FYS 2140 Hjemmeeksamen

15053

March 23, 2018

Oppgave 1

a1)

$$\lambda_C = \frac{h}{m_e c} \tag{1}$$

$$\begin{aligned} \lambda_C &= comptonblgelengden \\ h &= planckkonstanten \\ m_e &= elektronmassen \\ c &= lyshastighetetn \end{aligned}$$

Forskjellen i bølgelengde gis ved

$$\Delta\lambda = (\lambda' - \lambda_0) = \frac{h}{m_e c}(1 - cos\theta)$$

$$\Delta\lambda = \lambda_C(1 - cos\theta) \tag{2}$$

$$\lambda_C = \frac{h}{m_e c} = \underline{\underline{2.43x10^{-3}nm}} \tag{3}$$

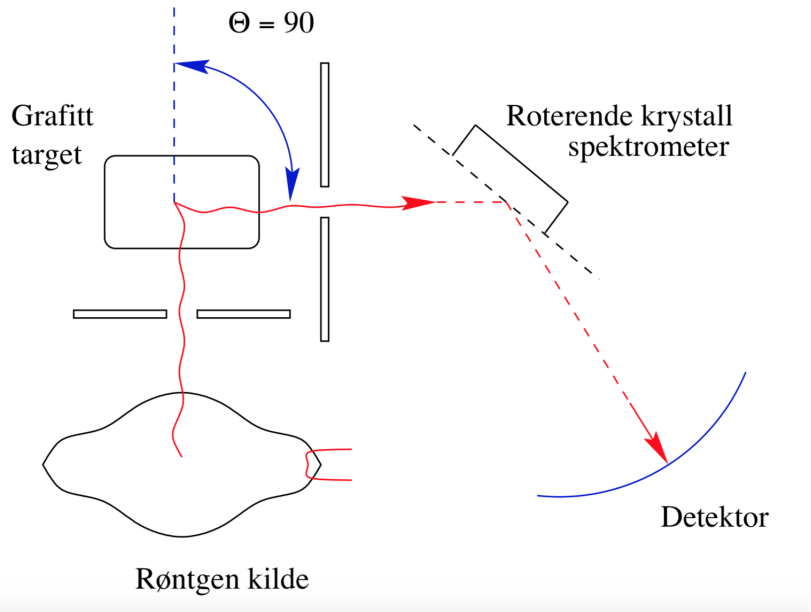

Figure 1: Eksperimentelt oppsett for Comptons forsøk, med $\Theta = 90$ grader. Bilde hentet fra kompendiet

a2)

For Fra formel (2) så har vi

$$\Delta\lambda = \lambda_C(1 - cos\theta)$$

Med verdier

$$\lambda = 0.0709nm$$
$$\lambda' = 0.0749nm$$
$$\Delta\lambda = 0.004nm$$

da får vi

$$\Delta\lambda = \lambda_C(1 - cos\theta)$$
$$\frac{\Delta\lambda}{\lambda_C} = 1 - \cos\theta$$
$$\theta = \cos^{-1}(1 - \frac{\Delta\lambda}{\lambda_C})$$
$$\theta = \cos^{-1}(1 - \frac{0.004nm}{2.43x10^{-3}nm})$$
$$\theta = \underline{\underline{130.5^\circ}}$$

a3)

