Breakdown kode øving 12

Simen Hustad

November 18, 2021

Dette dokumentet har til hensikt å bryte ned de individuelle kodesnuttene som kreves for hver oppgave på dataøvingen. All kode som står bak # er kommentarer i koden.

Eksempelkoden min her er langt ifra den mest effektive måten å løse problemene på, den er i stor grad ment å være forståelig og oversiktlig uavhengig av hvilket nivå man er på. Jeg antar også at dere vet hvordan å kalle (bruke) funksjoner i python, slik at det ikke dekkes i dette dokumentet.

Oppgave 1

a)

Oppgitt r_1 og r_2 er feil, skal være:

$$r_1 = r + (1, 0)$$
$$r_2 = r - (1, 0)$$

b)

For å kunne plotte et vektorfelt må vi først vite hvordan å plotte ved bruk av numpy og matplotlib.pyplot. Det første vi må gjøre er å importere pakkene:

```
import numpy as np
import matplotlib.pyplot as plt
```

Videre trenger vi å definere et koordinatsystem med punkter for å danne figuren. Ved å bruke np.linspace kan vi danne en liste med et bestemt antall punkter, hvor mellomrommet mellom hvert punkt er like stort. Vi setter derette en liste med punkter langs x-aksen sammen med en liste med punkter i y-aksen for å danne et koordinatsystem.

```
xpoints = np.linspace(-10, 10, 20)
ypoints = np.linspace(-10, 10, 20)
xp, yp = np.meshgrid(xpoints, ypoints)
```

Syntax:

```
np.linspace(minste verdi, st rste verdi, antall punkter)
np.meshgrid(akse 1, akse 2)
```

Merk: np.meshgrid danner et koordinatsystem mellom de bestemte rammene xpoints og ypoints

Koordinatsystemet vi har nå inneholder ingen informasjon eller verdier. Vi trenger å parametrisere funksjonen vår slik at vi kan legge til ønskede verdier for hvert punkt x og y. Numpy er ganske smart når det kommer til å jobbe med lister, slik at vi kan sette inn parametrisering direkte. Vi parametriserer $r = (x, y)$, og setter da inn i uttrykket for $v(r)$:

```
L = 3
M = 1
r = np.array([xp, yp])  #r = (x, y)
r1 = [r[0] + 1, r[1]] #r1 = r + (1, 0) = (x + 1, y)
r2 = [r[0] - 1, r[1]] #r2 = r - (1, 0) = (x - 1, y)
r1Len = np.sqrt(r1[0]**2 + r1[1]**2) #|r1|
r2Len = np.sqrt(r2[0]**2 + r2[1]**2) #|r2|
r1Hatt = r1/r1Len # r1Hatt = r1/|r1|
r2Hatt = r2/r2Len # r2Hatt = r2/|r1|
F = L/r1Len*r1Hatt - M/r2Len*r2Hatt #Setter inn i v
u = F[0]  # F rste koordinat til vektorfeltet
v = F[1]  # Andre koordinat til vektorfeltet
```

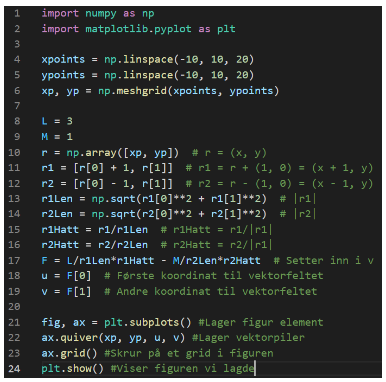
Merk: Vi bruker np.array() og ikke vanlige lister siden numpy har god integrasjon av den typen lister.

