```
#
# Reproducing "A Computer Code for High Altitude EMP" thesis AFIT
# by Terry C. Chapman, Capt., USAF
# Jan 1974
# Author: lagridcoalition.org
# Last Modified: 30 Aug 2016
#
# X,Y,Z is the Target location in meters.
#       for  the northern hemisphere
#       X is the magnetic west
#       Y is the magnetic south
#       Z is altitude
# HoB is the height of the burst in kilometers > 50 km
# GammaYld is gamma yield of burst in kilotons
# Bfield is the magnitude of Earth's magnetic field in the
#   absorption region below the burst in Webers/m^2
# Bangle is the dip angle of the magnetic field in degrees
# Nsteps is the number of steps Between Rmin and Rmax    50 <= Nsteps <=
500
#

import numpy as np
import math


# Confirm that GoB is > 50km
def HoBGet():
    HoB = eval(input("What is the height of the burst in kilometers? "))
    if int(HoB) < 50:
        print("Model has burst above absoprtion region, >50km.")
        HoBGet()
    return HoB

def OtherInputsGet():
    GammaYld = eval(input("Gamma Yield of Burst in kilotons: "))
    Bfield = eval(input("Magnitude of Earth's magnetic field in absorption
region: "))
    Bangle = eval(input("Dip angle 'A' of the magnetic field in degrees:
"))
    Nsteps = eval(input("The number of steps Between Rmin and Rmax. 50 <=
Nsteps <= 500: "))
    return GammaYld, Bfield, Bangle, Nsteps

def PositionGet():
    X = eval(input("Enter target location X: "))
```

```python
    Y = eval(input("Enter target location Y: "))
    Z = eval(input("Enter target height Z: "))
    return X, Y, Z

def Efun(R, E, COMPTJ, Sigma):
    # Constants
    c = 3.0e8
    RMUO = 12.56637e-7
    Efun = -(1/R+c*RMUO*Sigma/2)*E-COMPTJ*c*RMUO/2
    return Efun

# E(I+1) is calculated from E(I)
# using the runge-kutta method
def RNGKUT(E, R, H, Sigma, COMPTJ):
    RK1 = H*Efun(R, E, COMPTJ, Sigma)
    RK2 = H*Efun(R+H/2, E+RK1/2, COMPTJ, Sigma)
    RK3 = H*Efun(R+H/2, E+RK2/2, COMPTJ, Sigma)
    RK4 = H*Efun(R+H, E+RK3, COMPTJ, Sigma)
    E1 = E + (RK1+2*(RK2+RK3)+RK4)/6
    return E1

# Calculates F(T) for Runge-Kutta
# integration of Primary Electrons
def RKCMTN(R, Theta, omega, TP, TPRIME, A):
    SOLVE = ToFT(TP, TPRIME, Theta, omega)
    RKCMTN = FoFT(SOLVE, A, TP, 1)
    return RKCMTN

# Solves virgin transport and uses reaction rate to
# calculate the number density of racial electrons
def GOFR(R, A, HoB, PATH, GammaYld):
    SOLVE = (0.0226275/math.cos(A))*(-
1+math.exp(R*math.cos(A)/7000))*math.exp(-HoB/7000)
    Denom = 12.56637*R*R*PATH*1.5
    GOFR = math.exp(-SOLVE)*GammaYld/Denom
    return GOFR

# Calculates Compton lifetime at Radius = R
# Max acceptable lifetime = 100 shakes for
# the Karzas-latter high frequency approximation
def CLIFE(R, A, HoB):
    CLIFE = 1.041667e-8*math.exp((HoB-R*math.cos(A))/7000)
    if CLIFE > 1e-6:
        CLIFE = 1e-6
    return CLIFE
```

```python
def FoFT(T, A, To, Rn):
    B=0.958
    Tshake=T*1e8
    Denom = (8+A*math.exp((A+B)*(Tshake-To)))*Rn
    FOFT=(A+B)*math.exp(A*(Tshake-To))/Denom
    return FOFT

