Øving 12

Simen Hustad

November 10, 2021

Jeg løste oppgaven ved å lage et objekt som har visse metoder og attributter for å enklere gjennomføre ønskede operasjoner. Flere av kodesnuttene under gir da ikke fullt mening på egenhånd, men det skal være nok til å forstå hva som skjer. Fullstendig kode ligger som vedlegg på slutten i form av tekst og bilde.

Oppgave 1

a)

Tok ikke med

b)

Fant vektorfeltet $\vec{v}$ ved å sette inn $r = (x, y)$.

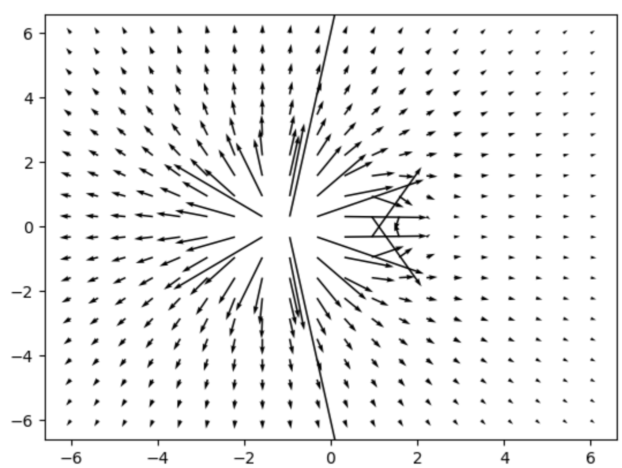

Figure 1: Vektorfelt $\vec{v}$

```
def f(self, pos, t):
    x, y = pos #Posisjon. Kan være lister med verdier
    r = np.array([x, y])  #r = (x, y)
    r1 = [r[0] + 1, r[1]] #r1 = (x + 1, y)
    r2 = [r[0] - 1, r[1]] #r2 = (x - 1, y)
    r12 = [r1, r2] #Samleliste for r1 og r2
    rLen = [self.absV(k) for k in r12]  # |r| regnes i absV
    rHatt = [r12[i]/rLen[i] for i in range(len(r12))]  # rhatt = r/|r|
    F = (self.L/rLen[0])*rHatt[0] - (self.M/rLen[1])*rHatt[1]  # Setter inn i funksjonsuttrykket
    u = F[0]  # Første koordinat til vektorfeltet
    v = F[1]  # Andre koordinat til vektorfeltet
    return u, v
```

Figure 2: Kode for å finne vektorfeltet

c)

Valgte 3 tilfeldige startposisjoner i nærheten av utløpet og brukte odeint for å danne kurvene.

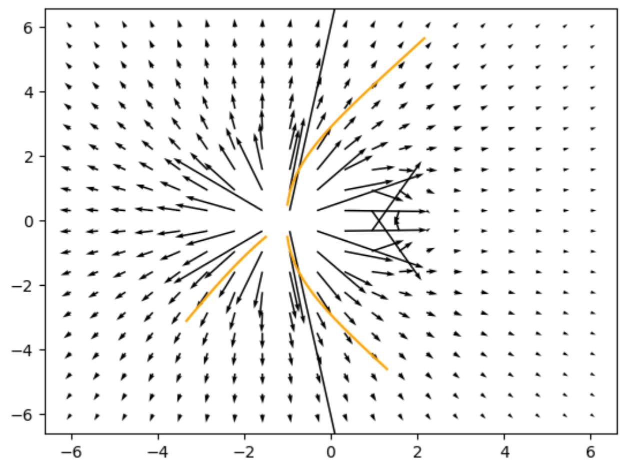

Figure 3: 3 integralkurver

```
def addIntCurve(self, startPos, interval):
    t = np.linspace(interval[0], interval[1], 100) #Setter intervalet til integralet
    sol = odeint(self.f, startPos, t) #Finner punktene langs integralkurven
    self.ax.plot(sol[:, 0], sol[:, 1], color="orange") #Legger resultatet til figuren
```

Figure 4: Kode for integralkurver

d)

```
L*log(x**2 + 2*x + y**2 + 1)/2 - M*log(x**2 - 2*x + y**2 + 1)/2
PS C:\Users\simen\Desktop\Prog\Python> 
```

