Tillegg C

Python kode

Programmer som er brukt for å plotte bølgefunksjoner og annet. De fleste skulle kunne tilpasses etter behov med å taste inn relevante parametre øverst i koden.

C.0.1 Verdenslinjer

```
import scipy
import numpy as np
import matplotlib.pyplot as plt
import math
# Her kan man plotte en verdenslinje for en relativistisk tur
# og finne tider for mottak av meldinger for turen
# Vi setter lyshastigheten lik 1 i dette programmet
# Hastighet til objekt som reiser ut og hjem
beta = 0.5
gamma = 1/np.sqrt(1.-beta*beta)
print(" beta ",beta," gamma ",gamma)
# Returnering ved første glimt etter dette tidspunktet
rt = 5.
# Signalfrekvens i rakettens hvilesystem til hjemsendte signaler
f0 = 1.5



# Dopplerforskjøvede frekvenser
f1 = np.sqrt((1+beta)/(1-beta))*f0
f2 = np.sqrt((1-beta)/(1+beta))*f0
dT0 = 1/f0
dT1 = 1/f1
dT2 = 1/f2

# Tid mellom hvert signal sendt fra romskip, som observert hjemme
dTlab = dT0*gamma
Tlab = 0.

# Plottegreier
# Ytterkanter av figuren
XHI = 6.
XLO = -1.
YLO = 0.
YHI = 12.
plt.axis([XLO,XHI,YLO,YHI])
plt.xlabel('x')
plt.ylabel('ct')
#linja ct = x
```

```
x1=[0.,10.]
y1=[0.,10.]
plt.plot(x1,y1,'ro--',linewidth=0.5,markersize=.5)
slope = beta
# xtick er tilbakelagt reiselengde i som sett i lab
# ytick er ct i lab
xtick = 0.
ytick = 0.
# tilbakelagt reiselengde mellom hvert blink
dxtick = dTlab*beta
dytick = dTlab

dprimex = []
dprimey = []
# Dopplerformeltid (doptx velges for plassering i plottet
dopt = []
dopx = []
while ytick < rt :
   xtick = xtick + dxtick
   ytick = ytick + dytick
   dprimex.append(xtick)
   dprimey.append(ytick)
   Tlab = Tlab + dT2
   dopt.append(Tlab)
   dopx.append(-.3)
   x1 = [xtick,0.]
   y1 = [ytick,ytick+xtick]
   plt.plot(x1,y1,'ro--',linewidth=0.5,markersize=.5)
snux = slope*ytick
x1=[0.,snux]
snuy = ytick
y1=[0.,snuy]
plt.plot(x1,y1,'k-',linewidth=1.,markersize=1)
while ytick < 2*rt :
   Tlab = Tlab + dT1
   xtick = xtick - dxtick
   ytick = ytick + dytick
   dprimex.append(xtick)
   dprimey.append(ytick)
   dopt.append(Tlab)
   dopx.append(-.3)
```

```
   x1 = [xtick,0.]
   y1 = [ytick,ytick+xtick]
   plt.plot(x1,y1,'ro--',linewidth=0.5,markersize=.5)
plt.plot(dprimex,dprimey,'ko',markersize=3.)
plt.plot(dopx,dopt,'kx',markersize=3.)
x1=[snux,0.]
y1=[snuy,snuy*2]
plt.plot(x1,y1,'k-',linewidth=1.,markersize=1)
#plt.arrow(1.,1.,4.,0.,head_width=.2,head_length=.5,fc='k',ec='k')
#plt.text(4.5,.5,'x',fontsize=sz)



plt.title("Verdenslinjer og Doppler-tider")
plt.grid()
#plt.axis('off')
plt.show()
```

