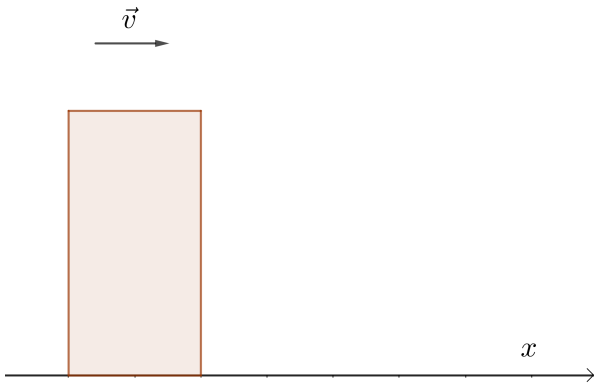
Oppgavesett uke 8

"Test deg selv"-oppgaver

Disse oppgavene er fine å gjøre hjemme før du kommer på gruppetimen. Om du har spørsmål til disse oppgavene, ta dem gjerne opp på ekstrahjelpgruppa.

TDS 8.1 Eskild Eskesen på joggetur

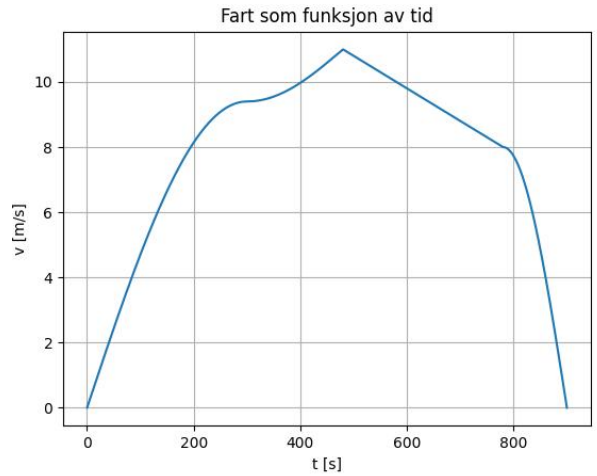
I denne oppgaven skal vi se på hastigheten til en person som har vært på joggetur.



Figuren over viser en noe forenklet person, Eskild Eskesen, som er ute og jogger. Vi ser kun på bevegelsen i en dimensjon (x-retning). Vi skal finne akselerasjonen til Eskild numerisk, og strekningen han har tilbakelagt, ved å bruke den målte hastigheten til Eskild i filen "velocity.txt". Vi skal også analysere og visualisere (plotte) resultatet. Deretter ser vi på et datasett med litegrann støy for å se hvordan det påvirker de numeriske resultatene.

a) Skriv et program som leser inn hastigheten og de tilhørende tidene fra filen "velocity.txt", og plott hastigheten som funksjon av tid. for hvilke tider er det Eskild akselererer negativt og positivt? Hvor lang tid varte joggeturen? Husk aksetitler og enheter!

Løsning: Eksempel-koder for alle løsningene finner dere i appendikset bakerst.

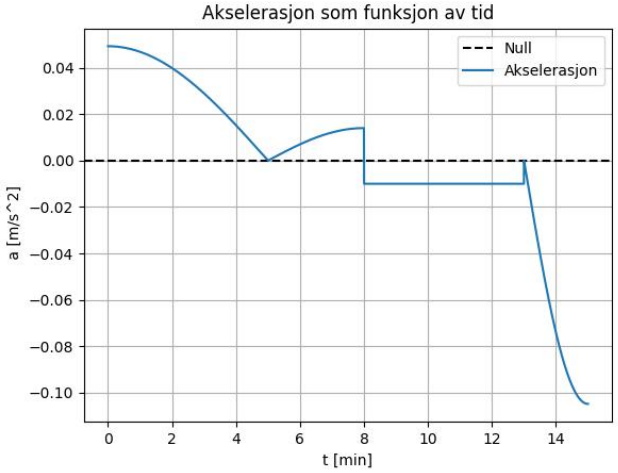


Vi kan se fra figuren at i begynnelsen så er akselerasjonen til Eskild positiv, siden hastigheten blir større. Akselerasjonen avtar fram til ca 300 s siden hastigheten øker saktere. Ved omtrent 300 sekunder øker akselerasjonen igjen. Deretter er akselerasjonen negativ fra ca 500 sekunder og ut joggeturen.