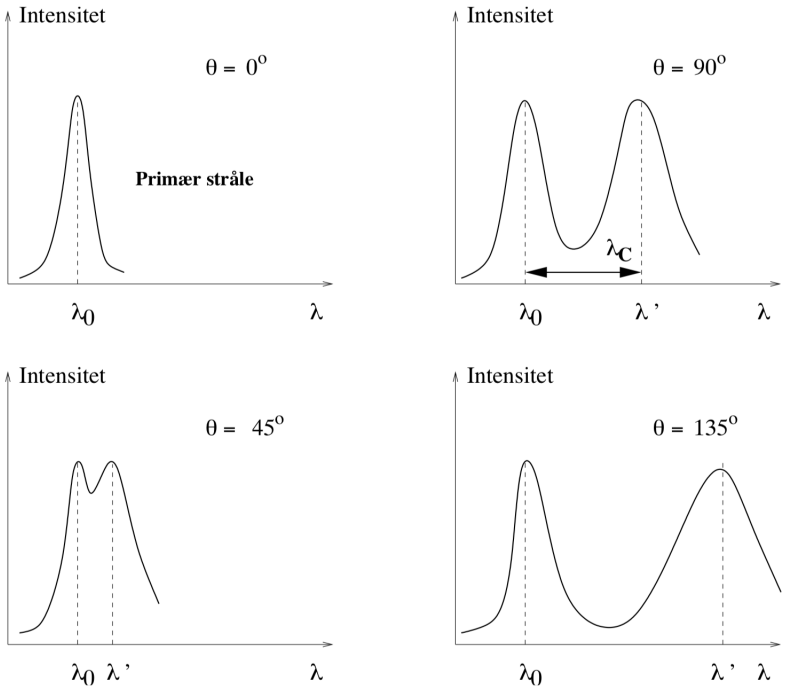

Figure 2: Plot over intensiteten til stråling i Comptons forsøk. λ_0 er den innkommende strålingen, λ' er den spredte strålingen.

Toppen i det første plottet i figur 2 og til venstre i alle de andre plotene i firgur 2 skyldes forskjell i kollisjon. Når vi har en endring i bølgelengde så vekselvirker fotonet med de elektronene som er svakest bundet til atomet, da gjør vi som fysikerer gjør best og forenkler litt og ser på det som et foton

mot et fritt elektron. Men hvis vi tenker at elektroner er sterkt bundet eller at fotonet ikke har nok energi så vil ikke elektronet bli slått løst, dermed blir kollisjonen som en kollisjon mellom et foton og hele atomet. Hvis vi antar at materialet består av karbon så må vi bytte ut massen til elektronet i kollisjonen med massen til et karbonatom. Massen til et karbonatom er ca $22000m_e$. Da blir comptonbølgelengden

$$\lambda_C = \frac{h}{22000m_e c} \approx 10^{-7}nm = 0.1fm \tag{4}$$

Den relative endringen $\frac{\lambda_C}{\lambda}$ blir såvidt observerbar.

Grunnen til av vi ikke bruker synlig lys til comptoneksperimentet er at synlig lys ikke har nok energi. Synlig lys har nok energi til å kunne få til fotoelektrisk effekt, men ikke til comptonspredning.

b1)

Vi setter inn verdiene i formelen, $T = 25°C = 298.15K$

$$\begin{aligned}\langle E\rangle &= k_B T \\ \langle E\rangle &= \frac{3}{2}\cdot 8.61\cdot 10^{-5}eVK^{-1}\cdot 298.15K \\ &= 0.0385 \approx \underline{\underline{38.5\cdot 10^{-3}eV}}\end{aligned}$$

$$\langle p\rangle = \frac{\langle E\rangle}{c} = 38.5\cdot 10^{-3}eV/c \approx 1.28\cdot 10^{-10}eV$$

$$\langle \lambda\rangle = \frac{hc}{\langle E\rangle} = \frac{1240eVnm}{38.5\cdot 10^{-3}eV} = 3.22\cdot 10^{4}nm$$

b2)

Vi bruker Bragg-likningen for n = 1.

$$\lambda = 2d\sin\theta \rightarrow \theta = \sin^{-1}(\frac{\lambda}{2d}) = 19.1° \tag{5}$$

Nøytroner med bølgelengde 1.85 Å har størst intensistet ved 19.1°. For å filtrere ut bølgelengder med spredning mindre enn $\frac{|\delta\lambda|}{\lambda} = 10\%$ så har vi

$$\frac{|\lambda' - \lambda|}{\lambda} = 10\%$$
$$\rightarrow \lambda'_{maks} = 1.05\lambda \; og \; \lambda'_{min} = 0.95\lambda$$
$$\rightarrow \lambda'_{maks} = 1.94 \; og \; \lambda'_{min} = 1.76$$

Bragg-likningen, n=1:

$$\theta_{maks} = \sin^{-1}(\frac{1.94}{2 \cdot 2.82}) = 20.1^\circ$$
$$\theta_{min} = \sin^{-1}(\frac{1.76}{2 \cdot 2.82}) = 18.2^\circ$$