Det vi nå står igjen med er u og v som er koordinatene til vektorfeltet basert på koordinatsystemet vi lagde. Vi trenger nå å lage et plotte element fra matplotlib og legge til verdiene vi har funnet:

```
fig, ax = plt.subplots() #Lager figur element
```

```
ax.quiver(xp, yp, u, v) #Lager vektorpiler
ax.grid() #Skrur p  et grid i figuren
plt.show() #Viser figuren vi lagde
```

Koden og plottet du sitter igjen med burde se noe slik ut:



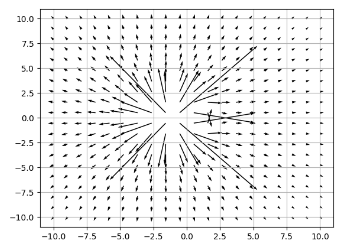
```
import numpy as np
import matplotlib.pyplot as plt

xpoints = np.linspace(-10, 10, 20)
ypoints = np.linspace(-10, 10, 20)
xp, yp = np.meshgrid(xpoints, ypoints)

L = 3
M = 1
r = np.array([xp, yp])  # r = (x, y)
r1 = [r[0] + 1, r[1]]  # r1 = r + (1, 0) = (x + 1, y)
r2 = [r[0] - 1, r[1]]  # r2 = r - (1, 0) = (x - 1, y)
r1Len = np.sqrt(r1[0]**2 + r1[1]**2)  # |r1|
r2Len = np.sqrt(r2[0]**2 + r2[1]**2)  # |r2|
r1Hatt = r1/r1Len  # r1Hatt = r1/|r1|
r2Hatt = r2/r2Len  # r2Hatt = r2/|r1|
F = L/r1Len*r1Hatt - M/r2Len*r2Hatt  # Setter inn i v
u = F[0]  # Første koordinat til vektorfeltet
v = F[1]  # Andre koordinat til vektorfeltet

fig, ax = plt.subplots() #Lager figur element
ax.quiver(xp, yp, u, v) #Lager vektorpiler
ax.grid() #Skrur på et grid i figuren
plt.show() #Viser figuren vi lagde
```

(a) Kodesnutt



(b) Figur

Figure 1: Plotting av vektorfelt

c)

Vi ønsker nå å tegne integralkurver gjennom et vektorfelt. Heldigvis har scipy integrerte funksjoner som hjelper oss med dette, spesifikt ønsker vi å bruke odeint. Vi starter med å importere odeint fra scipy.integrate og legger det øverst i dokumentet vårt.

```
from scipy.integrate import odeint
```

For å hindre at koden viser alt hver gang, samt å gjøre den mer oversiktlig tar vi først å pakker alt vi gjorde i oppgave b) inn i en funksjon. Vi kommer til å gjenbruke litt av koden, men for enkelthetsgrunn gjør vi det slik. Vi endrer ingenting på det som står.

```python
import numpy as np
import matplotlib.pyplot as plt


def OppgaveB():
    xpoints = np.linspace(-10, 10, 20)
    ypoints = np.linspace(-10, 10, 20)
    xp, yp = np.meshgrid(xpoints, ypoints)

    L = 3
    M = 1
    r = np.array([xp, yp])  # r = (x, y)
    r1 = [r[0] + 1, r[1]]  # r1 = r + (1, 0) = (x + 1, y)
    r2 = [r[0] - 1, r[1]]  # r2 = r - (1, 0) = (x - 1, y)
    r1Len = np.sqrt(r1[0]**2 + r1[1]**2)  # |r1|
    r2Len = np.sqrt(r2[0]**2 + r2[1]**2)  # |r2|
    r1Hatt = r1/r1Len  # r1Hatt = r1/|r1|
    r2Hatt = r2/r2Len  # r2Hatt = r2/|r1|
    F = L/r1Len*r1Hatt - M/r2Len*r2Hatt  # Setter inn i v
    u = F[0]  # Første koordinat til vektorfeltet
    v = F[1]  # Andre koordinat til vektorfeltet

    fig, ax = plt.subplots()  # Lager figur element
    ax.quiver(xp, yp, u, v)  # Lager vektorpiler
    ax.grid()  # Skrur på et grid i figuren
    plt.show()  # Viser figuren vi lagde
```

Figure 2: Oppgave b inne i en funksjon

For å bruke odeint trenger vi en funksjon som tar inn verdier for x og y og et tidsinterval t. Denne funksjonen gir da ut funksjonsverdiene vi leter etter. Vi definerer bare funksjonen som f med parametere pos og t. pos parameteren er en liste som inneholder x og y slik at $pos = [x, y]$.