# T(T) is Time Transformed to Karzas-Latter Form
def ToFT(T, Tprime, Theta, omega):
    B=0.958
    First = T-(-B*math.pow(math.cos(Theta),2))*Tprime
    Second = B*math.pow(math.sin(Theta), 2)*math.sin(omega*Tprime)/omega
    TOFT = First+Second
    return TOFT

# Calculate F(T,R) for Runge-Kutta integration of Phi
# Component of Compton Current
def CMPhi(HoB, R, A, Theta, omega, PATH, T, TPRIME, GammaYld):
    SOLVE = ToFT(T, TPRIME, Theta, omega)
    SOLVE = FoFT(SOLVE, A, T, 0.5)
    SOLVE = SOLVE*(-4.608e-11)*GOFR(R, A, HoB, PATH, GammaYld)
    CMPhi = SOLVE*math.sin(Theta)*math.sin(omega*TPRIME)
    return CMPhi

# Calculate F(T,R) for Runge-Kutta Integration
# of theta component of comption current
def CMTHET(HoB, R, A, Theta, omega, PATH, T, TPRIME, GammaYld):
    SOLVE = ToFT(T, TPRIME, Theta, omega)
    SOLVE = FoFT(SOLVE, A, T, R)
    SOLVE = SOLVE*(-4.608e-11)*GOFR(R, A, HoB, PATH, GammaYld)
    CMTHET =
SOLVE*math.sin(Theta)*math.cos(Theta)*(math.cos(omega*TPRIME)-1)
    return CMTHET

#Calculates sigma after finding nsecondary from nprimary
def CONDCT(Sigma, Pri, DTP, DT, HoB, R, A, k, Store1, Nsteps, Pri2):
    # Stores1 contains integral for negative Tau
    # Stores2 contains integral for positive Tau
    Store2 = np.zeros(Nsteps)
    COLISN = 4e2*math.exp((R*math.cos(A)-HoB)/7000)
    Store1 = Store1 - Pri*DTP
    Store2[k] = Store2[k] + Pri2*DT*(1.0e-9)
    SEC = Store2[k] - Store1
    Sigma = (1.6e-19**2)*SEC/(COLISN*9.11e-31)
    return Sigma

# Calculate the Two Components of the compton current at given T and R
# Calculate number of Primary electrons
# JTHETA    is the theta component of compton current
# JPHI      is phi component of compton current
# TMAX      is compton lifetime
# PATH      is altitude scaled compton mean free path
# TP        is negative retarded time
# T         is positive retarded time
# PRI       is number of primary electrons generated during TP
# PRI2      is number of primary electrons generated during T
# TPRIME    is variable of integration
def COMPTN(T, R, A, Theta, omega, HoB, GammaYld, TP):
    # Initialize constants
    JTHETA = 0.0
    JPHI   = 0.0
    Tprime = 5e-9
    TMAX = CLIFE(R, A, HoB)
    PRI = 0.0
    DT = TMAX/10
    PATH = 309*math.exp((HoB-R*math.cos(A))/7000)
    TPRIME = DT
    W = DT/2
    PRI2 = 0.0
    # Runge-Kutta integration of compton current
    for i in range(1, 10):
        RK1 = DT*CMTHET(HoB, R, A, Theta, omega, PATH, T, TPRIME,
GammaYld)
        RK2 = DT*CMTHET(HoB, R, A, Theta, omega, PATH, T, TPRIME+W,
GammaYld)
        RK3 = RK2
        RK4 = DT*CMTHET(HoB, R, A, Theta, omega, PATH, T, TPRIME+DT,
GammaYld)
        JTHETA = JTHETA+(RK1+2*(RK2+RK3)+RK4)/6
        RK5 = DT*CMPhi(HoB, R, A, Theta, omega, PATH, T, TPRIME, GammaYld)
        RK6 = DT*CMPhi(HoB, R, A, Theta, omega, PATH, T, TPRIME+W,
GammaYld)
        RK7 = RK6
        RK8 = DT*CMPhi(HoB, R, A, Theta, omega, PATH, T, TPRIME+DT,
GammaYld)
        JPHI = JPHI + (RK5+2*(RK6+RK7)+RK8)/6
        # Runge-Kutta integration of Primaries
        RKP1 = DT*RKCMTN(R, Theta, omega, TP, TPRIME, A)
