volume,
                       neutrons[1].charge, neutrons[1].mass, neutrons[1].volume,
                       neutrons[2].charge, neutrons[2].mass, neutrons[2].volume,
                       neutrons[3].charge, neutrons[3].mass, neutrons[3].volume,
                       neutrons[4].charge, neutrons[4].mass, neutrons[4].volume,
                       neutrons[5].charge, neutrons[5].mass, neutrons[5].volume,
                       neutrons[6].charge, neutrons[6].mass, neutrons[6].volume,
                       neutrons[7].charge, neutrons[7].mass, neutrons[7].volume,
                       neutrons[8].charge, neutrons[8].mass, neutrons[8].volume],
        'Index "psp"': ['charge1', 'mass1', 'volume1', 'charge2', 'mass2', 'volume2', 'charge3',
                        'mass3', 'volume3', 'charge4', 'mass4', 'volume4', 'charge5', 'mass5',
                        'volume5', 'charge6', 'mass6', 'volume6', 'charge7', 'mass7', 'volume7',
                        'charge8', 'mass8', 'volume8', 'charge9', 'mass9', 'volume9'],
         'Value "psp"': [psprotons[0].charge, psprotons[0].mass, psprotons[0].volume,
                         psprotons[1].charge, psprotons[1].mass, psprotons[1].volume,
                         psprotons[2].charge, psprotons[2].mass, psprotons[2].volume,
                         psprotons[3].charge, psprotons[3].mass, psprotons[3].volume,
                         psprotons[4].charge, psprotons[4].mass, psprotons[4].volume,
                         psprotons[5].charge, psprotons[5].mass, psprotons[5].volume,
                         psprotons[6].charge, psprotons[6].mass, psprotons[6].volume,
                         psprotons[7].charge, psprotons[7].mass, psprotons[7].volume,
                         psprotons[8].charge, psprotons[8].mass, psprotons[8].volume],

        'Index "psn"': ['charge1', 'mass1', 'volume1', 'charge2', 'mass2', 'volume2', 'charge3',
                        'mass3', 'volume3', 'charge4', 'mass4', 'volume4', 'charge5', 'mass5',
                        'volume5', 'charge6', 'mass6', 'volume6', 'charge7', 'mass7', 'volume7',
                        'charge8', 'mass8', 'volume8', 'charge9', 'mass9', 'volume9'],
        'Value "psn"': [psneutrons[0].charge, psneutrons[0].mass, psneutrons[0].volume,
                        psneutrons[1].charge, psneutrons[1].mass, psneutrons[1].volume,
                        psneutrons[2].charge, psneutrons[2].mass, psneutrons[2].volume,
                        psneutrons[3].charge, psneutrons[3].mass, psneutrons[3].volume,
                        psneutrons[4].charge, psneutrons[4].mass, psneutrons[4].volume,
                        psneutrons[5].charge, psneutrons[5].mass, psneutrons[5].volume,
                        psneutrons[6].charge, psneutrons[6].mass, psneutrons[6].volume,
                        psneutrons[7].charge, psneutrons[7].mass, psneutrons[7].volume,
                        psneutrons[8].charge, psneutrons[8].mass, psneutrons[8].volume]}

df = pd.DataFrame.from_dict(data4)

pandas_profiling.ProfileReport(df)
profile = pandas_profiling.ProfileReport(df)

cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)

df.iplot()

# Interrelation of mass, volume, charge within a proton, 3D graph

fig = plt.figure(figsize=plt.figaspect(0.3))

ax = fig.add_subplot(1, 2, 1, projection='3d')

Xpp = ([protons[0].charge, protons[1].charge, protons[2].charge, protons[3].charge,
        protons[4].charge, protons[5].charge, protons[6].charge, protons[7].charge,
```

