FYS1120 Elektromagnetisme 2014

Oblig 2

Partiklar i elektriske og magnetiske felt.

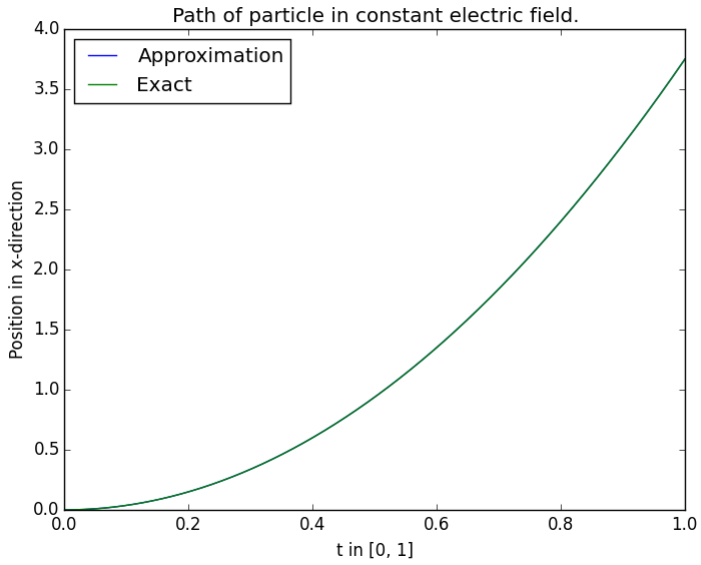
Øyvind Sigmundson Schøyen

21. oktober 2014

Innhold

1 Partikkel i elektrisk felt

Det er ikkje oppført i denne oppgåva om me skal sjå vekk frå gravitasjonskrafta, men eg har ikkje tatt ho med i utrekningane. Dette grunna då me nyttar dimensjonslause variablar som eg ikkje kjenner konstantane til.



Figur 1: I dette plottet ser me plott over posisjonen til partikkelen i x-retning mot tid samt den analytiske løysninga som ligg over. Desse ligg heilt oppå einannan. Me kan sjå at grunna konstant akselerasjon vil me ha ein lineær hastighet. Difor må posisjonen gå som ein kvadratisk funksjon. Dette vil hjelpe oss med å verifisere den analytiske løysninga.

Når me skal utleie den analytiske løysninga vil me nytte Newtons andre lov i lag med uttrykket for krafta i eit **E**-felt. Då får me

$$\mathbf{F} = q\mathbf{E} = m\mathbf{a}(t) \qquad \Rightarrow \qquad \mathbf{a}(t) = \frac{q}{m}\mathbf{E}.$$

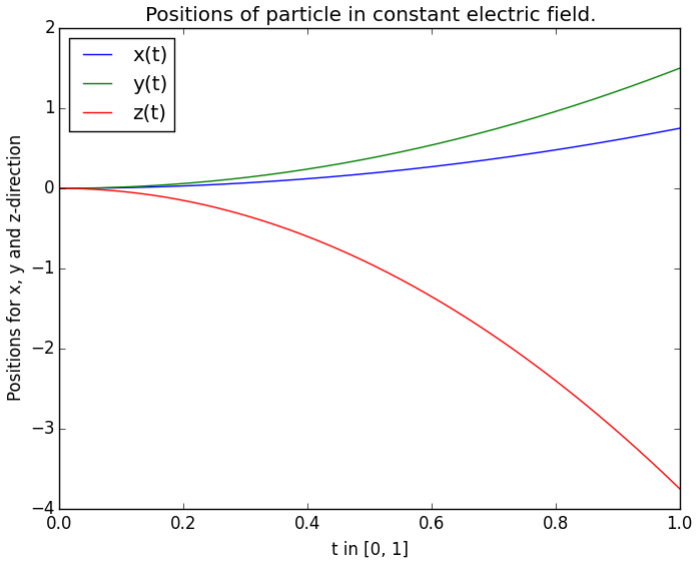
No vil me integrere med hensyn på tid. Me har heile vegen eit bestemt integral kor me byrjar på $t = 0$. Då får me

$$\mathbf{v}(t) = \int_0^t \mathbf{a}(t)\ dt = \int_0^t \frac{q}{m}\mathbf{E}\ dt = \frac{q}{m}\mathbf{E}t.$$