        RKP2 = DT*RKCMTN(R, Theta, omega, TP, TPRIME + W, A)
        RKP3 = RKP2
        RKP4 = DT*RKCMTN(R, Theta, omega, TP, TPRIME + DT, A)
        PRI = PRI + (RKP1+2*(RKP2+RKP3)+RKP4)/6
        RP1 = DT*RKCMTN(R, Theta, omega, T, TPRIME, A)
        RP2 = DT*RKCMTN(R, Theta, omega, T, TPRIME+W, A)
        RP3 = RP2
        RP4 = DT * RKCMTN(R, Theta, omega, T, TPRIME+DT, A)
        PRI2 = PRI2 + (RP1+2*(RP2+RP3)+RP4)/6
        TPRIME = TPRIME + DT
    # Multiply primaries by Q*G(R)/Tmax
    # to obtain rate of production of secondaries
    PRI = PRI*5e4*GOFR(R, A, HoB, PATH, GammaYld)/TMAX
    PRI2 = PRI2*5e4*GOFR(R, A, HoB, PATH, GammaYld)/TMAX
    return PRI, PRI2, JTHETA, JPHI

# Calculate the E-Fields in the absorption region
def EField(Rmin, Rmax, Nsteps, HoB, A, Theta, omega, GammaYld):
    # Fortran declares a 3d array... E(190), TIME(190), STORES(Nsteps)
###190???
    #efield_array=np.zeros((E[190], TIME[190], STORES[Nsteps]))
    E = np.zeros(190)
    Time = np.zeros(190)
    Stores = np.zeros(190)
    # iter is time of iteration in shakes 10<=Iter<=100
    # Read iter and change it to number of time steps
    Iter=100 #???
    Iter=100+(Iter-10)
    # Initialize arrays and constants
    for j in range(0,101):
        E[j] = 0.0
        Time[j] = 0.1*j
    for j in range(101, 190):
        E[j] = 0.0
        Time[j] =10.0+(j-100)
    for l in range(Nsteps):
        Stores[l]=0.0
    ETHE=0.0
    DELRN=Nsteps
    T=0
    DT=1.0
    EPHI = 0.0
    Deltar = (Rmax-Rmin)/DELRN
    R = Rmin + Deltar
    RNP = 1e-8*1 #1e-8*RNP How???....
    # Start integrations, outside loop is for calculation in retarded time
    # inside loop is for integration in at each time step
    for i in range(190):
        T = T + (1.0e-9)*DT
        Time[i] =  T * (1.0e8)
        IT = i
        if i > Iter:
            # Go to 42
            print("201: "+Time[Iter])
        TP = - Deltar/2.88e8
        DTP = TP
        Sigma = 0.0
        Store1 = 0.0
        # DO 31 K=1, NDLER
        for k in range(1, Nsteps):
            Pri, Pri2, JTHETA, JPHI = COMPTN(T, R, A, Theta, omega, HoB,
GammaYld, TP)
            Sigma = CONDCT(Sigma, Pri, DTP, DT, HoB, R, A, k, Store1,
Nsteps, Pri2)
            ETHENW = RNGKUT( ETHE, R, Deltar, Sigma, JTHETA)
            EPHINW = RNGKUT( EPHI, R, Deltar, Sigma, JPHI)
            ETHE = ETHENW
            EPHI = EPHINW
            R = R + Deltar
            TP = TP + DTP

        # Find magnitude of Efield
        E[i] = math.sqrt(math.pow(ETHE,2)+math.pow(EPHI,2))

        # Check for Divergence of Solution
        if E[i] > 1e15: # Go to 52
            # Print 301
            # Print 201, Time[IT]
            print("Solution has gone unstable")
            print(E[i])
            print("Iteration Terminated After: "+str(i))
            if IT < 10:
                return
        if i >= 100:
            DT = 10
        R = Rmin + Deltar
        print(str(i)+" "+ str(Time[i])+" "+  str(E[i])+" "+ str(Sigma))
    return max(E)

def main():
    #Use default values?