```
        protons[8].charge])
Ypp = ([protons[0].volume, protons[1].volume, protons[2].volume, protons[3].volume,
        protons[4].volume, protons[5].volume, protons[6].volume, protons[7].volume,
        protons[8].volume])
Zpp = ([protons[0].mass, protons[1].mass, protons[2].mass, protons[3].mass,
        protons[4].mass, protons[5].mass, protons[6].mass, protons[7].mass,
        protons[8].mass])

ax.plot(Xpp,Ypp,Zpp)

ax.set_xlabel('\n \n \n The quantity charge shell \n in Cl x E-20', fontsize = 15)
ax.set_zlabel('\n \n \n Mass in \n kg. x E-28', fontsize = 15)
ax.set_ylabel('\n \n \n Shell volume in\n cbm*E-44', fontsize = 15)

ax.text2D(0.2, 0.95,
          "Interrelation of mass, volume, \n charge within a proton",
          transform=ax.transAxes, fontsize = 16)


# Interrelation of mass, volume, charge within a neutron, 3D graph

ax = fig.add_subplot(1, 2, 2, projection='3d')

Xnn = ([neutrons[0].charge, neutrons[1].charge, neutrons[2].charge, neutrons[3].charge,
        neutrons[4].charge, neutrons[5].charge, neutrons[6].charge, neutrons[7].charge,
        neutrons[8].charge])
Ynn = ([neutrons[0].volume, neutrons[1].volume, neutrons[2].volume, neutrons[3].volume,
        neutrons[4].volume, neutrons[5].volume, neutrons[6].volume, neutrons[7].volume,
        neutrons[8].volume])
Znn = ([neutrons[0].mass, neutrons[1].mass, neutrons[2].mass, neutrons[3].mass,
        neutrons[4].mass, neutrons[5].mass, neutrons[6].mass, neutrons[7].mass,
        neutrons[8].mass])

ax.plot(Xnn,Ynn,Znn)

ax.set_xlabel('\n \n \n The quantity charge shell \n in Cl x E-20', fontsize = 15)
ax.set_zlabel('\n \n \n Mass in \n kg. x E-28', fontsize = 15)
ax.set_ylabel('\n \n \n Shell volume in\n cbm*E-44', fontsize = 15)

ax.text2D(0.2, 0.95,
          "Interrelation of mass, volume, \n charge within a neutron",
          transform=ax.transAxes, fontsize = 16)

# Interrelation of mass, volume, charge within a psproton, 3D graph

fig = plt.figure(figsize=plt.figaspect(0.3))

ax = fig.add_subplot(1, 2, 1, projection='3d')

Xpp = ([psprotons[0].charge, psprotons[1].charge, psprotons[2].charge, psprotons[3].charge,
        psprotons[4].charge, psprotons[5].charge, psprotons[6].charge, psprotons[7].charge,
        psprotons[8].charge])
Ypp = ([psprotons[0].volume, psprotons[1].volume, psprotons[2].volume, psprotons[3].volume,
        psprotons[4].volume, psprotons[5].volume, psprotons[6].volume, psprotons[7].volume,
        psprotons[8].volume])
Zpp = ([psprotons[0].mass, psprotons[1].mass, psprotons[2].mass, psprotons[3].mass,
        psprotons[4].mass, psprotons[5].mass, psprotons[6].mass, psprotons[7].mass,
        psprotons[8].mass])

ax.plot(Xpp,Ypp,Zpp)
ax.set_xlabel('\n \n \n The quantity charge shell \n in Cl x E-19', fontsize = 15)
ax.set_zlabel('\n \n \n Mass in \n kg. x E-28', fontsize = 15)
ax.set_ylabel('\n \n \n Shell volume in\n cbm*E-44', fontsize = 15)

ax.text2D(0.2, 0.95,
```

```
          "Interrelation of mass, volume, \n charge within a psproton",
          transform=ax.transAxes, fontsize = 16)

# Interrelation of mass, volume, charge within a psneutron, 3D graph

ax = fig.add_subplot(1, 2, 2, projection='3d')