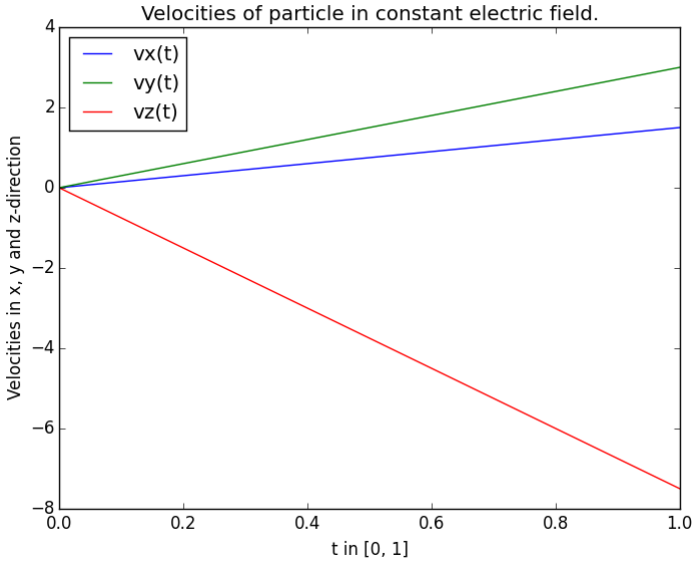
Me integrerer ei gong til for å finne posisjonen.

$$\mathbf{r}(t) = \int_0^t \mathbf{v}(t)\ dt = \int_0^t \frac{q}{m}\mathbf{E}t\ dt = \frac{1}{2}\frac{q}{m}\mathbf{E}t^2.$$

Resultatet av denne ser me i Figur 1.



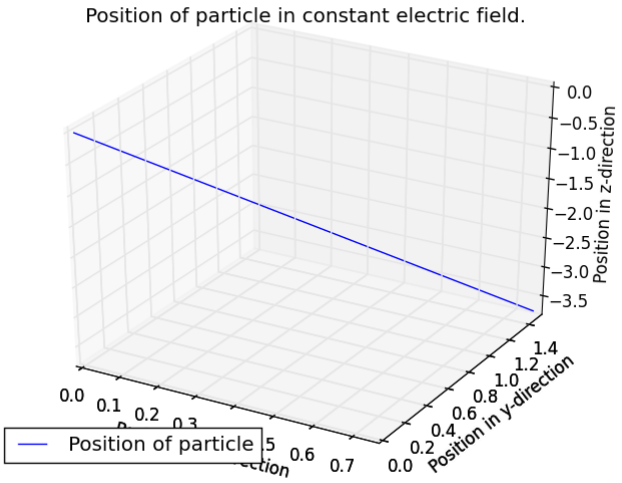
Figur 2: Her ser me plott for posisjonane i x, y og z-retning mot tid. Partikkelen fylgjer retningen til **E**-feltet.



Figur 3: Her ser me plott for hastighetane i x, y og z-retning mot tid. Siden **E**-feltet har komponentar i alle tre retningane har me ein konstant akselerasjon som gjer oss ein lineært aukande hastighet for alle retningane.

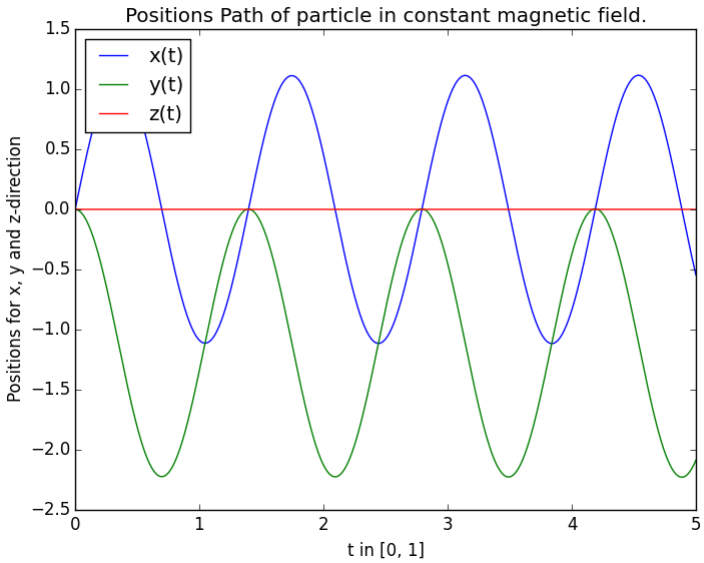
Me kan sjå at dette vil bli ei ballistisk rørsle då partikkelen vil ha ei

konstant aukande fart. Resultatet av dette gjer oss ein posisjon som går som ein andregradsfunksjon. Forflyttninga mellom kvar tideining vil auke og partikkelen vert "skoten ut".