Diffraksjon med beregning av standardavvik

```
#
# Diffraksjon/interferens plott
# Beregner også standardaavik av sannsynlighet i gitt intervall

import numpy as np
import scipy
from scipy.special import *
from scipy.stats import *
import matplotlib.pyplot as plt

def f(x,a,b,bl):
   arg1 = 3.14159*b*np.sin(x)/bl
   diftrm = np.sin(arg1)/arg1
   arg2 = 3.14159*a*np.sin(x)/bl
   inttrm = np.cos(arg2)
   return (diftrm*inttrm)**2

# hovedprogram
#
# Lage x-verdier (vinkler)
xl = -1.1
xh = 1.1
stp = 0.0001
th = np.arange(xl,xh,stp)
npfu = int((xh-xl)/stp) + 1
plt.axis([xl,xh,-.0,1.3]) #nedre og øvre grenser i plott

b = .25  # Bredden til hver spalte
a = 0. # (en spalt)1.  # Avstand mellom spaltene
bl = .25 # Bølgelengde
hbar = 1.
c = 1.
l = 1. # Avstand fra spalt til skjerm
y = f(th,a,b,bl)
#plt.plot(th,y)
xproj = th #l*np.tan(th)
yproj =y   #*np.cos(th)
plt.plot(xproj,yproj)
plt.title("Intensitet")
```

```
plt.xlabel('theta')
plt.ylabel('intensitet')
 #   Integrering er en lek!
intv = scipy.integrate.trapz(yproj,xproj)
print(" Verdi av funksjonens integral i definisjonsområdet er: ",intv )
print(" (trapes-regel med ", npfu,"  punkter)")
middelv = scipy.integrate.trapz(yproj*xproj,xproj)/intv
intv2 = scipy.integrate.trapz(yproj*xproj*xproj,xproj)/intv
sd = np.sqrt(intv2-middelv**2)
print(" Verdi av middelverdi og sd ",middelv, " " , sd )
E = hbar*2*3.14159*c/bl
p = E/c
pt = p*np.sin(sd)
Dx = b/np.sqrt(12)
DpDx=pt*Dx
print(" Usikkerhetsrelasjon, Dp ",pt," Dx ",Dx," DpDx ",DpDx)
plt.show()
```

Diffraksjon og interferens

```
# Diffraksjon og interferens
# Simulering av enkeltfotoner i.h.t. intensitetsfordeling gjennom to spalter
import numpy as np
import scipy
from scipy.special import *
from scipy.stats import *
import matplotlib.pyplot as plt

def f(x,a,b,bl):
   arg1 = 3.14159*b*np.sin(x)/bl
   diftrm = np.sin(arg1)/arg1
   arg2 = 3.14159*a*np.sin(x)/bl
   inttrm = np.cos(arg2)
   return (diftrm*inttrm)**2

# hovedprogram
#
# Lage x-verdier (vinkler)
xl = -1.
xh = 1.
stp = 0.0001
ntries = [100,1000,5000,10000]
sp =[221,222,223,224]
for i in range(4):
   ntreff = 0
   plt.subplot(int(sp[i]))
   plt.axis([xl,xh,-.0,1.]) #nedre og øvre grenser i plott
   thetas = xl + (xh-xl)*np.random.random(int(ntries[i]))
   yl = np.random.random(int(ntries[i]))
   ypos = np.random.random(int(ntries[i]))
   ys = []
   xs = []
   b = .25  # Bredden til hver spalte
   a = 1.  # Avstand mellom spaltene
   bl = .25 # Bølgelengde
   hbar = 1.
   c = 1.
   l = 1. # Avstand fra spalt til skjerm
   for i in range(int(ntries[i])):
```

```
      th = thetas[i]
      y = f(th,a,b,bl)
      yll = yl[i]
      if y > yll:
         ntreff = ntreff +1
         xs.append(th)
         ys.append(ypos[i])
         #print(" success ",th, " ",ypos)
      plt.plot(xs,ys,'o',markersize=.8)
      title0 = " fotoner"
      tit = str(ntreff)+title0
      plt.title(tit)
      plt.xlabel('theta')
      plt.ylabel('y')
plt.show()
```

Bølgepakke

```
"""
Bilde av bølge, og bølgepakke laget med
en bølgetallsvhangig vektfunksjon gjennom Fourier-cosinustransformasjon
som er eksplisitt programmert
"""
import numpy as np
import matplotlib.pyplot as plt
import scipy
from scipy.integrate import *

def gp(p0,pin,sig):
    # Gaussformet vektfunksjon
    g = (1/(np.sqrt(2*3.14159)*sig))*np.exp(-.5*((pin-p0)/sig)**2)
    return g
    # Nedenstående gir uniform fordeling av bølgetall
    #g = []

    #for i in range(len(pin)):
     #  if abs(pin[i]-p0) < sig:
      #   g.append(1.)
       #else:
        # g.append(0.)
    #return g

p0 =20.       # Gjennomsnittlig bølgetall
sig = 4.    # Her er spredningen av bølgetall. Må ses i relasjon til p0
steps = 2000. # Antall skritt i integrasjonene og vektorene


lmda = 2*np.pi/p0
# definere område for integrasjon
xlow = -10*lmda
xhigh = 10*lmda

dx = (xhigh-xlow)/steps
x = np.arange(xlow,xhigh, dx)        # x-array
```

```
plow = p0 - 10.*sig
phigh = p0 + 10.*sig
dp = (phigh-plow)/steps
p = np.arange(plow,phigh,dp)
fx = np.zeros(int(steps))
wx = []
y = gp(p0,p,sig)   # Vektfunksjon i impulsrom
N = scipy.integrate.trapz(y)
y = y/N
for k in range(int(steps)):
   for j in range(int(steps)):
      fx[j] = y[j]*np.cos(x[k]*p[j])
   wx1 = scipy.integrate.trapz(fx)
   wx.append(wx1)


fig, (ax0,ax1,ax2) = plt.subplots(nrows=3,ncols=1)
ax0.plot(x, np.sin(p0*x))
ax1.plot(p, y)
```