Xnn = ([psneutrons[0].charge, psneutrons[1].charge, psneutrons[2].charge, psneutrons[3].charge,
        psneutrons[4].charge, psneutrons[5].charge, psneutrons[6].charge, psneutrons[7].charge,
        psneutrons[8].charge])
Ynn = ([psneutrons[0].volume, psneutrons[1].volume, psneutrons[2].volume, psneutrons[3].volume,
        psneutrons[4].volume, psneutrons[5].volume, psneutrons[6].volume, psneutrons[7].volume,
        psneutrons[8].volume])
Znn = ([psneutrons[0].mass, psneutrons[1].mass, psneutrons[2].mass, psneutrons[3].mass,
        psneutrons[4].mass, psneutrons[5].mass, psneutrons[6].mass, psneutrons[7].mass,
        psneutrons[8].mass])

ax.plot(Xnn,Ynn,Znn)
ax.set_xlabel('\n \n \n The quantity charge shell \n in Cl x E-19', fontsize = 15)
ax.set_zlabel('\n \n \n Mass in \n kg. x E-28', fontsize = 15)
ax.set_ylabel('\n \n \n Shell volume in\n cbm*E-44', fontsize = 15)

ax.text2D(0.2, 0.95,
          "Interrelation of mass, volume, \n charge within a psneutron",
          transform=ax.transAxes, fontsize = 16)

# The relationship of wavelength, gravity and electromagnetism in a neutron, 3D graph
fig = plt.figure(figsize=plt.figaspect(0.3))

ax = fig.add_subplot(1, 2, 1, projection='3d')

Xnn = ([unit3.comptonln[0], unit3.comptonln[1], unit3.comptonln[2], unit3.comptonln[3],
        unit3.comptonln[4], unit3.comptonln[5], unit3.comptonln[6], unit3.comptonln[7],
        unit3.comptonln[8]])
Ynn = ([unit7.elektromagnetikn[0], unit7.elektromagnetikn[1], unit7.elektromagnetikn[2],
        unit7.elektromagnetikn[3], unit7.elektromagnetikn[4], unit7.elektromagnetikn[5],
        unit7.elektromagnetikn[6], unit7.elektromagnetikn[7], unit7.elektromagnetikn[8]])
Znn = ([unit11.gravn[0], unit11.gravn[0], unit11.gravn[0], unit11.gravn[0], unit11.gravn[0],
        unit11.gravn[0], unit11.gravn[0], unit11.gravn[0], unit11.gravn[0]])

ax.plot(Xnn,Ynn,Znn)

ax.set_xlabel('\n \n \n Compton wavelength \n ', fontsize = 15)
ax.set_zlabel('\n \n \n \n Electromagnetic \n indicator \n ', fontsize = 15)
ax.set_ylabel('\n \n \n Gravity \n indicator\n ', fontsize = 15)

ax.text2D(0.2, 0.95,
          "The relationship of wavelength, gravity and \n electromagnetism in a neutron",
          transform=ax.transAxes, fontsize = 16)

# The relationship of wavelength, gravity and electromagnetism in a proton, 3D graf

ax = fig.add_subplot(1, 2, 2, projection='3d')

Xnn = ([unit2.comptonlp[0], unit2.comptonlp[1], unit2.comptonlp[2], unit2.comptonlp[3],
        unit2.comptonlp[4], unit2.comptonlp[5], unit2.comptonlp[6], unit2.comptonlp[7],
        unit2.comptonlp[8]])
Ynn = ([unit6.elektromagnetikp[0], unit6.elektromagnetikp[1], unit6.elektromagnetikp[2],
        unit6.elektromagnetikp[3], unit6.elektromagnetikp[4], unit6.elektromagnetikp[5],
        unit6.elektromagnetikp[6], unit6.elektromagnetikp[7], unit6.elektromagnetikp[8]])
Znn = ([unit10.gravp[0], unit10.gravp[0], unit10.gravp[0], unit10.gravp[0], unit10.gravp[0],
        unit10.gravp[0], unit10.gravp[0], unit10.gravp[0], unit10.gravp[0]])

ax.plot(Xnn,Ynn,Znn)
```

```
ax.set_xlabel('\n \n \n Compton wavelength \n ', fontsize = 15)
ax.set_zlabel('\n \n \n \n Electromagnetic \n indicator \n ', fontsize = 15)
ax.set_ylabel('\n \n \n Gravity \n indicator\n ', fontsize = 15)

ax.text2D(0.2, 0.95,
          "The relationship of wavelength, gravity and \n electromagnetism in a proton",
          transform=ax.transAxes, fontsize = 16)