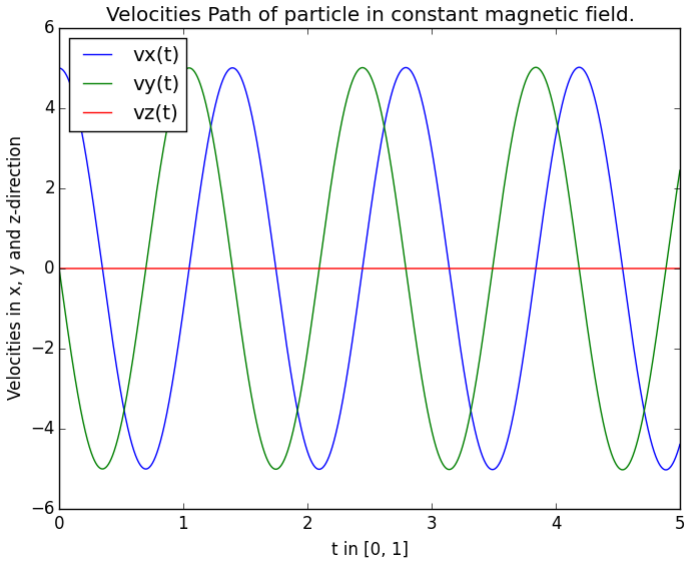


Figur 4: Her vil me sjå korleis partikkelen forflyttar seg i rommet. Me ser derimot ikkje korleis han auker i hastighet. Viss me hadde vist dette som ein film mot tid ville me sett korleis avstanden mellom kvart sekund ville auke.

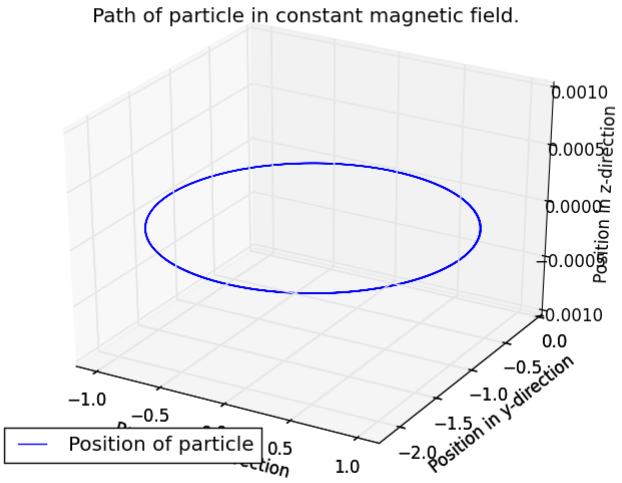
2 Partikkel i magnetisk felt



Figur 5: Her ser me eit plott over posisjonen til partikkelen i x, y og z-retning mot tid. Magnetfeltet gjer oss ei kraft som virker inn mot sentrum av ein sirkel. Me får difor svingningar for x og y-retning då partikkelen beveger seg i ein sirkel i xy-planet. I z-retning har me derimot ingen kraftkomponent og partikkelen endrar ikkje posisjon.



Figur 6: Plottet viser oss hastighetane i x, y og z-retning mot tid. Me får liknande svingningar for hastighetane som for posisjonane. Hastighetane svinger derimot om 0.



Figur 7: Her kjem det tydeleg fram korleis partikkelen beveger seg i ein sirkelbane i magnetfeltet. Dette fordi partikkelen vil ha ein konstant kraftkomponent inn mot sirkelen. Me får då ein sentripetalakselerasjon som vil trekke partikkelen rundt i ein sirkelbane.