Fra koden så ser vi at joggeturen til Eskild varte i 15 minutter.

b) Derivér numerisk hastigheten til Eskild og plott akselerasjonen som funksjon av tid. Stemmer resultatet for akselerasjonen med tolkningen din i forrige deloppgave?

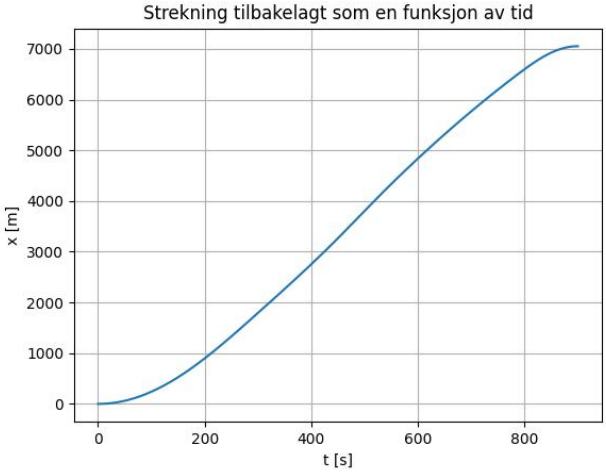
Løsning:



Vi kan se at akselerasjonen stemmer godt med det vi antok fra kun å studere hastighetsgrafen.

c) Integrer numerisk hastigheten for å finne strekningen som funksjon av tid. Plott resultatet. Hva var den totale distansen Eskild tilbakela på joggeturen? Er dette en realistisk distanse gitt den totale tiden som joggeturen varte?

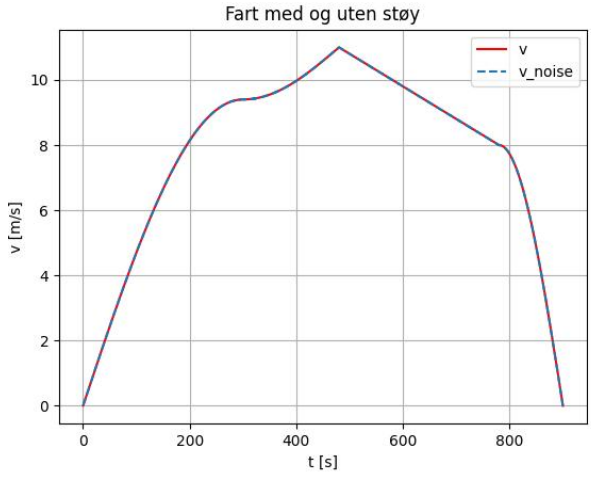
Løsning:

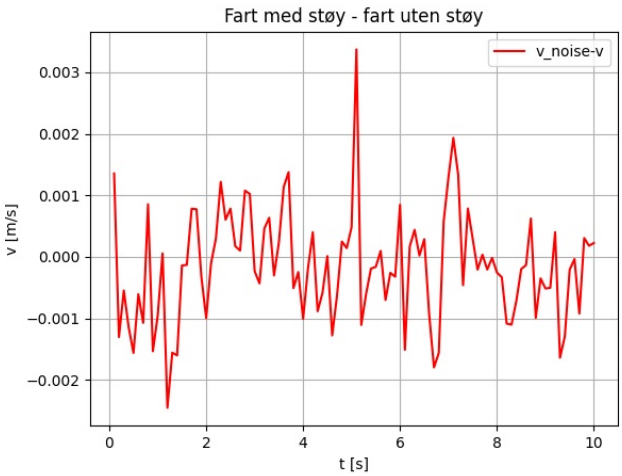


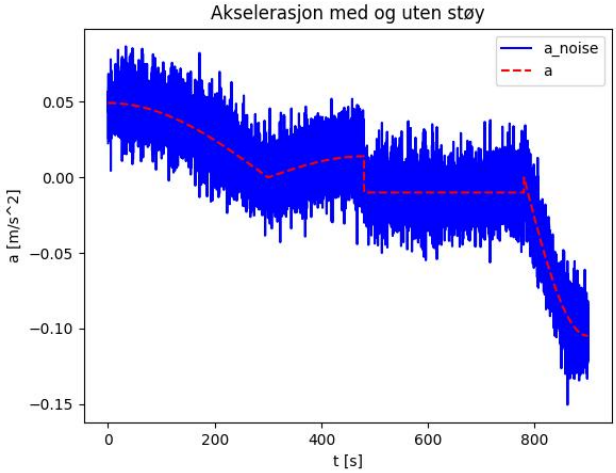
Fra koden ser vi at den totale distansen tilbakelagt var ca. 7 km. Å løpe 7 km på 15 minutter er veldig raskt, det tilsvarer ca. 2 minutter og 9 sekunder på kilometeren! Til sammenligning: verdensrekorden på 10000m på bane er 26 minutter og 11 sekunder, som blir omtrent 2 minutter og 36 sekunder på kilometeren. Så det er bra med futt i Eskild!

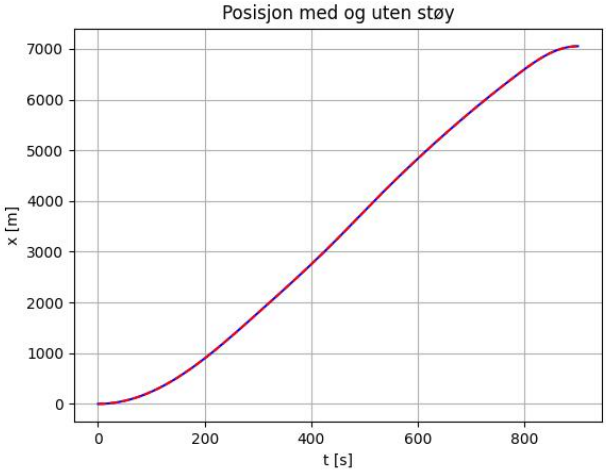
d) Gjør deloppgave a, b og c igjen, men nå med filen "velocitynoise.txt". Sammenlign resultatene med og uten støy ved å plotte dem sammen. Hvor stor er den typiske størrelsen av støyen i hastigheten? Ser dere en forskjell i de numeriske beregningene når man legger til litt støy? Hva er absoluttverdien av differansen mellom tilbakelagt strekning med og uten støy? Kommenter resultatene.

Løsning:









Støyen er relativt liten i forhold til størrelsen på hastigheten, noe som gjør at vi ikke ser forskjellen når vi plotter hastighet med og uten støy sammen. For å se hvor stor støy det er, plotter vi

forskjellen mellom hastighet med og uten støy, og sitter igjen med bare støyen. Da ser vi at den er like mye positiv og negativ, og med typiske verdier rundt 0.001 m/s. Dette gjenspeiler seg også i posisjonsberegningene, hvor støyen ikke gir et særlig stort bidrag. Dette er fordi støyen har like stor sannsynlighet for å være positiv som negativ, og når vi integrerer får vi derfor noen ganger en litt for stor verdi for funksjonen og noen ganger en litt for liten verdi. Når vi summerer sammen disse bidragene kansellerer de positive og negative støyverdiene hverandre, og absoluttverdien av differansen mellom den tilbakelagte strekningen med og uten støy er omtrent $|\Delta x| = 0.006$ m. Akselerasjonen er derimot den deriverte av hastigheten, og selv om støyen er liten i verdi, deles hastighetsdifferansen med det lille tidsintervallet mellom hvert beregningspunkt, og da kan feilen bli stor.

Løsning:

Oppgave TDS8.1a

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  v = []
5  t = []
6  with open("velocity.txt", "r") as infile:
7      f = infile.readline()
8      f = infile.readline()
9      for line in infile:
10         a = line.split(sep=", ")
11         v.append(float(a[0]))
12
13         t.append(float(a[1]))
14
15 v = np.array(v)
16 t = np.array(t)
17
18
19
20
21
22 if __name__ == "__main__":
23     plt.grid()
24     plt.plot(t,v)
25     plt.ylabel("v [m/s]")
26     plt.xlabel("t [s]")
27     plt.title("Fart som funksjon av tid")
28     plt.show()
29     print(f"Joggeturen varte i {t[-1] / 60:g} minutter")
```

Oppgave TDS8.1b