# Interrelation of frequency, gravity and electromagnetism in neutron, 3D graf

fig = plt.figure(figsize=plt.figaspect(0.3))

ax = fig.add_subplot(1, 2, 1, projection='3d')

Xnn = ([unit15.frequencen[0], unit15.frequencen[1], unit15.frequencen[2],
        unit15.frequencen[3], unit15.frequencen[4], unit15.frequencen[5],
        unit15.frequencen[6], unit15.frequencen[7], unit15.frequencen[8]])
Ynn = ([unit7.elektromagnetikn[0], unit7.elektromagnetikn[1], unit7.elektromagnetikn[2],
        unit7.elektromagnetikn[3], unit7.elektromagnetikn[4], unit7.elektromagnetikn[5],
        unit7.elektromagnetikn[6], unit7.elektromagnetikn[7], unit7.elektromagnetikn[8]])
Znn = ([unit11.gravn[0], unit11.gravn[0], unit11.gravn[0], unit11.gravn[0],
        unit11.gravn[0], unit11.gravn[0], unit11.gravn[0], unit11.gravn[0], unit11.gravn[0]])

ax.plot(Xnn,Ynn,Znn)

ax.set_xlabel('\n \n \n Frequency \n ', fontsize = 15)
ax.set_zlabel('\n \n \n \n Electromagnetic \n indicator \n ', fontsize = 15)
ax.set_ylabel('\n \n \n Gravity \n indicator\n ', fontsize = 15)

ax.text2D(0.2, 0.95,
          "Interrelation of frequency, gravity and \n electromagnetism in neutron",
          transform=ax.transAxes, fontsize = 16)

# Interrelation of frequency, gravity and electromagnetism in proton, 3D graf

ax = fig.add_subplot(1, 2, 2, projection='3d')

Xnn = ([unit14.frequencep[0], unit14.frequencep[1], unit14.frequencep[2],
        unit14.frequencep[3], unit14.frequencep[4], unit14.frequencep[5],
        unit14.frequencep[6], unit14.frequencep[7], unit14.frequencep[8]])
Ynn = ([unit6.elektromagnetikp[0], unit6.elektromagnetikp[1],
        unit6.elektromagnetikp[2], unit6.elektromagnetikp[3],
        unit6.elektromagnetikp[4], unit6.elektromagnetikp[5],
        unit6.elektromagnetikp[6], unit6.elektromagnetikp[7],
        unit6.elektromagnetikp[8]])
Znn = ([unit10.gravp[0], unit10.gravp[0], unit10.gravp[0], unit10.gravp[0],
        unit10.gravp[0], unit10.gravp[0], unit10.gravp[0], unit10.gravp[0],
        unit10.gravp[0]])

ax.plot(Xnn,Ynn,Znn)

ax.set_xlabel('\n \n \n Frequency \n ', fontsize = 15)
ax.set_zlabel('\n \n \n \n Electromagnetic \n indicator \n ', fontsize = 15)
ax.set_ylabel('\n \n \n Gravity \n indicator\n ', fontsize = 15)

ax.text2D(0.2, 0.95,
          "Interrelation of frequency, gravity and \n electromagnetism in proton",
          transform=ax.transAxes, fontsize = 16)

# The relationship of wavelength, gravity and electromagnetism in a pseudo neutron

fig = plt.figure(figsize=plt.figaspect(0.3))

