DAFTAR PUSTAKA

[1] M. Ferdows and F. Alzahrani, Study of non-isothermal incompressible flow and heat flux of nano-ferrofluid with induced magnetic induction, International Communications in Heat and Mass Transfer 109 (2019) 104352.

[2] J. Chen, S. Qi, X. Hong, P. Gu, R. Wei, C. Tang, Y. Huang, C. Zhao, High sensitive 3D metamaterial sensor based on diffraction coupling of magnetic plasmon resonances, Results in Physics 15 (2019) 102791.

[3] S. Halder, S. Bhuyan, S. N. Das, S. Sahoo, R. N. P. Choudhary, P. Das, K. Parlda, Structural, morphological, dielectric and impedance spectroscopy of lead-free $Bi(Zn_{2/3}Ta_{1/3})O_3$ electronic material, Applied Physics A 123, 781 (2017).

[4] K. Deb, A. Bera, K. L. Bhowmik, B. Saha, Conductive polyaniline on paper as a flexible electronic material with controlled physical properties through vapor phase polymerization, Polymer Engineering and Science 58 (12) (2018).

[5] P. Porfyrakis and N. L. Tsitsas, Nonlinear electromagnetic metamaterials: Aspects on mathematical modeling and physical phenomena, Microelectronic Engineering 216 (2019) 111028.

[6] T. S. Pham, H. N. Bui, J. W. Lee, Wave propogation control and switching for wireless power transfer using tunable 2-D magnetic metamaterials, Journal of Magnetism and Magnetic Materials, 485 (2019) 126-135.

[7] J. Chen, S. Qi, X. Hong, P. Gu, R. Wei, C. Tang, Y. Huang, C. Zhao, Highly sensitive 3D metamaterial sensor based on diffraction coupling of magnetic plasmon resonance, Results in Physics 15 (2019) 102791.

[8] Z. Hu, J. Kanagaraj, H. Hong, K. Yang, X. Ji, F. Q. Fan, Characterization of ferrice magnetic nanopaticle modified polymeric composites by modeling, Journal of Magnetism and Magnetic Materials 493 (2020) 165735.

[9] W. Xia, J. Lu, S. Tan, J. Liu, Z. Zhang, Chapt: Manipulating dielectric properties by modifying molecular structure of polymers. Dielectric Polymer Materials for High-Density Energy Storage, William Andrew Applied Science Publisher and Elsevier, China, 2018.

[10] A. Asenjo-Garcia, A. Manjavacas, V. Myroshnychenko, F. J. Garcia de Abajo, Magnetic polarization in the optical absorption of metallic nanoparticles, Optics Express 20 (27) (2012) 28142-28152.

[11] A. Deviles, X. Zambrana-Puyalto, B. Stout, N. Bonod, Mimicking localized surface plasmons with dielectric particles, Physical Review B 92 (2015) 241412.

[12] V. Myroshnchenko, J. Rodriguez-Fernandez, I. Pastoriza-Santos, A. M. Funston, C. Novo, P. Mulvaney, L. M. Liz-Marzan, F. J. Garcia de Abajo, Modelling the optical response of gold nanoparticle, Chemical Society Reviews 37 (2008) 1792-1805.

[13] D. Tzarouchis and A. Sihvola, Light scattering by dielectric sphere: Perspectives on the Mie resonance, Applied Science 8 (2) (2018) 184.

[14] H. Bech, A. Leder, Two-particle characterization by pulse induced and time resolved Mie scattering, Optik 122 (2011) 37-43.

[15] Z. Hu, J. Kanagaraj, H. Hong, K. Yang, X. Ji, Q. H. Fan, P. Kharel, Characterization of ferrite magnetic nanoparticle modified polymeric composites by modeling, Journal of Magnetism and Magnetic Materials 493 (2020) 165735.

[16] J. Aizpurua, P. Hanarp, D. S. Sutherland, M. Kaill, G. W. Bryant, F. J. G. de Abajo, Optical properties of gold nanorings, Physical Review Letters 90 (5) (2003) 057401.

[17] K. Bi, L. Zeng, H. Chen, C. Fang, Q. Wang, M. Lei, Magnetic coupling effect of Mie resonance-based metamaterial with inclution of split ring resonators, Journal of Alloys and Compounds 646 (2015) 680-684.