Programmet `Oppgave2.py` gjer oss utskrifta

Time spent on one revolution: T = 1.3963

Måten dette vert rekna ut vil bli forklart i programmet til slutt i besvarelsen.

For å finne syklotronfrekvensen nyttar me uttrykket for krafta frå det magnetiske feltet i lag med Newtons andre lov.

$$\mathbf{F}_B = q\left(\mathbf{v} \times \mathbf{B}\right) = m\mathbf{a}.$$

Siden det magnetiske feltet kun virker i z-retning og me er interesserte i den tangensielle hastigheten vil me jobbe med uttrykket

$$F_B = qvB = ma = m\frac{v^2}{r},$$

kor me i siste ledd har brukt at sentripetalakselerasjonen er gjeve ved

$$a = \frac{v^2}{r}.$$

Då vil me få

$$qvB = m\frac{v^2}{r} \qquad \Rightarrow \qquad v = \frac{qrB}{m}.$$

No nyttar me at vinkelhastigheten er gitt ved

$$\omega = \frac{v}{r} \qquad \Rightarrow \qquad v = r\omega.$$

Det gjer oss

$$\omega = \frac{qB}{m}.$$

Omlaupsperioden T er gjeve ved

$$T = \frac{2\pi r}{v} = \frac{2\pi}{\omega}.$$

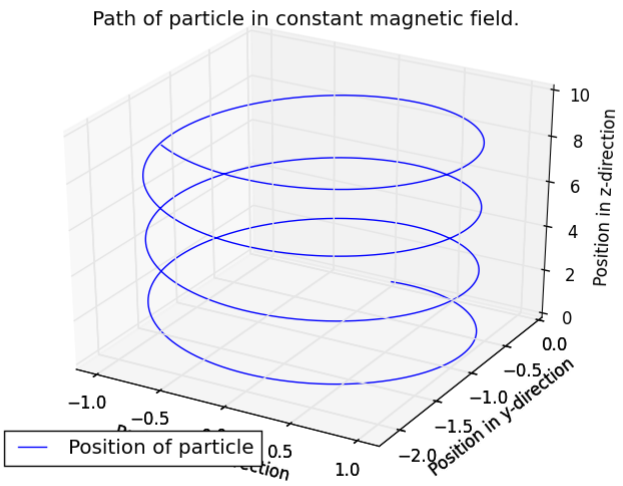
Då fylgjer det at

$$T = \frac{2\pi}{w} = \frac{2\pi m}{qB}.$$

Me vil då få ein omlaupsperiode på

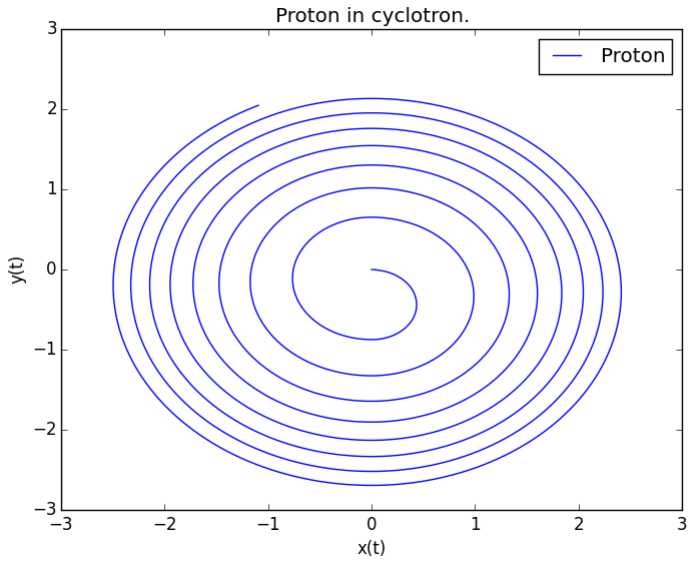
$$T = \frac{2\pi(2)}{(3)(3)} \approx 1.3962.$$

Dette er veldig nærme det me får frå det numeriske resultatet. Eventuelle feil vil kome frå feil i Euler-Cromer og numerisk avrunding.