ax = fig.add_subplot(1, 2, 1, projection='3d')
```

```
Xnn = ([unit4.comptonlpsn[0], unit4.comptonlpsn[1], unit4.comptonlpsn[2],
        unit4.comptonlpsn[3], unit4.comptonlpsn[4], unit4.comptonlpsn[5],
        unit4.comptonlpsn[6], unit4.comptonlpsn[7], unit4.comptonlpsn[8]])
Ynn = ([unit8.elektromagnetikpsn[0], unit8.elektromagnetikpsn[1],
        unit8.elektromagnetikpsn[2], unit8.elektromagnetikpsn[3],
        unit8.elektromagnetikpsn[4], unit8.elektromagnetikpsn[5],
        unit8.elektromagnetikpsn[6], unit8.elektromagnetikpsn[7],
        unit8.elektromagnetikpsn[8]])
Znn = ([unit12.gravpsn[0], unit12.gravpsn[0], unit12.gravpsn[0],
        unit12.gravpsn[0], unit12.gravpsn[0], unit12.gravpsn[0],
        unit12.gravpsn[0], unit12.gravpsn[0], unit12.gravpsn[0]])

ax.plot(Xnn,Ynn,Znn)

ax.set_xlabel('\n \n \n Compton wavelength \n ', fontsize = 15)
ax.set_zlabel('\n \n \n \n Electromagnetic \n indicator \n ', fontsize = 15)
ax.set_ylabel('\n \n \n Gravity \n indicator\n ', fontsize = 15)

ax.text2D(0.2, 0.95,
          "The relationship of wavelength, gravity and \n electromagnetism in a pseudo neutron",
          transform=ax.transAxes, fontsize = 16)

# The relationship of wavelength, gravity and electromagnetism in a pseudo proton, 3D graf

ax = fig.add_subplot(1, 2, 2, projection='3d')

Xnn = ([unit5.comptonlpsp[0], unit5.comptonlpsp[1], unit5.comptonlpsp[2],
        unit5.comptonlpsp[3], unit5.comptonlpsp[4], unit5.comptonlpsp[5],
        unit5.comptonlpsp[6], unit5.comptonlpsp[7], unit5.comptonlpsp[8]])
Ynn = ([unit9.elektromagnetikpsp[0], unit9.elektromagnetikpsp[1],
        unit9.elektromagnetikpsp[2], unit9.elektromagnetikpsp[3],
        unit9.elektromagnetikpsp[4], unit9.elektromagnetikpsp[5],
        unit9.elektromagnetikpsp[6], unit9.elektromagnetikpsp[7],
        unit9.elektromagnetikpsp[8]])
Znn = ([unit13.gravpsp[0], unit13.gravpsp[0], unit13.gravpsp[0],
        unit13.gravpsp[0], unit13.gravpsp[0], unit13.gravpsp[0],
        unit13.gravpsp[0], unit13.gravpsp[0], unit13.gravpsp[0]])

ax.plot(Xnn,Ynn,Znn)

ax.set_xlabel('\n \n \n Compton wavelength \n ', fontsize = 15)
ax.set_zlabel('\n \n \n \n Electromagnetic \n indicator \n ', fontsize = 15)
ax.set_ylabel('\n \n \n Gravity \n indicator\n ', fontsize = 15)

ax.text2D(0.2, 0.95,
          "The relationship of wavelength, gravity and \n electromagnetism in a pseudo proton",
          transform=ax.transAxes, fontsize = 16)

# Interrelation of frequency, gravity and electromagnetism in pseudo neutron, 3D graf

fig = plt.figure(figsize=plt.figaspect(0.3))

ax = fig.add_subplot(1, 2, 1, projection='3d')

Xnn = ([unit16.frequencepsn[0], unit16.frequencepsn[1], unit16.frequencepsn[2],
        unit16.frequencepsn[3], unit16.frequencepsn[4], unit16.frequencepsn[5],
        unit16.frequencepsn[6], unit16.frequencepsn[7], unit16.frequencepsn[8]])
Ynn = ([unit8.elektromagnetikpsn[0], unit8.elektromagnetikpsn[1],
        unit8.elektromagnetikpsn[2], unit8.elektromagnetikpsn[3],
        unit8.elektromagnetikpsn[4], unit8.elektromagnetikpsn[5],
        unit8.elektromagnetikpsn[6], unit8.elektromagnetikpsn[7],
        unit8.elektromagnetikpsn[8]])
Znn = ([unit12.gravpsn[0], unit12.gravpsn[0], unit12.gravpsn[0], unit12.gravpsn[0],
        unit12.gravpsn[0], unit12.gravpsn[0], unit12.gravpsn[0], unit12.gravpsn[0],
        unit12.gravpsn[0]])
```