Vår område i vinkler blir da fra 18.2° til 20.1°

Oppgave 2

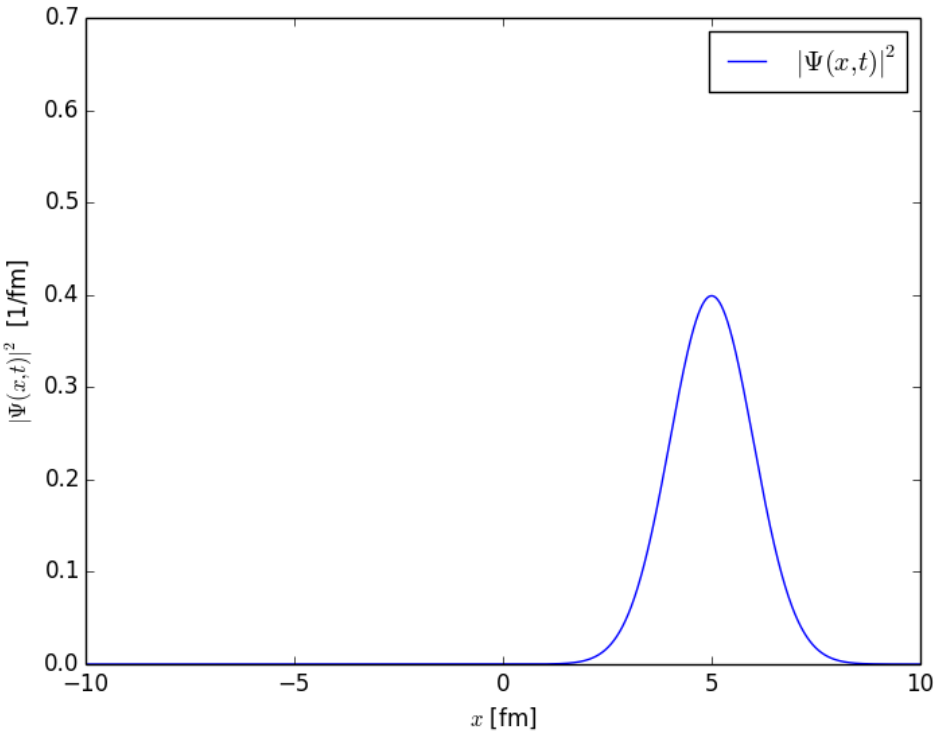

Figure 3: plottet for $\mid \Psi(x,0) \mid^2$

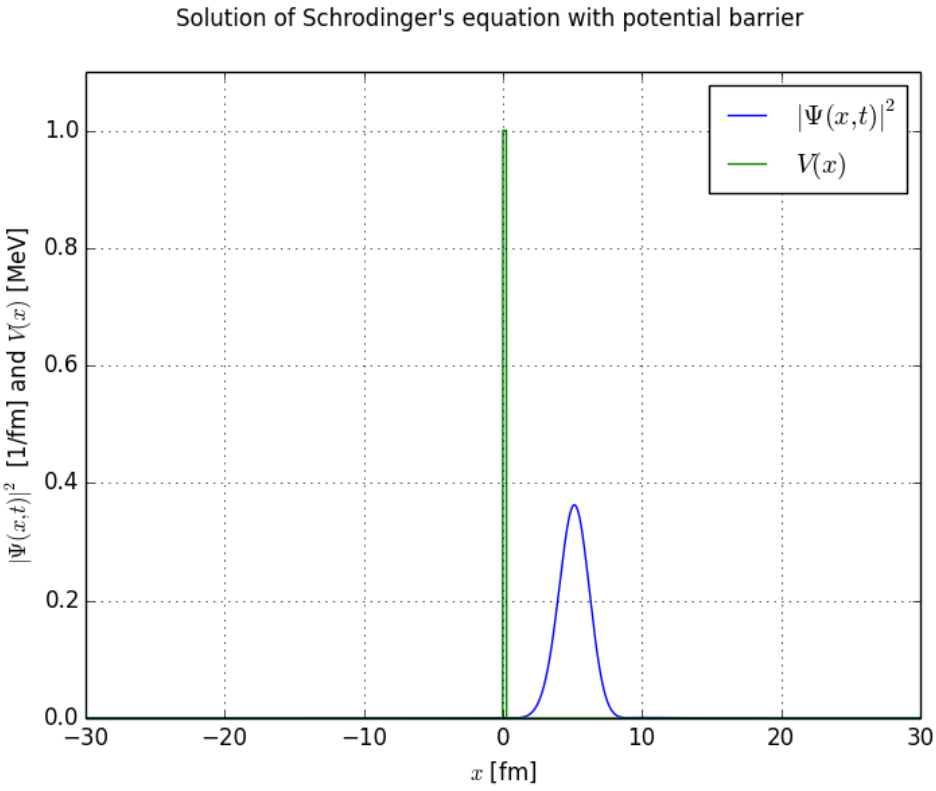

Figure 4: Løsning av likningen med potensialsperre