[18] W. Hu, N. Yi, S. Sun, L. Cui, Q. Song, S. Xiao, Enhancement of magnetic dipole emmision at yellow light in optical metamaterials, Optics Communication 350 (2015) 202-206.

[19] W. Li, J. Wei, W. Wang, D. Hu, Y. Li, J. Guan, Ferrite-based metamaterial microwave absorber with absorption frequency magnetically tunable in a wide range, Materials and Design 110 (2016) 27-34.

[20] H. Jin, G. Lin, L. Bai, M. Amjad, E. P. B. Filho, D. Wen, Photothermal conversion efficiency of nanofluids: Aan experimental and numerical study, Solar Energy 139 (2016) 278-289.

[21] J. Navarette, C. Siefe, S. Alcantar, M. Belt, G. D. Stucky, M. Moskovits, Merely measuring the UV-Visible spectrum of gold nanoparticles can change their charge state, Nano Letters 18 (2) (2018) 669-674.

[22] H. Yan, X. Song, X. Wang, Y. Wang, Electromagnetic wave absorption and scattering analysis for Fe_3O_4 with different scales particles, Chemical Physics Letters 723 (2019) 51-56.

[23] F. Monticone and A. Alu, Metamaterial, plasmonic, nanophotonic devices, Reports on Progress in Physics 80 (2017) 036401.

[24] M. Vos dan P. L. Grande, Simple model dielectric functions for insulators, Journal of Physical and Chemistry of Solids 104 (2017) 192-197.

[25] P. Porfyrakis, N. L.Tsitsas, Nonlinear electromagnetic metamaterials: Aspects on mathematical modeling and physical phenomena, Microelectronic Engineering 216 (2019) 111028.

[26] I. V. Fedorova, S. V. Eliseeva, D. I. Sementsov, Spectral and polarization properties of planar multiferroic structure, Optics Communication 458 (2020) 124881.

[27] J. Fiedler, P. Thiyam, A. Kurumbail, F. A. Burger, M. Walter, C. Persson, I. Brevik, D. F. Parsons, M. Bostrom, S. Y. Buhmann, Effective polarizability models, The Journal of Physical Chemistry A 121 (2017) 9742-9751.

[28] G. C. Psarras. Chapt: Fundamentals of dielectric theories. Dielectric Polymer Materials for High-Density Energy Storage, William Andrew Applied Science Publisher and Elsevier, Greece, 2018.

[29] M. Nieto-Vesperinas. Chapt: Fundamentals of Mie scattering. Dielectric Metamaterials: Fundamentals, Designs, and Applications, Woodhead Publishing and Elsevier, United Kingdom, 2020.

[30] I. S. Burkhanov, S. V. Krivokhizha, L. L. Chaikov, Stokes and anti-stokes stimulated Mie scattering on nanoparticle suspensions of latex, Optics Communications 381 (2016) 360-364.