Figure 5: Svar på potensialfelt i python

```python
def potField(self):
    x, y = sp.symbols("x y") #Definerer symbolene x og y
    L, M = sp.symbols("L M") #Definerer symbolene L og M
    R = CoordSys3D("R") #Danner et 3D koordinatsystem
    Rx, Ry = R.x, R.y #Definerer x og y aksen til systemet
    r = np.array([Rx, Ry])  # r = (x, y)
    r1 = [r[0] + 1, r[1]]  # r1 = (x + 1, y)
    r2 = [r[0] - 1, r[1]]  # r2 = (x - 1, y)
    r12 = [r1, r2] #Samleliste for r1 og r2
    rLen = [self.absV(k) for k in r12]  # |r| regnes i absV funksjonen
    r1Hatt = [r1[i]/rLen[0] for i in range(len(r))] #r1Hatt = r1/|r1|
    r2Hatt = [r2[i]/rLen[1] for i in range(len(r))] #r2Hatt = r2/|r2|
    F = [(L/rLen[0])*r1Hatt[i] - (M/rLen[1])*r2Hatt[i] for i in range(len(r))] #Setter inn i vektorf
    u = F[0]  # Første koordinat til vektorfeltet
    v = F[1]  # Andre koordinat til vektorfeltet
    V = u*R.i + v*R.j #Definerer vektorfeltet i 3D rommet
    potential = scalar_potential(V, R).subs({R.x: x, R.y: y}) #Finner potensialet til vektorfeltet
    return sp.simplify(potential) #Returnerer en forenklet versjon av uttrykket
```

Figure 6: Kode for å finne potensial

e)

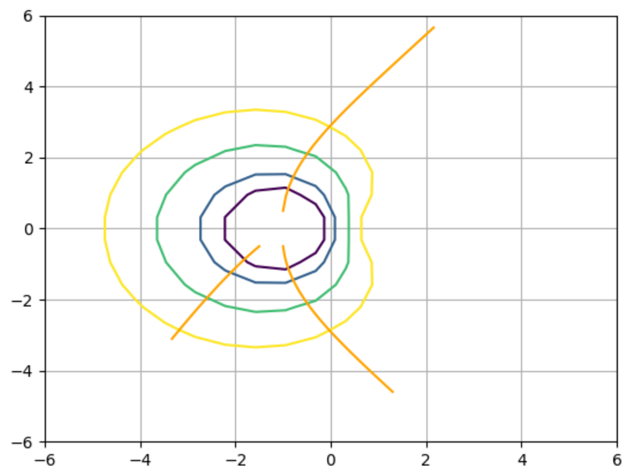

Figure 7: Nivåkurver med integralkurver

```python
def levelCurve(self, level):
    xpoints = np.linspace(self.xlims[0], self.xlims[1], self.meshPoints)  # Danner x-aksen
    ypoints = np.linspace(self.ylims[0], self.ylims[1], self.meshPoints)  # Danner y-aksen
    xp, yp = np.meshgrid(xpoints, ypoints) # Danner et grid med koordinater
    u, v = self.f([xp, yp], 0) #Henter parametriseringen av vektorfeltet
    phi = self.L*np.log2((xp + 1)**2 + yp**2)/2 - self.M*np.log2((xp-1)**2 + yp**2)/2 #Potensialet
    zp = phi #Lager meshgrid som legger til z-verdier
    self.ax.contour(xp, yp, zp, level) #Legger nivåkurven til i figuren
```

Figure 8: Kode for nivåkurver

f)

Gitt en inkompressibel væske vil fluksen ut og inn i en lukket kurve være lik null (hvis ikke er ikke væsken inkompressibel).

Ettersom vektorfeltet er konservativt $\vec{v} = \nabla\phi$ vet vi at sirkulasjonen $curl(\vec{v}) = 0$.