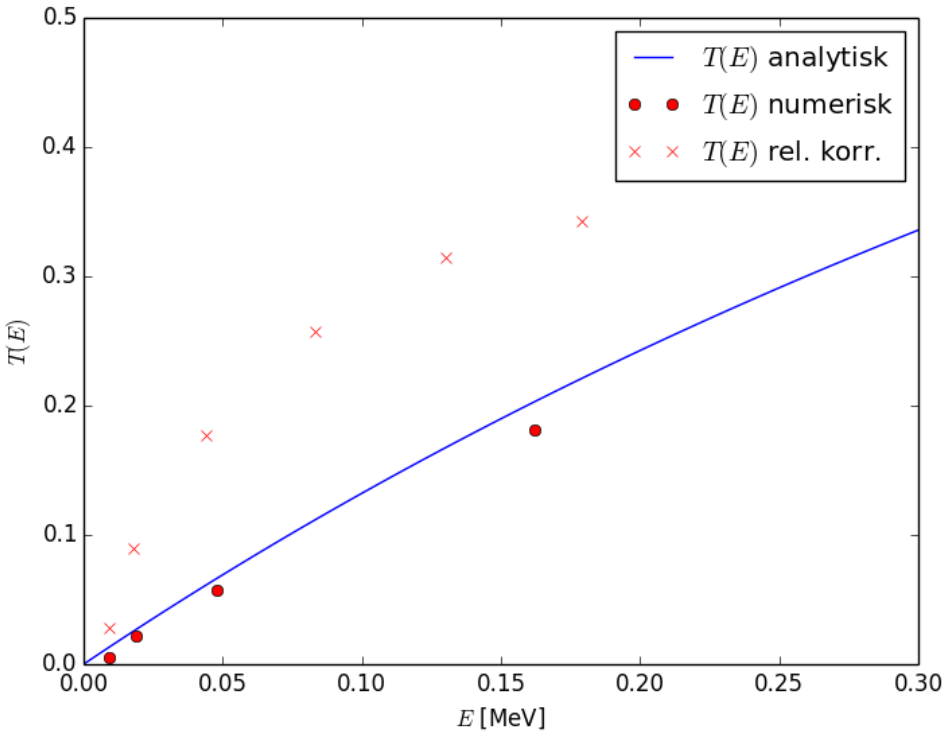

Figure 5: plottet for numeriske og den analytiske løsningen for T(E)

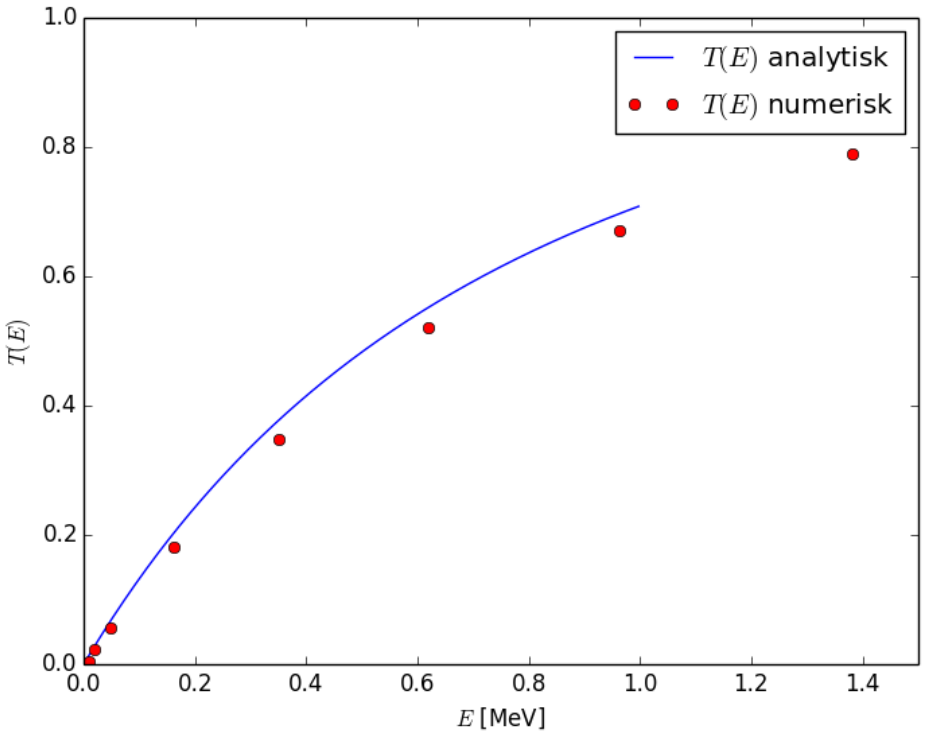

Figure 6: plottet for den numeriske og den analytiske løsningen for T(E)