LAMPIRAN

Baris Perintah

1. Persamaan

```
import numpy as np


# Medan Listrik

def link(X, Y, A, B):
    link = ((X-A)**2+(Y-B)**2)**0.5
    return link

def Medan(m,l,N):
    x_axis = np.linspace(m,l,N)
    y_axis = np.linspace(m,l,N)
    X, Y = np.meshgrid(x_axis, y_axis)
    return X, Y

def MedanListrik(Q, R, X, Y, A, B, eps0, l, Rc1, Rc2):
    link = ((X-A) ** 2 + (Y-B) ** 2) ** 0.5
    e = (Q / (4 * np.pi * eps0 * link **2)) * (10**-12.8)
    e[np.where(link < R)] = 0
    e[np.where(e < -l)] = -l
    # e[np.where(e > l)] = l
    ee = abs((e*(Y-B) / (link))) * -1
    ee[np.where(link < R)] = 0
    return e, ee, link


#########################################################################
## Fungsi Dielektrik

def PermP(wP, eps_inf,p, c, lmd):
    h = 4.135*10**-15
    hc = h/(2*np.pi)
    L = lmd * 10 ** -9
    w = (2 * np.pi * c) / (L)
    R = ((eps_inf)*(w**2)-
((wP/hc)**2)+(eps_inf)*((p/hc)**2))/((w**2)+((p/hc)**2))
    I = (((wP/hc)**2)*(p/hc))/((w**3)+(w*(p/hc)**2))
    return complex(R, I)

## Energy Loss Function

def ELF(R,I):
    elf = R / ((R ** 2) + (I ** 2))
    return elf


#########################################################################
# Polarizabilitas dengan pendekatan Rayleigh

def PolarE(R, epsP, eps_m):
    r = R*10**-9
    ae = 4 * np.pi * eps_m * ((r**3)*(epsP-1))/(epsP+2)
    return ae
```

```
def PolarB(R, epsP, lmd):
    L = lmd * 10 ** -9
    r = R * 10 ** -9
    am = ((r**3)*((r/L)**2)*(2*(np.pi**2)/15)*(epsP-1))
    return am


########################################################################
# Polarizabilitas
## Untuk bilangan kuantum orbital = 1

def p_pm(R, l, eps, eps_m):
    r = R * 10**-9
    lmd = l * 10**-9
    p = (2 * np.pi * r / lmd) * (np.sqrt(eps))
    pm = (2 * np.pi * r / lmd) * (np.sqrt(eps_m))
    return p, pm

## Fungsi Bessel
def j(p, pm):
    jm = (np.sin(pm) / pm ** 2) - (np.cos(pm) / pm)
    jp = (np.sin(p) / p ** 2) - (np.cos(p) / p)
    jd = ((((p ** 2) * np.cos(p)) - (2 * p * np.sin(p))) / (p ** 4)) +
(((p * np.sin(p)) + np.cos(p)) / (p ** 2))
    jmd = ((((pm ** 2) * np.cos(pm)) - (2 * pm * np.sin(pm))) / (pm **
4)) + (
             ((pm * np.sin(pm)) + np.cos(pm)) / (pm ** 2))
    return jm, jp, jd, jmd

## Fungsi Henkel
def h(pm):
    hmR = ((np.cos(pm) + pm * np.sin(pm)) / pm**2)
    hmI = ((np.sin(pm) - pm * np.cos(pm)) / pm**2)
    hm = complex(hmR, hmI)
    hmdR = (((np.cos(pm) * ((pm**3) - 2*pm)) - (2 * (pm**2) *
np.sin(pm))) / pm ** 4)
    hmdI = (((2 * (pm**2) * np.cos(pm)) + (np.sin(pm) * ((pm**3) -
2*pm))) / pm**4)
    hmd = complex(hmdR,hmdI)
    return hm, hmd

## Konstanta Mie (elektrik)
def tE(eps, eps_m, p, pm, jp, jd, jm, jmd, hm, hmd):
    a = - (eps_m * jm * (jp + p * jd)) + (eps * (jm + pm * jmd) *jp)
    b = (eps_m * hm * (jp + p * jd)) - eps * (hm + pm * hmd) * jp
    tE = (a/b)
    return tE

## Konstanta Mie (Magnetik)
def tM(p, pm, jm, jp, jd, jmd, hm, hmd):
    a = - (p * jm * jd) + (pm * jmd * jp)
    b = (p * hm * jd) - (pm * hmd * jp)
    tM = (a/b)
    return tM

## Polarizabilitas
def alpa(l, tE, tM):
    lmd = l * (10 ** -9)
    aE = (3/2) * ((lmd / (2 * np.pi)) ** 3) * tE
    aM = (3/2) * ((lmd / (2 * np.pi)) ** 3) * tM
    return aE, aM

## total cross-section
```

```
### Konstanta Mie dalam imaginer
def cross_sec(L, eps_m, tE, tM, n):
    l = L * 10 ** -9
    l_m = l / (np.sqrt(eps_m))
    a = ((l_m**2) / (2 * np.pi))
    b1 = []
    for i in range(0, n+1, 1):
        B = (2*i + 1) * (tE + tM)
        b1.append(B)
    b = sum(b1)
    c = a * b
    return c


######################################################################
# Model Polarisabilitas
## Excess polarizabilities
def Models(eps, eps_0, eps_m, alpa, r, rc):
    R = r * (10 ** -9)
    Rc = rc * (10 ** -9)
    Virt = (((eps_m + 2) / 3) ** 2) * alpa
    Ons = (((3 * eps_m) / (1 + (2 * eps_m))) ** 2) * alpa
    HS = 4 * np.pi * eps_0 * eps_m * (R ** 3) * ((eps - eps_m) / (eps
+ (2 * eps_m)))
    C = 4 * np.pi * eps_0 * eps_m * (Rc ** 3) * ((1 - eps_m) / (1 + (2
* eps_m)))
    # s = 4 * np.pi * eps_0 * (R ** 3) * ((eps - 1) / (eps + 2))
    s = alpa
    S = s * (((3 * eps_m) / ((2 * eps_m) + 1)) ** 2) * (
            1 / (1 + ((C * s) / (8 * (np.pi ** 2) * (eps_0 ** 2) * (Rc
** 6) * eps_m))))
    FS = C + S
    return Virt, Ons, HS, C, FS
```