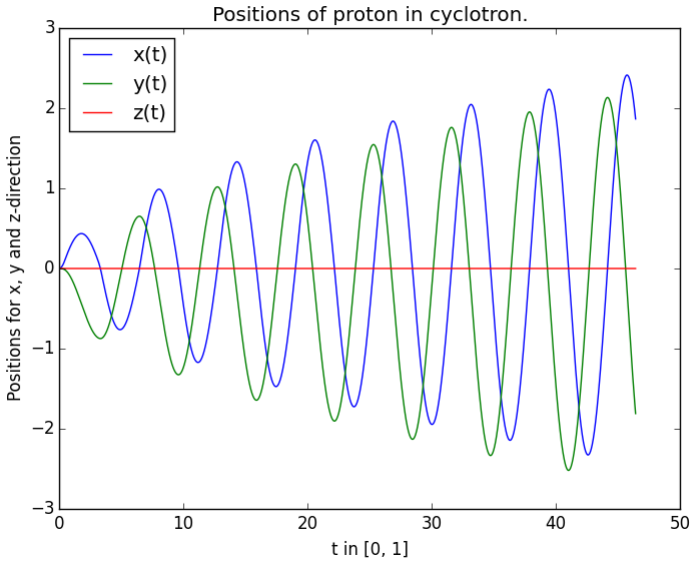
Figur 8: Plottet vil her danne ein spiral som flyttar seg i positiv z-retning då me ikkje har noko tyngdekraft som motvirkar. I tillegg vil all krafta frå det magnetiske feltet virke i xy-planet og vil difor ikkje endre på startverdien til $\mathbf{v}$ i z-retning.

3 Partikkel i syklotron

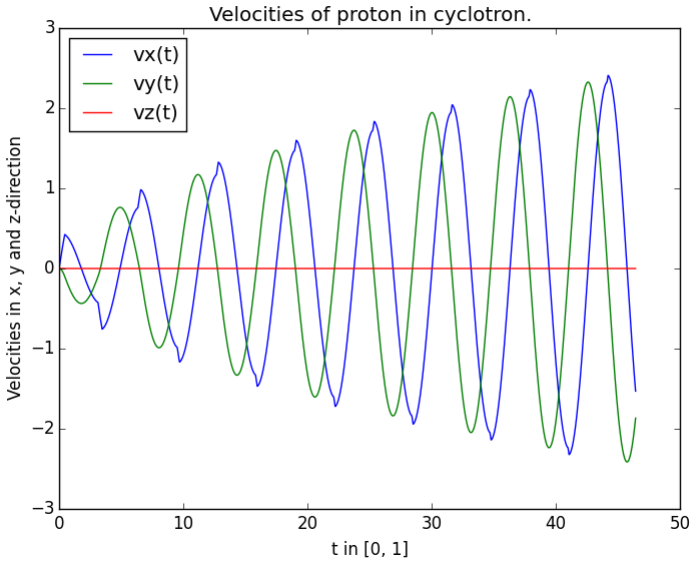


Figur 9: Me ser tydeleg korleis protonet flyttar seg utover i syklotronen etter som tida auker.

Radien auker derimot ikkje jamnt. Dette kjeme frå det faktum at partikkelen nyttar mindre og mindre tid mellom D'ane, kor protonet vert tilført meir energi, samanlikna med tid brukt inne i D'ane.



Figur 10: BLA



Figur 11: BLABLA

Farten partikkelen forlater syklotronen med vert rekna ut i programmet `Oppgave3.py`. Dette vil bli forklart ved programma. Programmet gjer oss utskrifta

Escape velocity of proton: v = 2.41456

Dette seier oss lite då v er dimensjonslaus. Når me no rekner ut energien vil me kunne finne eit uttrykk for hastigheta.

For å finne eit uttrykk for den kinetiske energien nyttar me igjen Newtons andre lov kor me ser sentripetalakselerasjonen og hastigheten i tangensiell retning.

$$v = \frac{qrB}{m}$$

Dette setter me inn i uttrykket for kinetisk energi gjeve ved

$$E_k = \frac{1}{2}mv^2.$$

Me får

$$E_k = \frac{1}{2}m\left(\frac{qrB}{m}\right)^2 = \frac{1}{2}\frac{(qrB)^2}{m}.$$

Me setter inn verdiane for eit proton som me finn i ein tabell, $m_P = 1.673 \times 10^{-27}$ kg, $q = 1.602 \times 10^{-19}$ C.