```
ax.plot(Xnn,Ynn,Znn)

ax.set_xlabel('\n \n \n Frequency \n ', fontsize = 15)
ax.set_zlabel('\n \n \n \n Electromagnetic \n indicator \n ', fontsize = 15)
ax.set_ylabel('\n \n \n Gravity \n indicator\n ', fontsize = 15)

ax.text2D(0.2, 0.95,
          "Interrelation of frequency, gravity and \n electromagnetism in pseudo neutron",
          transform=ax.transAxes, fontsize = 16)

# Interrelation of frequency, gravity and electromagnetism in pseudo proton, 3D graf

ax = fig.add_subplot(1, 2, 2, projection='3d')

Xnn = ([unit17.frequencepsp[0], unit17.frequencepsp[1], unit17.frequencepsp[2],
        unit17.frequencepsp[3], unit17.frequencepsp[4], unit17.frequencepsp[5],
        unit17.frequencepsp[6], unit17.frequencepsp[7], unit17.frequencepsp[8]])
Ynn = ([unit9.elektromagnetikpsp[0], unit9.elektromagnetikpsp[1],
        unit9.elektromagnetikpsp[2], unit9.elektromagnetikpsp[3],
        unit9.elektromagnetikpsp[4], unit9.elektromagnetikpsp[5],
        unit9.elektromagnetikpsp[6], unit9.elektromagnetikpsp[7],
        unit9.elektromagnetikpsp[8]])
Znn = ([unit13.gravpsp[0], unit13.gravpsp[0], unit13.gravpsp[0], unit13.gravpsp[0],
        unit13.gravpsp[0], unit13.gravpsp[0], unit13.gravpsp[0], unit13.gravpsp[0],
        unit13.gravpsp[0]])

ax.plot(Xnn,Ynn,Znn)

ax.set_xlabel('\n \n \n Frequency \n ', fontsize = 15)
ax.set_zlabel('\n \n \n \n Electromagnetic \n indicator \n ', fontsize = 15)
ax.set_ylabel('\n \n \n Gravity \n indicator\n ', fontsize = 15)

ax.text2D(0.2, 0.95,
          "Interrelation of frequency, gravity and \n electromagnetism in pseudo proton",
          transform=ax.transAxes, fontsize = 16)

# The cycle of charge distribution over shells in a free psneutron and psproton, 2D graph

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8])

# psneutron free state
y = np.array([psneutrons[0].charge, psneutrons[1].charge, psneutrons[2].charge,
              psneutrons[3].charge, psneutrons[4].charge, psneutrons[5].charge,
              psneutrons[6].charge, psneutrons[7].charge, psneutrons[8].charge])

# psproton free state

z = np.array([psprotons[0].charge, psprotons[1].charge, psprotons[2].charge,
              psprotons[3].charge, psprotons[4].charge, psprotons[5].charge,
              psprotons[6].charge, psprotons[7].charge, psprotons[8].charge])

xx = np.linspace(x.min(),x.max(), 1000)
fig, axs = plt.subplots(1, 1, figsize=(14, 11))

itp1 = PchipInterpolator(x,y)

window_size, poly_order = 57, 2

yy_sg = savgol_filter(itp1(xx), window_size, poly_order)


axs.plot(x, y, 'gs', label= 'The charge distribution in a free psneutron over shells')

axs.plot(xx, yy_sg, 'green', label= "Smoothed curve")
```

```
itp2 = PchipInterpolator(x,z)

window_size, poly_order = 57, 2

zz_sg = savgol_filter(itp2(xx), window_size, poly_order)


axs.plot(x, z, 'bs', label= 'The charge distribution in a free psproton over shells')

axs.plot(xx, zz_sg, 'b', label= "Smoothed curve")