2. Polarisabilitas

```
import numpy as np
import matplotlib.pyplot as plt
import Equations as Eq


def l(l_awal, l_akhir, c):

    l = np.linspace(l_awal, l_akhir, (l_akhir - l_awal) + 1)
    lmd = l * 10 ** -9
    h = 4.135 * 10 ** -15
    hc = h / (2 * np.pi)
    E = (hc * c) / (lmd)

    return lmd, E

######################################################################
def P(l_awal, l_akhir, c, wp, w_inf, L, R, eps_medium):

    Er = []
    Ei = []
    AeR = []
    AeI = []
    aeR = []
    aeI = []
    Mr = []
    Mi = []
    AmR = []
```

```
    AmI = []
    amR = []
    amI = []
    am_ae_I = []
    am_ae_R = []

    ELF = []

    for i in range(l_awal, l_akhir+1, 1):
        eps = Eq.PermP(wp, w_inf, L, c, i)
        elf = Eq.ELF(eps.real, eps.imag)

        p1, pm1 = Eq.p_pm(R, i, eps, eps_medium)
        jm, jp, jd, jmd = Eq.j(p1, pm1)
        hm, hmd = Eq.h(pm1)
        e_Ray = Eq.PolarE(R, eps, 1)
        m_Ray = Eq.PolarB(R, eps, i)
        e = Eq.tE(eps, eps_medium, p1, pm1, jp, jd, jm, jmd, hm, hmd)
        m = Eq.tM(p1, pm1, jm, jp, jd, jmd, hm, hmd)
        aE, aM = Eq.alpa(i, e, m)

        aM_aE_I = aM.imag / aE.imag
        aM_aE_R = aM.real / aE.real

        Er.append(e.real)
        Ei.append(e.imag)
        AeR.append(e_Ray.real)
        AeI.append(e_Ray.imag)
        aeR.append(aE.real)
        aeI.append(aE.imag)
        Mr.append(m.real)
        Mi.append(m.imag)
        AmR.append(m_Ray.real)
        AmI.append(m_Ray.imag)
        amR.append(aM.real)
        amI.append(aM.imag)
        am_ae_I.append(aM_aE_I)
        am_ae_R.append(aM_aE_R)

        ELF.append(elf)

    return Er, Ei, AeR, AeI, aeR, aeI, Mr, Mi, AmR, AmI, amR, amI,
am_ae_I, am_ae_R, ELF


######################################################################
## Cross-Section

def CS(l_awal, l_akhir, c, wp, w_inf, eps_medium, L, R, l):

    cs = []

    for i in range(l_awal, l_akhir+1, 1):
        eps = Eq.PermP(wp, w_inf, L, c, i)
        p1, pm1 = Eq.p_pm(R, i, eps, eps_medium)
        jm, jp, jd, jmd = Eq.j(p1, pm1)
        hm, hmd = Eq.h(pm1)

        e = Eq.tE(eps, eps_medium, p1, pm1, jp, jd, jm, jmd, hm, hmd)
        m = Eq.tM(p1, pm1, jm, jp, jd, jmd, hm, hmd)

        C = Eq.cross_sec(i, eps_medium, e.imag, m.imag, l)
        cs.append(C)
```

```
    return cs


#########################################################################
## Model Polarisabilitas

def Models(l_awal, l_akhir, c, wp_m, wp_p, w_inf, L_m, L_p, R, Rc,
eps_medium):

    VC_R = []
    VC_I = []
    OC_R = []
    OC_I = []
    HSC_R = []
    HSC_I = []
    CC = []
    FSC_R = []
    FSC_I = []

    for i in range(l_awal, l_akhir+1, 1):
        eps_p = Eq.PermP(wp_p, w_inf, L_p, c, i)
        eps_m = Eq.PermP(wp_m, w_inf, L_m, c, i)
        p1, pm1 = Eq.p_pm(R, i, eps_p, eps_m)
        jm, jp, jd, jmd = Eq.j(p1, pm1)
        hm, hmd = Eq.h(pm1)

        e = Eq.tE(eps_p, eps_m, p1, pm1, jp, jd, jm, jmd, hm, hmd)
        m = Eq.tM(p1, pm1, jm, jp, jd, jmd, hm, hmd)

        aE, aM = Eq.alpa(i, e, m)
        Virt, Ons, HS, C, FS = Eq.Models(eps_p, eps_medium, eps_m, aE,
R, Rc)

        VC_R.append(Virt.real)
        VC_I.append(Virt.imag)
        OC_R.append(Ons.real)
        OC_I.append(Ons.imag)
        HSC_R.append(HS.real)
        HSC_I.append(HS.imag)
        CC.append(C)
        FSC_R.append(FS.real)
        FSC_I.append(FS.imag)

    return VC_R, VC_I, OC_R, OC_I, HSC_R, HSC_I, CC, FSC_R, FSC_I
```