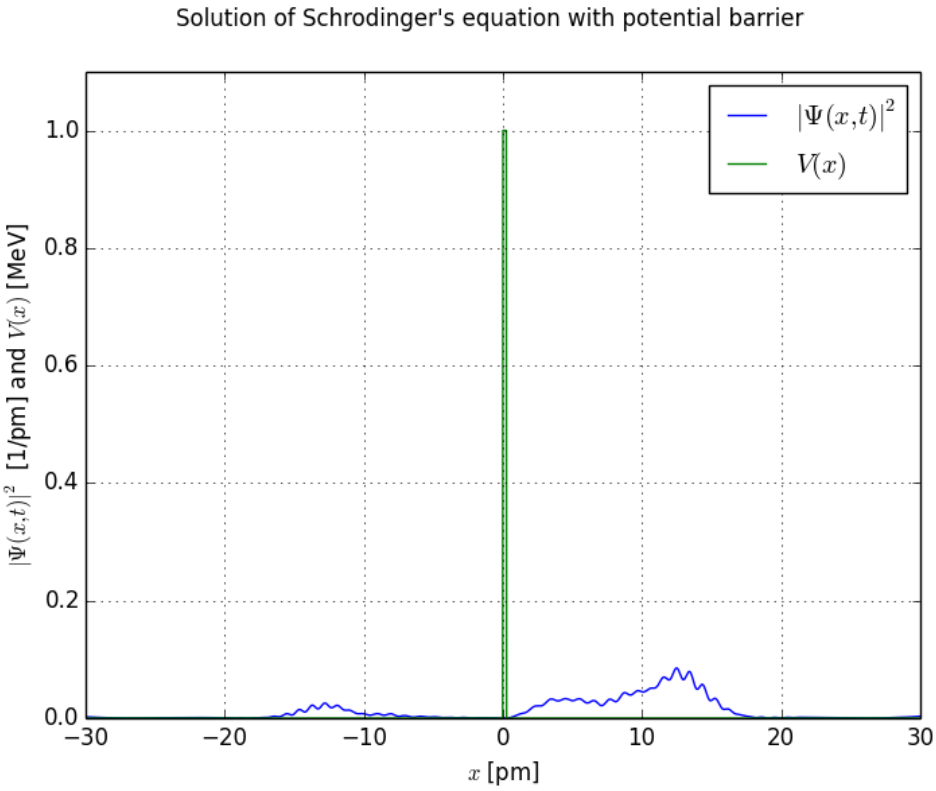

Figure 7: Løsning av likningen med potensialsperre

Code

1skript.py

```
from numpy import *
from matplotlib.pyplot import *

# Function definitions
def Psi0( x ):
    '''
    Initial state for a gaussian wave packet.
    '''
    x1 = 5.00    # [fm] Start here
    a  =  1.00    # [fm] Width of packet

    A = ( 1. / ( 2 * pi * a**2 ) )**0.25
    K1 = exp( - ( x - x1 )**2 / ( 4. * a**2 ) )

    return A * K1

# Main programme
if __name__ == '__main__':

    # Define some numbers for x-axis
    nx = 4001     # Number of points in x direction
    dx = 0.005 # Distance between x points [fm]

    # Interval [a,b], same amount of points each side
    a = - 0.5 * nx * dx
    b = 0.5 * nx * dx
    x = linspace( a, b, nx )   # x is now an array containing
        all x-values used

    # Plot initial state
    figure()         # Create a window for figure
    Psi = Psi0(x)    # Create the initial state as an array
       Psi from the array x
    plot( x, abs(Psi)**2, label='$|\Psi(x,t)|^2$' ) # Plot
    xlabel('$x$ [fm]')                 # Label for x-axis
    ylabel('$|\Psi(x, t)|^2$ [1/fm]') # Label for y-axis
    legend(loc='best')                 # Adds labels of the
       lines to the window
    savefig('psisq_init.eps')          # Save as .eps figure
    axis([-10, 10, 0, 0.7])            # Set axis range


    # Turn off interactive mode
    ioff()

```

```
    # Add show so that windows do not automatically close
    show()
```