Vedlegg

```python
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
from scipy.integrate import odeint
from sympy.vector import CoordSys3D, scalar_potential


class F:
    'Vektorfelt F'

    def __init__(self, L, M, xlim = [-6, 6], ylim = [-6, 6], points = 20):
        self.L = L #Lagrer L-verdien
        self.M = M #Lagrer M-verdien
        self.xlims = xlim #Lagrer initielle grenser for x
        self.ylims = ylim #Lagrer initielle grenser for y
        self.meshPoints = points #Lagrer initielt antall vektorpunkter
        self.fig, self.ax = plt.subplots()  # Oppretter en ny figur
        self.updateField() #Danner en figur

    def xLimits(self, lims):
        self.xlims = lims #Oppdaterer grensene til x
        self.updateField() #Oppdaterer figuren

    def yLimits(self, lims):
        self.ylims = lims #Oppdaterer grensene til y
        self.updateField() #Oppdaterer figuren
```

Figure 9: Kodesnutt

```python

    def points(self, amount):
        self.meshPoints = amount #Oppdaterer antall vektorpunkter
        self.updateField() #Oppdaterer figuren

    def updateField(self, vecField = False):
        xpoints = np.linspace(self.xlims[0], self.xlims[1], self.meshPoints) #Danner x-aksen
        ypoints = np.linspace(self.ylims[0], self.ylims[1], self.meshPoints) #Danner y-aksen
        xp, yp = np.meshgrid(xpoints, ypoints) #Danner et grid med koordinater
        u, v = self.f([xp, yp], 0) #Finner det parametriserte vektorfeltet
        if vecField: self.ax.quiver(xp, yp, u, v) #Lager vektorpiler i ønsker grid
        self.ax.grid() #Setter på akselinjer i figuren

    def addIntCurve(self, startPos, interval):
        t = np.linspace(interval[0], interval[1], 100) #Setter intervalet til integralet
        sol = odeint(self.f, startPos, t) #Finner punktene langs integralkurven
        self.ax.plot(sol[:, 0], sol[:, 1], color="orange") #Legger resultatet til figuren

    def plot(self):
        plt.show() #Viser figuren
```

Figure 10: Kodesnutt

```python

    def f(self, pos, t):
        x, y = pos #Posisjon. Kan være lister med verdier
        r = np.array([x, y])  #r = (x, y)
        r1 = [r[0] + 1, r[1]] #r1 = (x + 1, y)
        r2 = [r[0] - 1, r[1]] #r2 = (x - 1, y)
        r12 = [r1, r2] #Samleliste for r1 og r2
        rLen = [self.absV(k) for k in r12]  # |r| regnes i absV
        rHatt = [r12[i]/rLen[i] for i in range(len(r12))]  # rhatt = r/|r|
        F = (self.L/rLen[0])*rHatt[0] - (self.M/rLen[1])*rHatt[1]  # Setter inn i funksjonsuttrykket
        u = F[0]  # Første koordinat til vektorfeltet
        v = F[1]  # Andre koordinat til vektorfeltet
        return u, v
```

Figure 11: Kodesnutt

```python
    def potField(self):
        x, y = sp.symbols("x y") #Definerer symbolene x og y
        L, M = sp.symbols("L M") #Definerer symbolene L og M
        R = CoordSys3D("R") #Danner et 3D koordinatsystem
        Rx, Ry = R.x, R.y #Definerer x og y aksen til systemet
        r = np.array([Rx, Ry])  # r = (x, y)
        r1 = [r[0] + 1, r[1]]  # r1 = (x + 1, y)
        r2 = [r[0] - 1, r[1]]  # r2 = (x - 1, y)
        r12 = [r1, r2] #Samleliste for r1 og r2
        rLen = [self.absV(k) for k in r12]  # |r| regnes i absV funksjonen
        r1Hatt = [r1[i]/rLen[0] for i in range(len(r))] #r1Hatt = r1/|r1|
        r2Hatt = [r2[i]/rLen[1] for i in range(len(r))] #r2Hatt = r2/|r2|
        F = [(L/rLen[0])*r1Hatt[i] - (M/rLen[1])*r2Hatt[i] for i in range(len(r))] #Setter inn i vektorf
        u = F[0]  # Første koordinat til vektorfeltet
        v = F[1]  # Andre koordinat til vektorfeltet
        V = u*R.i + v*R.j #Definerer vektorfeltet i 3D rommet
        potential = scalar_potential(V, R).subs({R.x: x, R.y: y}) #Finner potensialet til vektorfeltet
        return sp.simplify(potential) #Returnerer en forenklet versjon av uttrykket


    def levelCurve(self, level):
        xpoints = np.linspace(self.xlims[0], self.xlims[1], self.meshPoints)  # Danner x-aksen
        ypoints = np.linspace(self.ylims[0], self.ylims[1], self.meshPoints)  # Danner y-aksen
        xp, yp = np.meshgrid(xpoints, ypoints) # Danner et grid med koordinater
        u, v = self.f([xp, yp], 0) #Henter parametriseringen av vektorfeltet
        phi = self.L*np.log2((xp + 1)**2 + yp**2)/2 - self.M*np.log2((xp-1)**2 + yp**2)/2 #Potensialet
        zp = phi #Lager meshgrid som legger til z-verdier
        self.ax.contour(xp, yp, zp, level) #Legger nivåkurven til i figuren
```