3. Plot

```
import numpy as np
import matplotlib.pyplot as plt

def Plot(x, y, i):
    plt.plot(x, y, alpha=0.7, linewidth=2, label= 'R = ' + str(i) + '
nm')
    # plt.yscale("log")
    # plt.xscale("log")
    plt.legend()

def Plot_L(a, E, e, R, labelx, labely):
    for i in range(0, a, 1):
        Plot(E, e[i], R[i])
    plt.xlabel(labelx)
```

```
    plt.ylabel(labely)
    plt.show()


def Plot_log_M(x, y, y1 = None, y2 = None, y3 = None, y4 = None):
    plt.plot(x, y, alpha=0.7, linewidth=2, label=r'$ \alpha $')
    if y1 is not None:
        plt.plot(x, y1, alpha=0.7, linewidth=2, label='Virtual
Cavity')
    if y2 is not None:
        plt.plot(x, y2, alpha=0.7, linewidth=2, label='Real Cavity')
    if y3 is not None:
        plt.plot(x, y3, alpha=0.7, linewidth=2, label='Hard Sphere')
    if y4 is not None:
        plt.plot(x, y4, alpha=0.7, linewidth=2, label='Finite Size')

    plt.yscale("log")
    # plt.xscale("log")
    plt.legend()
    plt.xlabel('Energy (eV)')
    plt.ylabel('log ' + r'$ \alpha $' + 'Im{E}')
    plt.show()


def Plot_M(x, y, y1 = None, y2 = None, y3 = None, y4 = None):
    plt.plot(x, y, alpha=0.7, linewidth=2, label=r'$ \alpha $')
    if y1 is not None:
        plt.plot(x, y1, alpha=0.7, linewidth=2, label='Virtual
Cavity')
    if y2 is not None:
        plt.plot(x, y2, alpha=0.7, linewidth=2, label='Real Cavity')
    if y3 is not None:
        plt.plot(x, y3, alpha=0.7, linewidth=2, label='Hard Sphere')
    if y4 is not None:
        plt.plot(x, y4, alpha=0.7, linewidth=2, label='Finite Size')

    plt.legend()
    plt.xlabel('Energy (eV)')
    plt.ylabel(r'$ \alpha $' + 'Im{E}')
    plt.show()
```