Selv om dette høres komplisert og mystisk ut er det en trøst å vite at f gjør akkurat det samme som vi gjorde i b, bare for generelle verdier for x og y. Funksjonen blir da seende slik ut:

```
def f(pos, t):
    x = pos[0]
    y = pos[1]
    L = 3
    M = 1
    r = np.array([x, y])  # r = (x, y)
    r1 = [r[0] + 1, r[1]]  # r1 = r + (1, 0) = (x + 1, y)
    r2 = [r[0] - 1, r[1]]  # r2 = r - (1, 0) = (x - 1, y)
    r1Len = np.sqrt(r1[0]**2 + r1[1]**2)  # |r1|
    r2Len = np.sqrt(r2[0]**2 + r2[1]**2)  # |r2|
    r1Hatt = r1/r1Len  # r1Hatt = r1/|r1|
    r2Hatt = r2/r2Len  # r2Hatt = r2/|r1|
    F = L/r1Len*r1Hatt - M/r2Len*r2Hatt  # Setter inn i v
    u = F[0]  # Første koordinat til vektorfeltet
    v = F[1]  # Andre koordinat til vektorfeltet
    return u, v
```

Figure 3: Funksjonen f

Merk: t gjør ingenting i funksjonen, men er en nødvendig parameter når vi bruker odeint. Den bestemmer derimot hvor lang integralkurven skal være, men det skjer internt i odeint.

Nå trenger vi bare å lage kode som gjør det vi ønsker i forhold til oppgave c. Vi pakker dette også inn i en funksjon som vi kaller OppgaveC. Vi må også vite syntaxen for å bruke odeint.

```
sol = odeint(funksjon, [start x, start y], tidspunkt)
```

Merk: sol er bare et vanlig navn på returverdien til odeint. sol står for solution.

Vi trenger da å lage oss noen verdier vi kan putte inn i odeint. Oppgaven ber oss om å lage 3 kurver, og vi kan lage noen variabler med relevante verdier for eksempel slik:

```
#Integralkurver [start posisjon, tidsinterval]
kurve1 = [[-1, 0.5], [0, 10]]
kurve2 = [[-1, -0.5], [0, 6]]
kurve3 = [[-1.5, -0.5], [0, 3]]
```

Merk: Både startposisjonen og tidsintervallet er lister med to verdier. Start posisjonen har [x, y] og tidsintervallet er [start, stopp].

Videre trenger vi å bestemme verdier for tidsintervallet. Dette kan vi gjøre med vår gode venn np.linspace, og vi lager et tidsinterval for hver kurve (litt variasjon er morro :)). Vi bruker verdiene vi lagret i kurvevariablene våre.

```
#t = np.linspace(start tid, slutt tid, antall punkter)
t1 = np.linspace(kurve1[1][0], kurve1[1][1], 100)
t2 = np.linspace(kurve2[1][0], kurve2[1][1], 100)
t3 = np.linspace(kurve3[1][0], kurve3[1][1], 100)
```

Da har vi alt vi trenger for å kjøre odeint, og det gjøres slik:

```
# Finner punktene langs integralkurvene
# sol = odeint(funksjon, [start x, start y], tidspunkt)
sol1 = odeint(f, kurve1[0], t1)
```

```
sol2 = odeint(f, kurve2[0], t2)
sol3 = odeint(f, kurve3[0], t3)
```

Det vi står igjen med nå er punkter vi kan plotte:

```
fig, ax = plt.subplots()  # Lager figur element
ax.plot(sol1[:, 0], sol1[:, 1], color="red")
ax.plot(sol2[:, 0], sol2[:, 1], color="green")
ax.plot(sol3[:, 0], sol3[:, 1], color="blue")
ax.grid()  # Skrur p   et grid i figuren
plt.show() #Plotter figuren
```

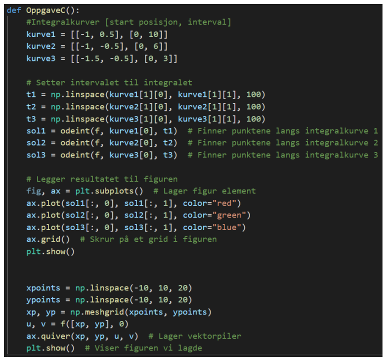
Merk: Syntaxen sol[:, 0] er skummel, men det er en forenklet versjon av å hente alle x-koordinatene.