Figure 12: Kodesnutt

```python
    def absV(self, v):
        #Har en try-except for å veksle mellom numpy og sympy kvadratrot
        try:
            return np.sqrt(sum([k**2 for k in v]))
        except:
            return sp.sqrt(sum([k**2 for k in v]))



v = F(3, 1) #Initierer vektorfeltet med L = 3 og M = 1
```

Figure 13: Kodesnutt

```python
def b():
    global v
    v.updateField(vecField=True)
    v.plot() #Plotter vektorfeltet

def c(var = True):
    global v
    #Integralkurver [start posisjon, interval]
    kurve1 = [[-1, 0.5], [0, 10]]
    kurve2 = [[-1, -0.5], [0, 6]]
    kurve3 = [[-1.5, -0.5], [0, 3]]
    #Legger kurvene til i figuren
    v.addIntCurve(kurve1[0], kurve1[1])
    v.addIntCurve(kurve2[0], kurve2[1])
    v.addIntCurve(kurve3[0], kurve3[1])
    if var: v.plot()

def d():
    global v
    potential = v.potField() #Beregnet potensial
    print(potential)

def e():
    global v
    c(var = False) #Stiller figuren til å ha integralkurvene
    levels = [-0.7, 0.5, 2.0, 3.2] #Nivåkurvene som skal plottes
    v.levelCurve(levels) #Legger nivåkurvene til i figuren
    v.plot() #Viser figuren
```