# or fit to a global function
def func(x, A, B, x0, sigma):
    return abs(A)+B*np.tanh((x-x0)/sigma)

fit, _ = curve_fit(func, x, y)
yy_fit = func(xx, *fit)


axs.plot(xx, yy_fit, 'g--', label=r"$f(xn) = |A| + B \tanh\left(\frac{x-x_0}{\sigma}\right)$")

plt.ylabel('The amount of charge \n \n in Cl x E-19', fontsize=15)

plt.xlabel('Shell number', fontsize=15)

yticks(fontsize=12)

plt.title('THE CHARGE DISTRIBUTION OVER SHELLS IN A FREE PSNEUTRON and PSPROTON \n',
          fontsize=17)
grid(True)
plt.legend(loc='upper left', fontsize=16)

# The cycle of charge distribution over shells in a free neutron and proton, 2D graph

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8])

# neutron free state
y22 = np.array([neutrons[0].charge, neutrons[1].charge, neutrons[2].charge,
                neutrons[3].charge, neutrons[4].charge, neutrons[5].charge,
                neutrons[6].charge, neutrons[7].charge, neutrons[8].charge])

# proton free state
z22 = np.array([protons[0].charge, protons[1].charge, protons[2].charge, protons[3].charge,
                protons[4].charge, protons[5].charge, protons[6].charge, protons[7].charge,
                protons[8].charge])

xx = np.linspace(x.min(),x.max(), 1000)
fig, axs = plt.subplots(1, 1, figsize=(14, 11))

itp1 = PchipInterpolator(x,y22)
itp2 = PchipInterpolator(x,z22)
window_size, poly_order = 57, 2

y22y22_sg = savgol_filter(itp1(xx), window_size, poly_order)
z22z22_sg = savgol_filter(itp2(xx), window_size, poly_order)

axs.plot(x, y22, 'gs', label= 'The charge distribution in a free neutron over shells')

axs.plot(xx, y22y22_sg, 'green', label= "Smoothed curve")

axs.plot(x, z22, 'bs', label= 'The charge distribution in a free proton over shells')
axs.plot(xx, z22z22_sg, 'b', label= "Smoothed curve")

# or fit to a global function
```

```
def func(x, A, B, x0, sigma):
    return abs(A)+B*np.tanh((x-x0)/sigma)

fit, _ = curve_fit(func, x, y22)
y22y22_fit = func(xx, *fit)

fit, _ = curve_fit(func, x, z22)
z22z22_fit = func(xx, *fit)

axs.plot(xx, y22y22_fit, 'g--',
         label=r"$f(xn) = |A| + B \tanh\left(\frac{x-x_0}{\sigma}\right)$")

axs.plot(xx, z22z22_fit, 'b--',
         label=r"$f(xp) = |A| + B \tanh\left(\frac{x-x_0}{\sigma}\right)$")

plt.ylabel('The amount of charge \n \n in Cl x E-20', fontsize=15)

plt.xlabel('Shell number', fontsize=15)

yticks(fontsize=12)

plt.title('THE CHARGE DISTRIBUTION OVER SHELLS IN A FREE NEUTRON AND PROTON \n',
          fontsize=17)
grid(True)

print('\n Significant comments')
print(table1, "\n")

print("\n Values of quarks 'u' and 'd' by \n"
      "shells in Qe (electron charges) \n")
print(table)

print("   ", '\n Values of quarks "u" by shells \n')
print(table2)

print("   ", '\n Values of quarks "d" by shells \n')
print(table3)

print('\n Detailed description for proton by shells \n')
print(table4)

print('\n Detailed description for neutron, by shells \n')
print(table5)

print("\n Values of quarks 'u' and 'd' by \n"
      "shells in Qe (electron charges) \n"
     "for pseudo particles")

print(pstable)

print("   ", '\n Values of quarks "u" by shells for pseudo particles \n')

print(pstable2)

print("   ", '\n Values of quarks "d" by shells for pseudo particles \n')

print(pstable3)

print('\n Detailed description for pseudo proton by shells \n')

print(pstable4)

print('\n Detailed description for pseudo neutron by shells \n')

print(pstable5)
```

```
plt.legend(loc='upper left', fontsize=16)

plt.show()
```