    Q1 = input("Use Default values, (Y, N)? ")
    if Q1 == "Y":
        print("Using default values")
        #    Getting input data
        HoB=100
        X, Y, Z = 0, 0, 0
        print("X=Y=Z=0")
        #Get other inputs
        GammaYld, Bfield, Bangle, Nsteps = 0.001, 0.00002, 40, 100
        print("GammaYld, Bfield, Bangle, Nsteps = 0.001, 0.00002, 20, 50")
    if Q1 == "N":
        print("Please enter values you would like to use.")
        #    Getting input data
        HoB=HoBGet()
        #print(HoB)
        X, Y, Z = PositionGet()
        #Get other inputs
        GammaYld, Bfield, Bangle, Nsteps = OtherInputsGet()
    #Calculate the distance from burst to Target
    R=math.sqrt(X*X+Y*Y+math.pow(HoB*1000-Z,2))
    #print inputs
    print("Burst Location: "+str(HoB)+ "km")
    print("Target Location: "+str(X)+", "+str(Y)+", "+str(Z))
    print("Distance between Targert and Burst: "+str(R)+"m")
    #   Convert Data to MKS Unsits
    HoB = HoB * 1000
    GammaYld = 2.61625e25*GammaYld
    Bangle = 0.017453295*Bangle
    omega = 1.6e-19*Bfield/(3.505*9.11e-31)

#### Print Type of Calcuation. Reflected wave calculation assumes 100%
Reflection
#### and uses mirror image of target below ground set Z=-Z if reflective
wave is to be used
    REFLCT = 49000.0
    if Z > REFLCT:
        print("2007: Target is above absorption region so reflected wave
is being calculated")
        Z=-Z
    if Z < 0:
        print("2008: Reflected wave is being calculated")
    if Z <= REFLCT and Z > 0:
        print("2009: Direct wave is being calculated")
    if Z > HoB-1000:
        print("2007: Target is above absorption region so reflected wave 
is being calculated")
        Z=-Z
#   Determine Angles
    A=math.acos((HoB-Z)/R)
    Theta=math.acos(math.cos(Bangle)*Y/R+math.sin(Bangle)*(Z-HoB)/R)
    print(A, Theta)

#   Calculating Rmin and Rmax
    ZRmin = 5e4
    if HoB < ZRmin:
        ZRmin = HoB
    TA = (ZRmin-HoB)/(Z-HoB)
    XRmin = TA*X
    YRmin = TA*Y
    Rmin = math.sqrt(math.pow(XRmin, 2)+math.pow(YRmin, 2)+math.pow(ZRmin-
HoB, 2))
    ZRmax = Z
    if Z < 2e4:
        ZRmax = 2e4
    TB = (ZRmax-HoB)/(Z-HoB)
    XRmax = TB*X
    YRmax = TB*Y
    Rmax = math.sqrt(math.pow(XRmax,2)+math.pow(YRmax,2)+math.pow((ZRmax-
HoB),2))
    print(Rmax)

# Calculate Efield at Bottom of Absorption Region
    ef = EField(Rmin, Rmax, Nsteps, HoB, A, Theta, omega, GammaYld)
    print("Peak Efield at Target is "+str(ef)+" Volts/Meter")

####
    # Calculate Cycoltron Frequency of Electrons ??????


main()
```