Hvis vi ønsker å ha med vektorpilene i figuren kan vi legge til en kodesnutt fra b, og fjerner da den forrige plt.show()

```
xpoints = np.linspace(-10, 10, 20)
ypoints = np.linspace(-10, 10, 20)
xp, yp = np.meshgrid(xpoints, ypoints)
u, v = f([xp, yp], 0)
ax.quiver(xp, yp, u, v)  # Lager vektorpiler
plt.show()  # Viser figuren vi lagde
```

Husk: All koden vi har laget i denne oppgaven legger vi i en funksjon, OppgaveC(), i kronologisk rekkefølge.

Funksjonen OppgaveC og figuren vi nå står igjen med burde se ut som bildene under. Vi har også laget en funksjon f som vist i figur 3



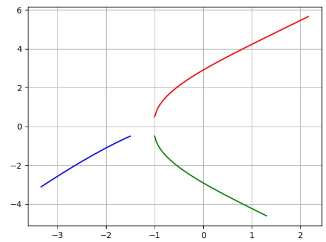
```python
def OppgaveC():
    #Integralkurver [start posisjon, interval]
    kurve1 = [[-1, 0.5], [0, 10]]
    kurve2 = [[-1, -0.5], [0, 6]]
    kurve3 = [[-1.5, -0.5], [0, 3]]

    # Setter intervalet til integralet
    t1 = np.linspace(kurve1[1][0], kurve1[1][1], 100)
    t2 = np.linspace(kurve2[1][0], kurve2[1][1], 100)
    t3 = np.linspace(kurve3[1][0], kurve3[1][1], 100)
    sol1 = odeint(f, kurve1[0], t1)  # Finner punktene langs integralkurve 1
    sol2 = odeint(f, kurve2[0], t2)  # Finner punktene langs integralkurve 2
    sol3 = odeint(f, kurve3[0], t3)  # Finner punktene langs integralkurve 3

    # Legger resultatet til figuren
    fig, ax = plt.subplots()  # Lager figur element
    ax.plot(sol1[:, 0], sol1[:, 1], color="red")
    ax.plot(sol2[:, 0], sol2[:, 1], color="green")
    ax.plot(sol3[:, 0], sol3[:, 1], color="blue")
    ax.grid()  # Skrur på et grid i figuren
    plt.show()


    xpoints = np.linspace(-10, 10, 20)
    ypoints = np.linspace(-10, 10, 20)
    xp, yp = np.meshgrid(xpoints, ypoints)
    u, v = f([xp, yp], 0)
    ax.quiver(xp, yp, u, v)  # Lager vektorpiler
    plt.show()  # Viser figuren vi lagde
```

(a) Kodesnutt



(b) Figur

Figure 4: Plotting av integralkurver

Merk: Jeg plottet uten vektorfelt, siden det er mer oversiktlig :).

d)

For å finne potensialfunksjonen kan vi løse uttrykket:

$$\phi(x, y) = \int_0^1 \vec{F}(\vec{r}(t)) \cdot \dot{\vec{r}}(t) dt$$

Vi endrer da $\vec{r} = (x, y)$ til $\vec{r} = (tx, ty)$ slik at $\vec{r}$ er en funksjon av t.