4. Grafik

```
import numpy as np
import matplotlib.pyplot as plt
import Equations as Eq
import Polar as Pl
import Plot as plot
from pandas import DataFrame

c = 3 * 10 ** 8
eps0 = 1
wp_g = 8.89
wp_s = 9.04
w_inf = 1
L_g = 0.071
L_s = 0.021
l_awal = 120
l_akhir = 500
r_mula = 10
r_akhir = 50
selisih = 10
```

```
inc = int(((r_akhir+selisih) - r_mula) / selisih)


#########################################################################
# Polarisabilitas

R = []
er = []
ei = []
aER = []
aEI = []
ER_R = []
EI_R = []
mr = []
mi = []
aMR = []
aMI = []
MR_R = []
MI_R = []
aM_aE_I = []
aM_aE_R = []
ELF = []

for r in range(r_mula, r_akhir+selisih, selisih):
    I = r
    Er, Ei, AeR, AeI, aeR, aeI, Mr, Mi, AmR, AmI, amR, amI, am_ae_I,
am_ae_R, elf = Pl.P(l_awal, l_akhir, c, wp_g, w_inf, L_g, r, eps0)
    # Er, Ei, AeR, AeI, aeR, aeI, Mr, Mi, AmR, AmI, amR, amI, am_ae,
am_ae_2, elf = Pl.P(l_awal, l_akhir, c, wp_s, w_inf, L_s, r, eps0)
    R.append(I)
    er.append(Er)
    ei.append(Ei)
    aER.append(aeR)
    aEI.append(aeI)
    ER_R.append(AeR)
    EI_R.append(AeI)
    mr.append(Mr)
    mi.append(Mi)
    aMR.append(amR)
    aMI.append(amI)
    MR_R.append(AmR)
    MI_R.append(AmI)
    aM_aE_I.append(am_ae_I)
    aM_aE_R.append(am_ae_R)

    ELF.append(elf)

l, E = Pl.l(l_awal, l_akhir, c)


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#Data

El = {'Energy' : E,
        'aER1' : aER[0],
        'aER2' : aER[1],
        'aER3' : aER[2],
        'aER4' : aER[3],
        'aER5' : aER[4],
        'aEI1' : aEI[0],
        'aEI2' : aEI[1],
        'aEI3' : aEI[2],
        'aEI4' : aEI[3],
```

```
        'aEI5' : aEI[4]}

Ma = {'Energy' : E,
      'aM_aE_R1' : aM_aE_R[0],
      'aM_aE_R2' : aM_aE_R[1],
      'aM_aE_R3' : aM_aE_R[2],
      'aM_aE_R4' : aM_aE_R[3],
      'aM_aE_R5' : aM_aE_R[4],
      'aM_aE_I1' : aM_aE_I[0],
      'aM_aE_I2' : aM_aE_I[1],
      'aM_aE_I3' : aM_aE_I[2],
      'aM_aE_I4' : aM_aE_I[3],
      'aM_aE_I5' : aM_aE_I[4]
        }

el = DataFrame(El, columns= ['Energy', 'aER1', 'aER2', 'aER3', 'aER4',
'aER5', 'aEI1', 'aEI2', 'aEI3', 'aEI4', 'aEI5'])

ma = DataFrame(Ma, columns= ['Energy', 'aM_aE_R1', 'aM_aE_R2',
'aM_aE_R3', 'aM_aE_R4', 'aM_aE_R5',
                             'aM_aE_I1', 'aM_aE_I2', 'aM_aE_I3',
'aM_aE_I4', 'aM_aE_I5'])

export_excel = el.to_excel (r'Polarizability_e.xlsx', index=None,
header=True)
export_excel = ma.to_excel (r'Polarizability_m..e.xlsx', index=None,
header=True)

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Plot

## Polarizabilitas dengan pendekatan Rayleigh
plot.Plot_L(inc, E, ER_R, R, 'Energy (eV)', 'ER_Rayleigh')
plot.Plot_L(inc, E, EI_R, R, 'Energy (eV)', 'EI_Rayleigh')
plot.Plot_L(inc, E, MR_R, R, 'Energy (eV)', 'ER_Rayleigh')
plot.Plot_L(inc, E, MI_R, R, 'Energy (eV)', 'EI_Rayleigh')

## Konstanta Mie
plot.Plot_L(inc, E, er, R, 'Energy (eV)', 'C_ER')
plot.Plot_L(inc, E, ei, R, 'Energy (eV)', 'C_EI')
plot.Plot_L(inc, E, mr, R, 'Energy (eV)', 'C_ER')
plot.Plot_L(inc, E, mi, R, 'Energy (eV)', 'C_EI')

## Polarizabilitas
plot.Plot_L(inc, E, aER, R, 'Energy (eV)', 'Re' + '{' + r'$ \alpha $'
+ 'E' + '}')
plot.Plot_L(inc, E, aEI, R, 'Energy (eV)', 'Im' + '{' + r'$ \alpha $'
+ 'E' + '}')
plot.Plot_L(inc, E, aMR, R, 'Energy (eV)', 'Re' + '{' + r'$ \alpha $'
+ 'M' + '}')
plot.Plot_L(inc, E, aMI, R, 'Energy (eV)', 'Im' + '{' + r'$ \alpha $'
+ 'M' + '}')

plot.Plot_L(inc, E, aM_aE_R, R, 'Energy (eV)', 'Re' + '{' + r'$ \alpha
$' + 'M' + '}' + ' / ' + 'Re' +
            '{' + r'$ \alpha $' + 'E' + '}')
plot.Plot_L(inc, E, aM_aE_I, R, 'Energy (eV)', 'Im' + '{' + r'$ \alpha
$' + 'M' + '}' + ' / ' + 'Im' +
            '{' + r'$ \alpha $' + 'E' + '}')
plot.Plot_L(inc, E, ELF, R, 'Energy (eV)', 'Im (- (1 / Re (eps)))')


#####################################################################
```

```
# Cross-Section

C_S = []
R = []

for r in range(r_mula, r_akhir+selisih, selisih):
    I = r
    cs = Pl.CS(l_awal, l_akhir, c, wp_g, w_inf, eps0, L_g, r, 1)
    R.append(I)
    C_S.append(cs)

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#Data
cs = {'Energy' : E,
        'CS1' : C_S[0],
        'CS2' : C_S[1],
        'CS3' : C_S[2],
        'CS4' : C_S[3],
        'CS5' : C_S[4],
      }