$$\begin{aligned} E_k &= \frac{1}{2}\frac{\left((1.602 \times 10^{-19}\text{ C})(1\text{ T})(1\text{ m})\right)^2}{1.673 \times 10^{-27}\text{ kg}} \\ &\approx 7.67 \times 10^{-12}\text{ J} \\ &\approx 47.87\text{ MeV}. \end{aligned}$$

Viss me no setter dette lik det klassiske uttrykket for kinetisk energi finner me at protonet har ei hastighet på

$$v \approx 0.32\text{ c},$$

kor c er lysfarta. Dette er ei såpass høg hastighet at me ikkje kan neglisjere den relativistiske effekten på hastigheta.

For å rekne ut antal rundar som trengs vil me finne eit forholdstal mellom E_k og tilført energi U. Dette vil vere gjeve ved

$$E_k = nU,$$

kor n vil vere antal rundar. Siden protonet opplever ei energitilføring to gonger per runde vil me få formelen

$$n = \frac{E_k}{2U}.$$

Då har me at U er gjeve ved

$$U = qV$$

. Eg antar at ein eksperimentelt tilgjengeleg spenning vil vere $V = 50$ kV. Det gjer oss

$$\begin{aligned} U = qV &= (1.602 \times 10^{-19}\ \text{C})(50 \times 10^{3}\ \text{V}) \\ &\approx 8.01 \times 10^{-15}\ \text{J}. \end{aligned}$$

Då finner me antal rundar til å vere

$$\begin{aligned} n &= \frac{7.67 \times 10^{-12}\ \text{J}}{2(8.01 \times 10^{-15}\text{J})} \\ &\approx 479\ \text{Rundar}. \end{aligned}$$

Frekvensen er definert som "ein runde per sekund". Det gjer oss

$$\begin{aligned} f = \frac{1}{T} &= \frac{qB}{2\pi m} \\ &= \frac{(1.602 \times 10^{-19}\ \text{C})(1\ \text{T})}{2\pi(1.673 \times 10^{-27}\ \text{kg})} \\ &\approx 15.2\ \text{MHz}. \end{aligned}$$