For å utføre denne oppgaven kan vi bruke funksjoner fra sympy biblioteket. Vi importerer det i toppen av koden og gir det forkortelsen sp:

```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
from scipy.integrate import odeint
```

Figure 5: Importerer sympy

Vi bruker koden fra funksjonen f som vi lagde tidligere, men modifiserer den til å inneholde $\vec{r} = (tx, ty)$ og tilpasse den bruker av sympy istedenfor numpy.

```
x, y, L, M, t = sp.symbols("x y L M t") #Definerer x, y, L, M  og t som symboler uten verdier

r = np.array([t*x, t*y])  # r = (x, y)
r1 = [r[0] + 1, r[1]]  # r1 = r + (1, 0) = (x + 1, y)
r2 = [r[0] - 1, r[1]]  # r2 = r - (1, 0) = (x - 1, y)
r1Len = sp.sqrt(r1[0]**2 + r1[1]**2)  # |r1|
r2Len = sp.sqrt(r2[0]**2 + r2[1]**2)  # |r2|
r1Hatt = [k/r1Len for k in r1]  # r1Hatt = r1/|r1|
r2Hatt = [k/r2Len for k in r2]  # r2Hatt = r2/|r1|
F = [L/r1Len*r1Hatt[i] - M/r2Len*r2Hatt[i] for i in range(2) ] # Setter inn i v
```

(a) For sympy (ny)

```
x = pos[0]
y = pos[1]
L = 3
M = 1
r = np.array([x, y])  # r = (x, y)
r1 = [r[0] + 1, r[1]]  # r1 = r + (1, 0) = (x + 1, y)
r2 = [r[0] - 1, r[1]]  # r2 = r - (1, 0) = (x - 1, y)
r1Len = np.sqrt(r1[0]**2 + r1[1]**2)  # |r1|
r2Len = np.sqrt(r2[0]**2 + r2[1]**2)  # |r2|
r1Hatt = r1/r1Len  # r1Hatt = r1/|r1|
r2Hatt = r2/r2Len  # r2Hatt = r2/|r1|
F = L/r1Len*r1Hatt - M/r2Len*r2Hatt  # Setter inn i v
u = F[0]  # Første koordinat til vektorfeltet
v = F[1]  # Andre koordinat til vektorfeltet
return u, v
```

(b) For numpy (gammel)

Figure 6: Sammenlikning av ny og gammel kode

Merk: Vi har byttet ut verdier til x, y, L, M, t med symboler. I tillegg er det små forskjeller i hvordan operasjonene gjennomføres. Dette kommer av at numpy kan bruke vanlig algebra med lister, men det kan ikke sympy.

Vi må også ha noen linjer for å løse og printe integralet vårt:

```
integrand = sum([F[i]*sp.diff(r[i], t) for i in range(2)])
phi = sp.simplify(sp.integrate(integrand, (t, 0, 1)))
print(phi)
```

Figure 7: Kodesnutt

Merk: Integranden er utregningen av skalarproduktet $\vec{F} \cdot \dot{\vec{r}}$

Integralet vi løser er ganske heftig, og programmet vil bruke noen sekunder på å beregne svaret. Ikke bli bekymret om du ikke får svaret med en gang :). Direkte kopiert fra python vil resultatet vi får se slik ut:

$$L * log((x * *2 + 2 * x + y * *2 + 1)/(x * *2 + y * *2))/2$$
$$- L * log(1/(x * *2 + y * *2))/2$$
$$- M * log((x * *2 - 2 * x + y * *2 + 1)/(x * *2 + y * *2))/2$$
$$+ M * log(1/(x * *2 + y * *2))/2$$

Merk: Svaret over er $\phi(x, y) = ...$, men siden uttrykket er så stort har jeg ikke skrevet $\phi(x, y)$.

Ettersom dette er innviklede og nøstede uttrykk klarer ikke sympy å forenkle det, men vi kan gjøre det selv ved å bruke logaritmeregning.

$$ln(a*b) = ln(a) + ln(b) \qquad ln(\frac{a}{b}) = ln(a) - ln(b)$$
$$ln(a^b) = b*ln(a) \qquad log = ln \text{ i sympy}$$