2skript.py

```
import scipy.sparse as sparse
import scipy.sparse.linalg
from numpy import *
from matplotlib.pyplot import *

"""Physical constants"""
_E0e =   0.511        # Rest energy for an electron [MeV]
_hbarc = 197.3       # [MeV fm]
_c = 3.0e5            # Spees of light [fm / as]

"""Parameters of initial state"""
a  =  1.00 # [fm]
l  =  1.38 # [1 / fm]

# Define functions
def Psi0( x , a, l ):
    '''
    Initial state for a travelling gaussian wave packet.
    '''
    x0 = 5.00 # [fm]

    A = ( 1. / ( 2 * pi * a**2 ) )**0.25
    K1 = exp( - ( x - x0 )**2 / ( 4. * a**2 ) )
    K2 = exp( 1j * l * x )

    return A * K1 * K2#5

def potentialBarrier( x, height=1, width=0.0025 ):
    """
    Gives the potential for a potential well of depth depth
       and width width.

    @param height Gives the height of the potential well.
       Given as the magnitude
    (positive integer / double / float).
    @param width Gives the width of the potential well. Given
        as positive
    integer definig the fraction of the x spectrum to contain
        the well. For
    example, 1 will mean that the well covers the whole
       spectrum and 0.5 that
    it covers half of it.
    """
    # Declare new empty array with same length as x
```

```
40      potential = zeros( len( x ) )
41
42      potential[ 0.5*len(potential) : (0.5+width)*len(potential
            ) ] = height
43
44      return potential
45
46  if __name__ == '__main__':
47      nx = 5001 # Number of points in x direction
48      dx = 0.020 # Distance between x points [fm]
49
50      # Use zero as center, same amount of points each side
51      x1 = - 0.5 * nx * dx
52      x2 = 0.5 * nx * dx
53      x = linspace( x1, x2, nx )
54
55      # Time parameters
56      T = 0.100 # How long to run simulation [as]
57      dt = 5e-4 # The time step [as]
58      t = 0
59      time_steps = int( T / dt ) # Number of time steps
60
61      # Constants - save time by calculating outside of loop
62      k1 = ( 1j * _hbarc * _c) / (2. * _E0e )
63      k2 = - ( 1j * _c ) / _hbarc
64
65      # Create the initial state Psi
66      Psi = Psi0(x,a,l)
67
68      # Create the matrix containing central differences. It it
            used to
69      # approximate the second derivative.
70      data = ones((3, nx))
71      data[1] = -2*data[1]
72      diags = [-1,0,1]
73      D2 = k1 / dx**2 * sparse.spdiags(data,diags,nx,nx)
74
75      # Identity Matrix
76      I = sparse.identity(nx)
77
78      # Create the diagonal matrix containing the potential.
79      V_data = potentialBarrier(x)
80      V_diags = [0]
81      V = k2 * sparse.spdiags(V_data, V_diags, nx, nx)
82
83      # Put mmatplotlib in interactive mode for animation
84      ion()
85
86      # Setup the figure before starting animation
```

```
87      fig = figure() # Create window
88      ax = fig.add_subplot(111) # Add axes
89      line, = ax.plot( x, abs(Psi)**2, label='$|\Psi(x,t)|^2$'
            ) # Fetch the line object
90
91      # Also draw a green line illustrating the potential
92      ax.plot( x, V_data, label='$V(x)$' )
93
94      # Add other properties to the plot to make it more
            elegant
95      fig.suptitle("Solution of Schrodinger's equation with
            potential barrier") # Title of plot
96      ax.grid('on') # Square grid lines in plot
97      ax.set_xlabel('$x$ [fm]') # X label of axes
98      ax.set_ylabel('$|\Psi(x, t)|^2$ [1/fm] and $V(x)$ [MeV]')
            # Y label of axes
99      ax.set_xlim([-30, 30])  # Sets x-axis range
100     ax.set_ylim([0, 1.1])   # Sets y-axis range
101     ax.legend(loc='best')   # Adds labels of the lines to the
            window
102     draw() # Draws first window
103
104     # Time loop
105     while t < T:
106         """
107         For each iteration: Solve the system of linear
                equations:
108         (I - k/2*D2) u_new = (I + k/2*D2)*u_old
109         """
110         # Set the elements of the equation
111         A = (I - dt/2. * (D2 + V))
112         b = (I + dt/2. * (D2 + V)) * Psi
113
114         # Calculate the new Psi
115         Psi = sparse.linalg.spsolve(A,b)
116
117         # Update time
118         t += dt
119
120         # Plot this new state
121         line.set_ydata( abs(Psi)**2 ) # Update the y values
                of the Psi line
122         draw() # Update the plot
123
124
125     # Integral to find transmission probability for a given
            energy
126     E = _hbarc**2*(l**2 + 1./(4.*a**2))/(2*_E0e)  # Calculate
            energy expectation value [MeV]
```

```
127     Psi_pluss = Psi[(nx-1)/2:]                    # Slice
            away list for x < 0
128     I = trapz(abs(Psi_pluss)**2,None,dx)          # Integrate
            with trapezoidal method
129     print E, I
130
131
132     # Turn off interactive mode
133     ioff()
134
135     # Add show so that windows do not automatically close
136     show()
```