4 Programma

4.1 Oppgave1.py

```
from numpy import zeros, linspace, sin, array, amin, amax
from matplotlib.pylab import plot, show, title, legend, xlabel,\
        ylabel, hold, figure, savefig
from mpl_toolkits.mplot3d.axes3d import Axes3D

"
Superclass used for calculation, plotting and storing of necessary
components for a particle in an electric field.
"
class ParticleInElectricField:

    "
    Storing variables and creating necessary arrays.
    "
    def __init__(self, E, m, q, r0, v0, t_start, t_final, dt):

        self.E = E # Electric field.
        self.m = m # Mass.
        self.q = q # Charge.
        self.r0= r0 # Initial position.
        self.v0 = v0 # Initial velocity.
        self.t_start = t_start # Start value for time.
        self.t_final = t_final # End value for time.
        self.dt = dt # Timestep.

        # Creating arrays and timestep.
        self.n = int(self.t_final/float(self.dt)) # Number of iterations.
        self.t = linspace(self.t_start, self.t_final, self.n+1) # Time.
        self.r = zeros(shape=(self.n+1, 3)) # Array containing positions.
        self.r[0, :] = self.r0
        self.v = zeros(shape=(self.n+1, 3)) # Array containing velocities.
        self.v[0, :] = self.v0
        self.EXACT = zeros(shape=(self.n+1, 3)) # Array containing positions.

    "
    Calculating path of particle in an electric field.
    "
    def calculatePath(self):

        # Implementing Euler-Cromer.
```

```
            for i in range(self.n):
                F = self.q*self.E
                a = F/float(self.m)
                self.v[i+1, :] = self.v[i, :] + a*self.dt
                self.r[i+1, :] = self.r[i, :] + self.v[i+1, :]*self.dt
                # Implementing exact formula for path.
                # This formula is found from analytical integration.
                self.EXACT[i+1, :] = 0.5*self.q/float(self.m)*self.E*self.t[i]**2

        "
        Versatile method for plotting different values and components.
        "
        def plotPath(self, TITLE, optn):

            if optn == 0:

                "
                Plotting the path in x-direction as a function of time.
                Showing the exact and the numerical representation in the same plot.
                "
                plot(self.t, self.r[:, 0])
                hold('on')
                plot(self.t, self.EXACT[:, 0])
                hold('off')
                title(TITLE)
                legend(('Approximation', 'Exact'), loc=2)
                xlabel(t in [0, 1]")
                ylabel(Position in x-direction")
#               savefig('A.png')
                show()

            elif optn == 1:

                "
                Plotting the path and the velocity of the particle.
                Showing all directions as a function of time.
                "
                figure()
                plot(self.t, self.r[:, 0])
                hold('on')
                plot(self.t, self.r[:, 1])
                plot(self.t, self.r[:, 2])
                hold('off')
                title(Positions " + TITLE)
```

```
            legend(('x(t)', 'y(t)', 'z(t)'), loc=2)
            xlabel(t in [0, 1]")
            ylabel(Positions for x, y and z-direction")
#            savefig('3A1.png')
            show()

            figure()
            plot(self.t, self.v[:, 0])
            hold('on')
            plot(self.t, self.v[:, 1])
            plot(self.t, self.v[:, 2])
            hold('off')
            title("Velocities " + TITLE)
            legend(('vx(t)', 'vy(t)', 'vz(t)'), loc=2)
            xlabel('t in [0, 1]')
            ylabel('Velocities in x, y and z-direction')
#            savefig('3A2.png')
            show()

        else:

            "
            Plotting the path of the particle in 3D.
            "
            # Something is funky here...
            fig = figure()
            ax = fig.add_subplot(1, 1, 1, projection='3d')
            ax.plot3D(self.r[:, 0], self.r[:, 1], self.r[:, 2],\
                      label='Position of particle')
            ax.legend(loc = 'lower left')
            ax.set_xlabel('Position in x-direction')
            ax.set_ylabel('Position in y-direction')
            ax.set_zlabel('Position in z-direction')
            ax.set_title(TITLE)
            ax.set_xlim(amin(self.r[:, 0]), amax(self.r[:, 0]))
            ax.set_ylim(amin(self.r[:, 1]), amax(self.r[:, 1]))
            ax.set_zlim(amin(self.r[:, 2]), amax(self.r[:, 2]))
#            savefig('2A4.png')
            show()


if __name__ == '__main__':

    # Assignment 1a) and 1b)
```

```
    E = array([5, 0, 0])
    r0 = array([0, 0, 0])
    v0 = array([0, 0, 0])
    dt = 1.0e-4
    t0 = 0
    tend = 1
    m = 2
    q = 3
    particle = ParticleInElectricField(E, m, q, r0, v0, t0, tend, dt)
    particle.calculatePath()
    particle.plotPath(Path of particle in constant electric field.", 0)

    # Assignment 1c) and 1d)
    E = array([1, 2, -5])
    particle = ParticleInElectricField(E, m, q, r0, v0, t0, tend, dt)
    particle.calculatePath()
    particle.plotPath("of particle in constant electric field.", 1)
    particle.plotPath(Position of particle in constant electric field.", 2)
```

4.2 Oppgave2.py

```
from Oppgave1 import ParticleInElectricField
from numpy import linalg, cross, array

"
Subclass of ParticleInElectricField.
The class makes use of some of the methods made in ParticleInElectricField.
"
class ParticleInMagneticField(ParticleInElectricField):

    "
    Constructor storing most of the values in superclass.
    "
    def __init__(self, E, B, m, q, r0, v0, t_start, t_final, dt):

        ParticleInElectricField.__init__(self,\
                E, m, q, r0, v0, t_start, t_final, dt)
        self.B = B # Magnetic Field.

    "
    Method overriding ParticleInElectricFiled.calculatePath().
    "
    def calculatePath(self):
```

```
            for i in range(self.n):
                F = self.q*cross(self.v[i, :], self.B)
                a = F/float(self.m)
                self.v[i+1, :] = self.v[i, :] + a*dt
                self.r[i+1, :] = self.r[i, :] + self.v[i+1, :]*dt

        "
        Method calculating time used for a particle to make one complete revolution.
        "
        def calculateRevolutionTime(self):

            eps = 1.0e-3
            for i in range(1, self.n):
                # From the first graph we see that x(t) oscillates between positive
                # and negative values. We want to find the point where the function
                # changes from negative to positive values.
                if self.r[i, 0] <= 0 <= self.r[i+1, 0]:
                    T = self.t[i+1]
                    break

            print (Time spent on one revolution: T = %g" % T)



    if __name__ == '__main__':

        # Assignment 2a) and 2b)
        B = array([0, 0, 3])
        m = 2
        q = 3
        r0 = array([0, 0, 0])
        v0 = array([5, 0, 0])
        t0 = 0
        tend = 5
        dt = 1.0e-4
        particle = ParticleInMagneticField(0, B, m, q, r0, v0, t0, tend, dt)
        particle.calculatePath()
        particle.calculateRevolutionTime()
        particle.plotPath(Path of particle in constant magnetic field.", 1)
        particle.plotPath(Path of particle in constant magnetic field.", 2)

        # Assignment 2d)
        v0 = array([5, 0, 2])
```

```
    particle = ParticleInMagneticField(0, B, m, q, r0, v0, t0, tend, dt)
    particle.calculatePath()
    particle.plotPath(Path of particle in constant magnetic field.", 2)
```