Først omgjør vi det kopierte uttrykket til vanlig skrivemåte:

$$\phi(x,y) = \frac{L}{2}ln\left(\frac{x^2+2x+y^2+1}{x^2+y^2}\right) - \frac{L}{2}ln\left(\frac{1}{x^2+y^2}\right) + \frac{M}{2}ln\left(\frac{1}{x^2+y^2}\right) - \frac{M}{2}ln\left(\frac{x^2-2x+y^2+1}{x^2+y^2}\right)$$

Videre tar vi å bruker logaritmeregler for å forenkle uttrykk:

$$-\frac{L}{2}ln\left(\frac{1}{x^2+y^2}\right) = \frac{L}{2}ln\left(\frac{1^{-1}}{(x^2+y^2)^{-1}}\right) = \frac{L}{2}ln(x^2+y^2)$$
$$\frac{M}{2}ln\left(\frac{1}{x^2+y^2}\right) = -\frac{M}{2}ln(x^2+y^2)$$

Forenkler leddene med L og M hver for seg:

$$\begin{aligned}
&\frac{L}{2}ln\left(\frac{x^2+2x+y^2+1}{x^2+y^2}\right) - \frac{L}{2}ln\left(\frac{1}{x^2+y^2}\right)\\
&= \frac{L}{2}ln\left(\frac{x^2+2x+y^2+1}{x^2+y^2}\right) + \frac{L}{2}ln(x^2+y^2)\\
&= \frac{L}{2}ln\left(\frac{(x^2+2x+y^2+1)(x^2+y^2)}{x^2+y^2}\right)\\
&= \frac{L}{2}ln(x^2+2x+1+y^2)\\
&= \frac{L}{2}ln((x+1)^2+y^2)
\end{aligned}$$

Gjør det samme for M:

$$
\begin{aligned}
&\frac{M}{2} ln \left(\frac{1}{x^2 + y^2} \right) - \frac{M}{2} ln \left(\frac{x^2 - 2x + y^2 + 1}{x^2 + y^2} \right) \\
&= -\frac{M}{2} ln(x^2 + y^2) - \frac{M}{2} ln \left(\frac{x^2 - 2x + y^2 + 1}{x^2 + y^2} \right) \\
&= -1 \left(\frac{M}{2} ln(x^2 + y^2) + \frac{M}{2} ln \left(\frac{x^2 - 2x + y^2 + 1}{x^2 + y^2} \right) \right) \\
&= -1 \left(\frac{M}{2} ln(x^2 - 2x + 1 + y^2) \right) \\
&= -\frac{M}{2} ln((x-1)^2 + y^2)
\end{aligned}
$$

Vi ender da med endelig uttrykk for potensialfeltet vårt:

$$\phi(x, y) = \frac{L}{2} ln((x+1)^2 + y^2) - \frac{M}{2} ln((x-1)^2 + y^2)$$

Den fullstendige kodesnutten vi nå har lagt til kan vi legge inn i en funksjon som kjøres separat fra alt vi har gjort tidligere. Vi kaller funksjonen OppgaveD og den burde se noe slik ut:

```
def OppgaveD():
    x, y, L, M, t = sp.symbols("x y L M t") #Definerer x, y, L, M  og t som symboler uten verdier

    r = np.array([t*x, t*y])  # r = (x, y)
    r1 = [r[0] + 1, r[1]]  # r1 = r + (1, 0) = (x + 1, y)
    r2 = [r[0] - 1, r[1]]  # r2 = r - (1, 0) = (x - 1, y)
    r1Len = sp.sqrt(r1[0]**2 + r1[1]**2)  # |r1|
    r2Len = sp.sqrt(r2[0]**2 + r2[1]**2)  # |r2|
    r1Hatt = [k/r1Len for k in r1]  # r1Hatt = r1/|r1|
    r2Hatt = [k/r2Len for k in r2]  # r2Hatt = r2/|r1|
    F = [L/r1Len*r1Hatt[i] - M/r2Len*r2Hatt[i] for i in range(2) ] # Setter inn i v

    integrand = sum([F[i]*sp.diff(r[i], t) for i in range(2)])
    phi = sp.simplify(sp.integrate(integrand, (t, 0, 1)))
    print(phi)
```

Figure 8: Fullstendig kodesnutt for d

e)