3skript.py

```
1  from numpy import *
2  from matplotlib.pyplot import *
3
4
5  """Physical constants"""
6  _E0e =   0.511        # Rest energy for an electron [MeV]
7  _hbarc = 0.1973       # [MeV pm]
8
9
10 # Function definition
11 def T( E ):
12     '''
13     Expression for transmission probability
14     '''
15     V0 = 34.    # Height of potential [MeV]
16     L = 17.  # Length of potential box [pm]
17
18     T = 1. / (1. + V0**2/(4.*(V0-E)*E)*sinh(sqrt(2.*_E0e*(V0-
           E))/_hbarc*L)**2)
19
20     return T
21
22 # Main programme
23 if __name__ == '__main__':
24
25     # Define some numbers for axis
26     nE = 400     # Number of points
27     dE = 0.0025  # Distance between points [MeV]
28
29     # Interval [a,b]
30     a = 0
31     b = nE * dE
32     E = linspace( a, b, nE )   # An array containing all
           values of energy used
```

```
33
34      # Plotting
35      figure()            # Create a window for figure
36      plot( E, T(E), label='$T(E)$ analytisk' ) # Plot T(E)
            function
37      Evalues =
            [0.009,0.019,0.048,0.162,0.352,0.619,0.962,1.381]
38      Tvalues =
            [0.005,0.022,0.057,0.181,0.348,0.520,0.671,0.789]
39      plot(Evalues, Tvalues, 'ro', label='$T(E)$ numerisk' ) #
            Plot data points
40      xlabel('$E$ [MeV]')                 # Label for x-axis
41      ylabel('$T(E)$')                    # Label for y-axis
42      legend(loc='best')                  # Adds labels of the
            lines to the window
43      savefig('TE1.eps')                  # Save as .eps figure
44      axis([0, 1.5, 0, 1.0])                # Set axis range
45
46      # Plotting
47      figure()            # Create a window for figure
48      plot( E, T(E), label='$T(E)$ analytisk' ) # Plot T(E)
            function
49      plot(Evalues, Tvalues, 'ro', label='$T(E)$ numerisk') #
            Plot data points
50      Evalues = [0.009,0.018,0.044,0.083,0.130,0.179]
51      Tvalues = [0.028,0.090,0.177,0.257,0.315,0.343]
52      plot(Evalues, Tvalues, 'rx',label='$T(E)$ rel. korr.') #
            Plot data points
53      xlabel('$E$ [MeV]')                 # Label for x-axis
54      ylabel('$T(E)$')                    # Label for y-axis
55      legend(loc='best')                  # Adds labels of the
            lines to the window
56      axis([0, 0.3, 0, .5])               # Set axis range
57      savefig('TE2.eps')                  # Save as .eps figure
58
59
60      # Turn off interactive mode
61      ioff()
62
63      # Add show so that windows do not automatically close
64      show()
```