Harmonisk oscillator

```
# Plotte egenfunksjoner for Harmonisk oscillatorer
# Sammenligne med klassisk sannsynlighet for å finne partikkel i x
# vi setter hbar=1

import numpy as np
import scipy.special as sp
#import scipy.misc as sm
import matplotlib.pyplot as plt

def f(x,m,omega,n,p):          # Definer funksjonen
    N = 1./np.sqrt((2.**n)*sp.factorial(n))
    xx = x*np.sqrt(m*omega)
    y = (sp.eval_hermite(n,xx)*np.exp(-xx*xx/2.)*N)**p
    #y = np.sin(x)/x
    return y
def cl(x,m,omega,n):
    # Klassisk sannsynlighet for partikkelens posisjon
    # (ikke normert)
    E = (n+.5)*omega
    a = np.sqrt(2*E/m)/omega
    y = 1/(omega*np.sqrt(a**2-x**2))  # Python lar oss fortsette selv om vi iblant h
    return y
#
#  Hovedprogram
#
n = 1  # Kvantetall (som gir energinivå
p = 1   #Bølgefunksjon: p=1, eller sannsynlighetsfordeling (psi**2): p=2
N = 1./np.sqrt((2.**n)*sp.factorial(n))  # Normeringskonstant

m = 1.
omega = 10.
A = np.sqrt(2*(n+.5)/(m*omega))
print(" N ",N, " Amplitude ", A)
fig, (ax1,ax2) = plt.subplots(nrows=2,ncols=1)
xl = -np.sqrt((2*n+1)/omega)
xh = np.sqrt((2*n+1)/omega)
x1 = np.arange(xl*2,xh*2,0.001)
x = np.arange(xl*2,xh*2,0.001)
y = f(x,m,omega,n,p)
```

```
ax1.plot(x,y)

y1 = cl(x1,m,omega,n)
y2 = y**2
ax2.plot(x1,y1,'r--')
ax2.plot(x1,y2)
ax2.axis([xl*2,xh*2,0.,0.3])

plt.show()   # Vise plottet
```

Radielle bølgefunksjoner til hydrogenatomet

```
# Plotte Radielle bølgefunksjoner for Hydrogenatomet

import numpy as np
import scipy
import scipy.special as sp
import scipy.misc as sm
import matplotlib.pyplot as plt

def psi(x,n,l):         # Define your function here
    # Noe er ikke OK med normeringen
    N = 1/sp.factorial(n)
    a = 0.529  # Bohr-radien i Ångstrøm)
    rho = 2*x/(n*a)
    y = N*(rho**l)*np.exp(-.5*rho)*sp.assoc_laguerre(rho,n-l-1,2*l+1)
    return y

#Sett verdier inn her
clor = ['r','b','g','k']
lstyle = ['solid','dashed','dotted','dashdot']

fig, (ax1,ax2) = plt.subplots(nrows=2,ncols=1)
n = 10

for i in range(0,2):
   l = 9*i
   xl = 0.
   # Skala kan defineres utfra forventet midlere avstand
   #  som er .5*(3*n**2 - l**2-l)
   #xh = 3.*n**2
   xh = 150.  # ..eller manuelt
   x = np.arange(xl,xh,0.01)
   y = psi(x,n,l)    # Bølgefunksjonen

   y1 = (x**2)*(y**2)
   intv = np.trapz(y1,x)
   print ("Normering", intv)
   y = y/np.sqrt(intv)
   y1 = y1/intv
```

```
   ax1.plot(x,y,color=clor[i],linestyle=lstyle[i]) # 'make' the plot
   ax2.plot(x,y1,color=clor[i],linestyle=lstyle[i])


   y2 = x*y1/intv
   intv = np.trapz(y2,x)
   print (" forventningsverdi av avstand, <x>= ",intv/.529," Bohrradier")
plt.show()   # Show the plot (Without this you don't see anything on the scre
```