Figure 14: Kodesnutt

Kode

```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
from scipy.integrate import odeint
from sympy.vector import CoordSys3D, scalar_potential


class F:
    'Vektorfelt F'

    def __init__(self, L, M, xlim = [-6, 6], ylim = [-6, 6], points = 20):
        self.L = L #Lagrer L-verdien
        self.M = M #Lagrer M-verdien
        self.xlims = xlim #Lagrer initielle grenser for x
        self.ylims = ylim #Lagrer initielle grenser for y
        self.meshPoints = points #Lagrer initielt antall vektorpunkter
        self.fig, self.ax = plt.subplots()  # Oppretter en ny figur
        self.updateField() #Danner en figur

    def xLimits(self, lims):
        self.xlims = lims #Oppdaterer grensene til x
        self.updateField() #Oppdaterer figuren

    def yLimits(self, lims):
        self.ylims = lims #Oppdaterer grensene til y
        self.updateField() #Oppdaterer figuren

    def points(self, amount):
        self.meshPoints = amount #Oppdaterer antall vektorpunkter
        self.updateField() #Oppdaterer figuren

    def updateField(self, vecField = False):
        #Danner x-aksen
        xpoints = np.linspace(self.xlims[0], self.xlims[1], self.meshPoints)
        #Danner y-aksen
        ypoints = np.linspace(self.ylims[0], self.ylims[1], self.meshPoints)
        xp, yp = np.meshgrid(xpoints, ypoints) #Danner et grid med koordinater
        u, v = self.f([xp, yp], 0) #Finner det parametriserte vektorfeltet
        #Lager vektorpiler i  nsket  grid
        if vecField: self.ax.quiver(xp, yp, u, v)
        self.ax.grid() #Setter p  akselinjer i figuren

    def addIntCurve(self, startPos, interval):
        #Setter intervalet til integralet
```

```
        t = np.linspace(interval[0], interval[1], 100)
        #Finner punktene langs integralkurven
        sol = odeint(self.f, startPos, t)
        #Legger resultatet til figuren
        self.ax.plot(sol[:, 0], sol[:, 1], color="orange")

    def plot(self):
        plt.show() #Viser figuren

    def f(self, pos, t):
        x, y = pos #Posisjon. Kan v re lister med verdier
        r = np.array([x, y])  #r = (x, y)
        r1 = [r[0] + 1, r[1]] #r1 = (x + 1, y)
        r2 = [r[0] - 1, r[1]] #r2 = (x - 1, y)
        r12 = [r1, r2] #Samleliste for r1 og r2
        rLen = [self.absV(k) for k in r12]  # |r| regnes i absV
        rHatt = [r12[i]/rLen[i] for i in range(len(r12))]
# rhatt = r/|r|
        F = (self.L/rLen[0])*rHatt[0] - (self.M/rLen[1])*rHatt[1]
# Setter inn i funksjonsuttrykket
        u = F[0]  # F rste koordinat til vektorfeltet
        v = F[1]  # Andre koordinat til vektorfeltet
        return u, v

    def potField(self):
        x, y = sp.symbols("x y") #Definerer symbolene x og y
        L, M = sp.symbols("L M") #Definerer symbolene L og M
        R = CoordSys3D("R") #Danner et 3D koordinatsystem
        Rx, Ry = R.x, R.y #Definerer x og y aksen til systemet
        r = np.array([Rx, Ry])  # r = (x, y)
        r1 = [r[0] - 1, r[1]]  # r1 = (x - 1, y)
        r2 = [r[0] + 1, r[1]]  # r2 = (x + 1, y)
        r12 = [r1, r2] #Samleliste for r1 og r2
        rLen = [self.absV(k) for k in r12]  # |r| regnes i absV funksjonen
        r1Hatt = [r1[i]/rLen[0] for i in range(len(r))] #r1Hatt = r1/|r1|
        r2Hatt = [r2[i]/rLen[1] for i in range(len(r))] #r2Hatt = r2/|r2|
        #Setter inn i vektorfeltuttrykket
        F = [(L/rLen[0])*r1Hatt[i] - (M/rLen[1])*r2Hatt[i] for i in range(len(r)
        u = F[0]  # F rste koordinat til vektorfeltet
        v = F[1]  # Andre koordinat til vektorfeltet
        V = u*R.i + v*R.j #Definerer vektorfeltet i 3D rommet
        #Finner potensialet til vektorfeltet
        potential = scalar_potential(V, R).subs({R.x: x, R.y: y})
        #Returnerer en forenklet versjon av uttrykket
        return sp.simplify(potential)
```

```
    def levelCurve(self, level):
        # Danner x-aksen
        xpoints = np.linspace(self.xlims[0], self.xlims[1], self.meshPoints)
        # Danner y-aksen
        ypoints = np.linspace(self.ylims[0], self.ylims[1], self.meshPoints)
        xp, yp = np.meshgrid(xpoints, ypoints) # Danner et grid med koordinater
        u, v = self.f([xp, yp], 0) #Henter parametriseringen av vektorfeltet
        #Potensialfunksjonen
        phi = self.L*np.log2((xp + 1)**2 + yp**2)/2 - self.M*np.log2((xp-1)**2 +
        zp = phi #Lager meshgrid som legger til z-verdier
        self.ax.contour(xp, yp, zp, level) #Legger niv kurven til i figuren


    def absV(self, v):
        #Har en try-except for   veksle mellom numpy og sympy kvadratrot
        try:
            return np.sqrt(sum([k**2 for k in v]))
        except:
            return sp.sqrt(sum([k**2 for k in v]))


v = F(3, 1) #Initierer vektorfeltet med L = 3 og M = 1

#Funksjoner som kalles for   vise svarene til respektive oppgaver

def b():
    global v
    v.updateField(vecField=True)
    v.plot() #Plotter vektorfeltet

def c(var = True):
    global v
    #Integralkurver [start posisjon, interval]
    kurve1 = [[-1, 0.5], [0, 10]]
    kurve2 = [[-1, -0.5], [0, 6]]
    kurve3 = [[-1.5, -0.5], [0, 3]]
    #Legger kurvene til i figuren
    v.addIntCurve(kurve1[0], kurve1[1])
    v.addIntCurve(kurve2[0], kurve2[1])
    v.addIntCurve(kurve3[0], kurve3[1])
    if var: v.plot()

def d():
    global v
    potential = v.potField() #Beregnet potensial
```

```
    print(potential)

def e():
    global v
    c(var = False) #Stiller figuren til  ha integralkurvene
    levels = [-0.7, 0.5, 2.0, 3.2] #Niv kurvene som skal plottes
    v.levelCurve(levels) #Legger niv kurvene til i figuren
    v.plot() #Viser figuren
```