4.3 Oppgave3.py

```
from Oppgave2 import ParticleInMagneticField
from numpy import array, cos, linalg, cross, zeros, linspace
from matplotlib.pylab import plot, show, title, legend, xlabel, ylabel, savefig

"
Subclass of subclass ParticleInMagneticField.
"
class Cyclotron(ParticleInMagneticField):

    "
    Constructor storing most values in parentclass which in turn stores these
    in the superclass.
    "
    def __init__(self, omega, E, B, m, q, r0, v0, t_start, t_final, dt):
        ParticleInMagneticField.__init__(self,\
                E, B, m, q, r0, v0, t_start, t_final, dt)
        self.omega = omega # Angular velocity.

    "
    Method overriding ParticleInMagneticField.calculatePath().
    "
    def calculatePath(self):

        for i in range(self.n):
            # Conditions for calculating the force with the electric field.
            # If this condition is fulfilled the particle is between the dee's.
            if -0.1 <= self.r[i, 0] <= 0.1:
                F = q*(cross(self.v[i, :], self.B)\
                        + array([cos(self.omega*self.t[i]), 0, 0]))
            else:
                F = q*cross(self.v[i, :], self.B)

            a = F/float(self.m)
            self.v[i+1, :] = self.v[i, :] + a*self.dt
            self.r[i+1, :] = self.r[i, :] + self.v[i+1, :]*self.dt

    "
    Method calculating the escape velocity of the particle when it
```

```
    reaches the exit".
    "
    def escapeVelocity(self, rD):

        for i in range(self.n+1):
            if rD - linalg.norm(self.r[i, :]) <= 0:
                self.vesc = linalg.norm(self.v[i, :])
                finalTime = i
                # Avoiding arrays of length n+1.
                break

        tempr = zeros(shape=(finalTime, 3))
        tempv = zeros(shape=(finalTime, 3))
        tempt = zeros(finalTime)

        for i in range(finalTime):
            tempr[i, :] = self.r[i, :]
            tempv[i, :] = self.v[i, :]
            tempt[i] = self.t[i]

        self.r = tempr
        self.v = tempv
        self.t = tempt
        print (Escape velocity of proton: v = %g" % self.vesc)

    "
    Method used for plotting of a particle in a cyclotron.
    "
    def plotCPath(self, TITLE):

        plot(self.r[:, 0], self.r[:, 1])
        title(TITLE)
        xlabel("x(t)")
        ylabel("y(t)")
        legend((Proton",), loc=1)
#        savefig('3A.png')
        show()



if __name__ == '__main__':

    rD = 2.6
```

```
m = 1
q = 1
v = array([0, 0, 0])
r = array([0, 0, 0])
tstart = 0
tend = 50
dt = 1.0e-3
B = array([0, 0, 1])
omega = q*linalg.norm(B)/float(m)
E = array([0, 0, 0])

cyclotron = Cyclotron(omega, E, B, m, q, r, v, tstart, tend, dt)
cyclotron.calculatePath()
cyclotron.plotCPath(Proton in cyclotron.")
cyclotron.escapeVelocity(rD)
cyclotron.plotPath("of proton in cyclotron.", 1)
```