Projiserte tettheter og romlige elektronskyer

```
# sannsynlighet for å finne elektron et bestemt sted i z-x planet
# Etter å ha laget dette så simuleres et antall 'eksperimenter'
# som gir posisjon av et elektron
# i rommet i henhold til sannsynlighetsfunksjonen
import numpy as np
import scipy
import scipy.special as sp
import scipy.misc as sm
import matplotlib.pyplot as plt

def psi(x,n,l):         # Define your function here
    # Normering ikke OK
    N = 1
    a = 0.529  # Bohr-radien i Ångstrøm)
    rho = 2*x/(n*a)
    y = N*(rho**l)*np.exp(-.5*rho)*sp.assoc_laguerre(rho,n-l-1,2*l+1)
    return y
# Her velges kvantetall
values = []
line = input("tast n,l og m :")
values = line.split()
n = int(values[0])
l = int(values[1])
m = int(values[2])
while n>0 :

    valg = 1   # Valg = 1: psi**2    Noe annet:  (r*psi)**2
    nbins = 100
    # Skala defineres utfra forventet midlere avstand
    #  som er .529*(3*n**2 - l**2-l)
    rm = 1.5*n**2
     x = []
    z = []
    w = []
    bw2 = .5*rm/float(nbins)
    wmax = 0.
    for i in range(nbins):
        zz = -rm + 2*i*rm/float(nbins)+bw2
```

```
        for j in range(nbins):
            phi = 0.  # Vi skal alltid kvadrere Ylm, så det er OK å bruke ver
            xx = -rm + 2*j*rm/float(nbins)+bw2
            r = np.sqrt(xx**2+zz**2)
            theta = abs(np.arccos(zz/r))
            wr = psi(r,n,l)
            ww = sp.sph_harm(int(m),int(l),phi,theta)
            if valg == 1:
                weight = (wr*ww.real)**2

            else:
                weight = (r*wr*ww.real)**2
            if weight > wmax:
               wmax = weight
            w.append(weight)
            x.append(xx)
            z.append(zz)
    plt.hist2d(x,z,bins=nbins,weights=w)
    plt.show()
    print("wmax ", wmax) # maksimumsverdi i tetthetsplottet, trengs for MC si
    # her begynner MonteCarlo-simulering
    ntries = 1000000
    npinarray = 0
    xdata = []
    ydata = []
    zdata = []

    trz = np.random.random(ntries)
    trx = np.random.random(ntries)
    tryc = np.random.random(ntries)
    trtry = np.random.random(ntries)*wmax

    for k in range(ntries):
       # lage vilkårlig punkt i rommet
       zz = -rm + 2*trz[k]*rm
       xx = -rm + 2*trx[k]*rm
       yy = -rm + 2*tryc[k]*rm
       r = np.sqrt(xx**2 +yy**2 + zz**2)
       theta = np.arccos(zz/r)
       wr = psi(r,n,l)
       ww = sp.sph_harm(int(m),int(l),0.,theta)
```

```
        if valg == 1:
           weight = (wr*ww.real)**2
        else:
           weight = (r*wr*ww.real)**2
        # sjekke sannsynligheten av dette punktet, akseptere med frekvens i henhold t
        if weight > trtry[k]:
           npinarray = npinarray + 1
           zdata.append(zz)
           xdata.append(xx)
           ydata.append(yy)
    ax = plt.axes(projection='3d')
    ax.scatter3D(xdata,ydata,zdata,s=.01)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    print(" npinarray ",npinarray)
    plt.show()   # Show the plot (Without this you don't see anything on the screen)
    line = input("tast n,l og m :")
    values = line.split()
    n = int(values[0])
    l = int(values[1])
    m = int(values[2])
exit()
```

Kjerneradius

```
# Ladningsfordeling i kjerne

import numpy as np
import scipy
from scipy.linalg import *
#from scipy.special import *
from scipy.stats import *
#import scipy.misc as sm
import matplotlib.pyplot as plt
def norm(r,c,a):
    y = 4*3.1415*r**2/(1+np.exp((r-c)/a))
    return y
def f(r,c,a,N):         # Definer funksjonen
    y = N/(1+np.exp((r-c)/a))
    return y

color = ['r','b','k']
linest =['solid','dashed','dotted']
names =['Al','Fe','Pb']
txtposx=[2.15,4.5,6.5]
txtposy=[.04,.04,.04]
Zs = [13,26,82]
As = [27,56,208]
c1 = 1.07
a = 0.54
xl = 0.
xh = 12.
xhn = 10*xh
x = np.arange(xl,xh,0.01)
xn = np.arange(xl,xhn,.01)
for i in range(3):
   Z = Zs[i]
   A = As[i]
   c = (A**.333333)*c1
   yn = norm(xn,c,a)
   N = scipy.integrate.trapz(yn,xn)
   y = f(x,c,a,Z/N)
   y2 = 4*3.14159*(x**2)*y
   N1 = scipy.integrate.trapz(y2,x)
```

```
   print("A,Z,N = ",A," ",Z," ",N," ",N1)
   plt.text(txtposx[i],txtposy[i],names[i],fontsize=10)
   plt.plot(x,y,color=color[i],linestyle=linest[i])
plt.title("Modell for ladningsfordeling i kjerner")
plt.xlabel(" fm ")
plt.ylabel("$e/fm^{3}$")
plt.show()   # Vise plottet
```