Cs = DataFrame(cs, columns= ['Energy', 'CS1', 'CS2', 'CS3', 'CS4',
'CS5'])

export_excel = Cs.to_excel (r'Cross Section.xlsx', index=None,
header=True)

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
## Plot

plot.Plot_L(inc, E, C_S, R, 'Energy (eV)', 'Total Cross Section')

#########################################################################
###
# Model Polarisabilitas listrik

wp_m = wp_s
wp_p = wp_g
L_m = L_s
L_p = L_g

R = []
VCM_R = []
VCM_I = []
OM_R = []
OM_I = []
HSM_R = []
HSM_I = []
C = []
FSM_R = []
FSM_I = []

for r in range(r_mula, r_akhir+selisih, selisih):
    I = r
    Rc = r + (10 ** -9)
    VC_R, VC_I, OC_R, OC_I, HSC_R, HSC_I, CC, FSC_R, FSC_I =
Pl.Models(l_awal, l_akhir, c, wp_m, wp_p, w_inf, L_m, L_p, r, Rc,
eps0)
    R.append(I)
    VCM_R.append(VC_R)
    VCM_I.append(VC_I)
    OM_R.append(OC_R)
    OM_I.append(OC_I)
```

```
    HSM_R.append(HSC_R)
    HSM_I.append(HSC_I)
    C.append(CC)
    FSM_R.append(FSC_R)
    FSM_I.append(FSC_I)


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
## By One
r = 50
rc = r + 1

Er, Ei, AeR, AeI, aeR, aeI, Mr, Mi, AmR, AmI, amR, amI, am_ae_I,
am_ae_R, elf = Pl.P(l_awal, l_akhir, c, wp_g, w_inf, L_g, r, eps0)
VC_R, VC_I, OC_R, OC_I, HSC_R, HSC_I, CC, FSC_R, FSC_I =
Pl.Models(l_awal, l_akhir, c, wp_m, wp_p, w_inf, L_m, L_p, r, rc,
eps0)

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#Data

vc = {'Energy' : E,
        'VCMR1' : VCM_R[0],
        'VCMR2' : VCM_R[1],
        'VCMR3' : VCM_R[2],
        'VCMR4' : VCM_R[3],
        'VCMR5' : VCM_R[4],
        'VCMI1' : VCM_I[0],
        'VCMI2' : VCM_I[1],
        'VCMI3' : VCM_I[2],
        'VCMI4' : VCM_I[3],
        'VCMI5' : VCM_I[4]
        }
##
om = {'Energy' : E,
      'OMR1' : OM_R[0],
      'OMR2' : OM_R[1],
      'OMR3' : OM_R[2],
      'OMR4' : OM_R[3],
      'OMR5' : OM_R[4],
      'OMI1' : OM_I[0],
      'OMI2' : OM_I[1],
      'OMI3' : OM_I[2],
      'OMI4' : OM_I[3],
      'OMI5' : OM_I[4]
       }
##
hs = {'Energy' : E,
        'HSMR1' : HSM_R[0],
        'HSMR2' : HSM_R[1],
        'HSMR3' : HSM_R[2],
        'HSMR4' : HSM_R[3],
        'HSMR5' : HSM_R[4],
        'HSMI1' : HSM_I[0],
        'HSMI2' : HSM_I[1],
        'HSMI3' : HSM_I[2],
        'HSMI4' : HSM_I[3],
        'HSMI5' : HSM_I[4]
        }
##
fs = {'Energy' : E,
        'FSMR1' : FSM_R[0],
        'FSMR2' : FSM_R[1],
```

```
        'FSMR3' : FSM_R[2],
        'FSMR4' : FSM_R[3],
        'FSMR5' : FSM_R[4],
        'FSMI1' : FSM_I[0],
        'FSMI2' : FSM_I[1],
        'FSMI3' : FSM_I[2],
        'FSMI4' : FSM_I[3],
        'FSMI5' : FSM_I[4]
        }


vcm = DataFrame(vc, columns= ['Energy', 'VCMR1', 'VCMR2', 'VCMR3',
'VCMR4', 'VCMR5',
                              'VCMI1', 'VCMI2', 'VCMI3', 'VCMI4',
'VCMI5'])
oms = DataFrame(om, columns= ['Energy', 'OMR1', 'OMR2', 'OMR3',
'OMR4', 'OMR5',
                              'OMI1', 'OMI2', 'OMI3', 'OMI4', 'OMI5'])
hsm = DataFrame(hs, columns= ['Energy', 'HSMR1', 'HSMR2', 'HSMR3',
'HSMR4', 'HSMR5',
                              'HSMI1', 'HSMI2', 'HSMI3', 'HSMI4',
'HSMI5'])
fsm = DataFrame(fs, columns= ['Energy', 'FSMR1', 'FSMR2', 'FSMR3',
'FSMR4', 'FSMR5',
                              'FSMI1', 'FSMI2', 'FSMI3', 'FSMI4',
'FSMI5'])
#
export_excel = vcm.to_excel (r'VCM.xlsx', index=None, header=True)
export_excel = oms.to_excel (r'OMS.xlsx', index=None, header=True)
export_excel = hsm.to_excel (r'HSM.xlsx', index=None, header=True)
export_excel = fsm.to_excel (r'FSM.xlsx', index=None, header=True)


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
## Plot

plot.Plot_L(inc, E, VCM_R, R, 'Energy (eV)', 'Re {' + r'$ \alpha $' +
'E' + '}')
plot.Plot_L(inc, E, OM_R, R, 'Energy (eV)', 'Re {' + r'$ \alpha $' +
'E' + '}')
plot.Plot_L(inc, E, HSM_R, R, 'Energy (eV)', 'Re {' + r'$ \alpha $' +
'E' + '}')
plot.Plot_L(inc, E, FSM_R, R, 'Energy (eV)', 'Re {' + r'$ \alpha $' +
'E' + '}')

plot.Plot_L(inc, E, VCM_I, R, 'Energy (eV)', 'Im {' + r'$ \alpha $' +
'E' + '}')
plot.Plot_L(inc, E, OM_I, R, 'Energy (eV)', 'Im {' + r'$ \alpha $' +
'E' + '}')
plot.Plot_L(inc, E, HSM_I, R, 'Energy (eV)', 'Im {' + r'$ \alpha $' +
'E' + '}')
plot.Plot_L(inc, E, FSM_I, R, 'Energy (eV)', 'Im {' + r'$ \alpha $' +
'E' + '}')

plot.Plot_log_M(E, aeI, VC_I, OC_I, HSC_I, FSC_I)
```