4skript.py

```
1  import scipy.sparse as sparse
2  import scipy.sparse.linalg
3  from numpy import *
4  from matplotlib.pyplot import *
5
```

```
"""Physical constants"""
_E0e =   0.511        # Rest energy for an electron [MeV]
_hbarc = 0.1973       # [MeV pm]
_c = 3.0e2            # Spees of light [pm / as]

"""Parameters of initial state"""
a  =  1.00 # [pm]
l  =  1.38 # [1 / pm]


# Define functions
def Psi0( x , a, l ):
    '''
    Initial state for a travelling gaussian wave packet.
    '''
    x0 = 5.00 # [pm]

    A = ( 1. / ( 2 * pi * a**2 ) )**0.25
    K1 = exp( - ( x - x0 )**2 / ( 4. * a**2 ) )
    K2 = exp( 1j * l * x )

    return A * K1 * K2

def potentialBarrier( x, height=1., width=0.0025 ):
    """
    Gives the potential for a potential well of depth depth
       and width width.

    @param height Gives the height of the potential well.
       Given as the magnitude
    (positive integer / double / float).
    @param width Gives the width of the potential well. Given
        as positive
    integer definig the fraction of the x spectrum to contain
        the well. For
    example, 1 will mean that the well covers the whole
       spectrum and 0.5 that
    it covers half of it.
    """
    # Declare new empty array with same length as x
    potential = zeros( len( x ) )

    potential[ 0.5*len(potential) : (0.5+width)*len(potential
       ) ] = height

    return potential

if __name__ == '__main__':
    nx = 5001 # Number of points in x direction
```

```
    dx = 0.020 # Distance between x points [pm]

    # Use zero as center, same amount of points each side
    x1 = - 0.5 * nx * dx
    x2 = 0.5 * nx * dx
    x = linspace( x1, x2, nx )

    # Time parameters
    T = 0.120 # How long to run simulation [as]
    dt = 5e-4 # The time step [as]
    t = 0
    time_steps = int( T / dt ) # Number of time steps

    # Constants - save time by calculating outside of loop
    k1 = - ( 1j * _hbarc * _c ) / (2. * _E0e )
    k2 = ( 1j * _c ) / _hbarc
    k3 = - ( 1j  * _hbarc**3  * _c ) / (8. * _E0e**3 )

    # Create the initial state Psi
    Psi = Psi0(x,a,l)

    # Create the matrix containing central differences. It it
        used to
    # approximate the second derivative.
    data = ones((3, nx))
    data[1] = -2*data[1]
    diags = [-1,0,1]
    D2 = k1 / dx**2 * sparse.spdiags(data,diags,nx,nx)

    # Create another matrix for the fourth derivative
    data3 = ones((5, nx))
    data3[1] = -4*data3[1]
    data3[2] = 6*data3[2]
    data3[3] = -4*data3[3]
    diags3 = [-2,-1,0,1,2]
    D3 = k3 / dx**4 * sparse.spdiags(data3,diags3,nx,nx)

    # Identity Matrix
    I = sparse.identity(nx)

    # Rest energy term. Can be ignored. Please comment out to
        try.
    #M = k2 * 0.511 * I

    # Create the diagonal matrix containing the potential.
    V_data = potentialBarrier(x)
    V_diags = [0]
    V = k2 * sparse.spdiags(V_data, V_diags, nx, nx)

```

```
    # Put mmatplotlib in interactive mode for animation
    ion()

    # Setup the figure before starting animation
    fig = figure() # Create window
    ax = fig.add_subplot(111) # Add axes
    line, = ax.plot( x, abs(Psi)**2, label='$|\Psi(x,t)|^2$' ) # Fetch the line object

    # Also draw a green line illustrating the potential
    ax.plot( x, V_data, label='$V(x)$' )

    # Add other properties to the plot to make it more elegant
    fig.suptitle("Solution of Schrodinger's equation with potential barrier") # Title of plot
    ax.grid('on') # Square grid lines in plot
    ax.set_xlabel('$x$ [pm]') # X label of axes
    ax.set_ylabel('$|\Psi(x, t)|^2$ [1/pm] and $V(x)$ [MeV]') # Y label of axes
    ax.set_xlim([-30, 30])  # Sets x-axis range
    ax.set_ylim([0, 1.1])   # Sets y-axis range
    ax.legend(loc='best')   # Adds labels of the lines to the window
    draw() # Draws first window

    # Time loop
    while t < T:
        """
        For each iteration: Solve the system of linear equations:
        (I - k/2*D) u_new = (I + k/2*D)*u_old
        """
        # Set the elements of the equation
        #A = (I - dt/2. * (D2 + D3 + M + V))
        #b = (I + dt/2. * (D2 + D3 + M + V)) * Psi
        A = (I - dt/2. * (D2 + D3 + V))
        b = (I + dt/2. * (D2 + D3 + V)) * Psi

        # Calculate the new Psi
        Psi = sparse.linalg.spsolve(A,b)

        # Update time
        t += dt

        # Plot this new state
        line.set_ydata( abs(Psi)**2 ) # Update the y values of the Psi line
        draw() # Update the plot
```

```
138
139
140     # Integral to find transmission probability for a given
            energy
141     E = _hbarc**2*(l**2 + 1./(4.*a**2))/(2*_E0e)
142     E = E - _hbarc**4*(l**4 + 1.5*l**2/a**2 + 3./16./a**4)
            /(8.*_E0e**3) # [MeV]
143     Psi_pluss = Psi[(nx-1)/2:]              # Slice away list
            for x < 0
144     I = trapz(abs(Psi_pluss)**2,None,dx)          # Integrate
            with trapezoidal method
145     print E, I
146
147
148     # Turn off interactive mode
149     ioff()
150
151     # Add show so that windows do not automatically close
152     show()
```