5. Medan 2D

```
import numpy as np
import matplotlib.pyplot as plt
```

```
import Equations as Eq
from pandas import DataFrame
import pandas as pd

r = 2*10**-8
rc1 = 2.1*10**-8
rc2 = 2.3*10**-8
eps0 = 8.85 * 10 **(-12)
q = (-1.602 *10**(-19))
m = 0
l = 10**-6.5
L = 1**-6.5
N = 1000
A, B = (l/2,l/2)

xx,yy = Eq.Medan(-l,l,N)

X, Y = Eq.Medan(m,l,N)
e, ee, link = Eq.MedanListrik(q, r, X, Y, A, B, eps0, l, rc1, rc2)
E = ee+yy
E[np.where(E > l)] = l
E[np.where(E < -l)] = -l

fig = plt.figure(facecolor='w')
ax = fig.add_subplot(1, 1, 1)
ax.set_aspect('equal')
lvl = np.linspace(-l,l,1000)
E = ax.contourf(X, Y, ee, levels = lvl, cmap= 'jet')

#Linkaran
sphere = np.linspace(0, 2*np.pi, 100)
xsp = r*np.cos(sphere)+A
ysp = r*np.sin(sphere)+B
ax.plot(xsp,ysp,'k-',lw=2)

ax.set_xlabel('$y$ (m)')
ax.set_ylabel('$x$ (m)')

plt.show()
```