```python
import numpy as np
import matplotlib.pyplot as plt
from S1a import v,t

def num_deriv(x,t):
    dxdt = []
    for i in range(len(x)-1):
        deriv = (x[i+1]-x[i])/(t[i+1]-t[i])
        dxdt.append(deriv)
    return np.array(dxdt), t[0:len(dxdt)]

a,t_a = num_deriv(v,t)
if __name__ == "__main__":
    plt.axhline(y=0,linestyle = "--",color = "k", label = "Null")
    plt.plot(t_a/60,a, label = "Akselrasjon")
    plt.legend()
    plt.grid()
    plt.xlabel("t [min]")
    plt.ylabel(" a [m/s^2]")
    plt.title("Akselerasjon som funksjon av tid")
    plt.show()
```

Oppgave TDS8.1c

```python
import numpy as np
import matplotlib.pyplot as plt
from S1a import v,t



def num_int(v,t):
    x = np.zeros(len(v))
    for i in range(len(v)-1):
        dt = t[i+1]-t[i]
        x[i+1] = x[i]+v[i]*dt
    return x
x = num_int(v,t)
if __name__ == "__main__":
    plt.plot(t,x)
    plt.grid()
    plt.xlabel("t [s]")
    plt.ylabel("x [m]")
    plt.title("Strekning tilbakelagt som en funksjon av tid")
    plt.show()

    print(f"Den totale strekningen tilbakelagt var {x[-1]:g}m")
```

Oppgave TDS8.1d

```python
import numpy as np
import matplotlib.pyplot as plt
from S1a import v,t
from S1b import a,t_a,num_deriv
from S1c import num_int,x


# Importerer datapunktene med støy
vs = []
ts = []
with open("velocitynoise.txt", "r") as infile:
    f = infile.readline()
    f = infile.readline()
    for line in infile:
        linesegments = line.split(sep=", ")
        vs.append(float(linesegments[0]))

        ts.append(float(linesegments[1]))

vs = np.array(vs)
ts = np.array(ts)



a_noise,t_noise =num_deriv(vs,ts)

#Plotter de to hastighetene
plt.plot(t,v,"r",label = "v")
plt.plot(ts,vs,"--",label = "v_n")
plt.grid()
plt.ylabel("v [m/s]")
plt.xlabel("t [s]")
plt.title("Fart med og uten støy")
plt.legend()
plt.show()



#Plotter de to akselerasjonene
plt.plot(t_noise,a_noise,"b",label = "a_noise")
plt.plot(t_a,a,"--r",label = "a")
plt.axis()
plt.legend()
plt.ylabel("a [m/s^2]")
plt.xlabel("t [s]")
plt.title("Akselerasjon med og uten støy")
plt.show()

#Plotter posisjonen med og uten støy
x_noise = num_int(vs,ts)
```

```
51 plt.plot(t,x_noise,"b",label = "x_noise")
52 plt.plot(t,x,"--r",label = "x")
53 plt.xlabel("t [s]")
54 plt.ylabel("x [m]")
55 plt.grid()
56 plt.title("Posisjon med og uten støy")
57 plt.show()
58
59 print(f"Differansen mellom tilbakelagt strekning med og uten støy er↩
       {np.abs(x[-1]-x_noise[-1])}m")
```

Fasit til "Test deg selv"-oppgaver

TDS 8.1

a) Joggeturen til Eskild varte i 15 minutter.

c) Ca. 7.0 km.

Gruppeoppgaver

G 8.1 Eksamenstrening

Siden dette er uka før midtveis-eksamen, starter vi gruppetimen med litt eksamenstrening. Gruppelæreren vil dele ut flervalgsoppgaver fra en tidligere eksamen. Dere besvarer først oppgavene på egen hånd, med en *kort* forklaring og begrunnelse for svaralternativet dere velger. Så går dere sammen to og to (evt. tre), og diskuterer svarene, spesielt der dere har svart forskjellig. Tilslutt vil gruppelæreren dele ut løsningsforslag, og eventuelt gjennomgå den/de oppgavene som viste seg å være vanskelige.

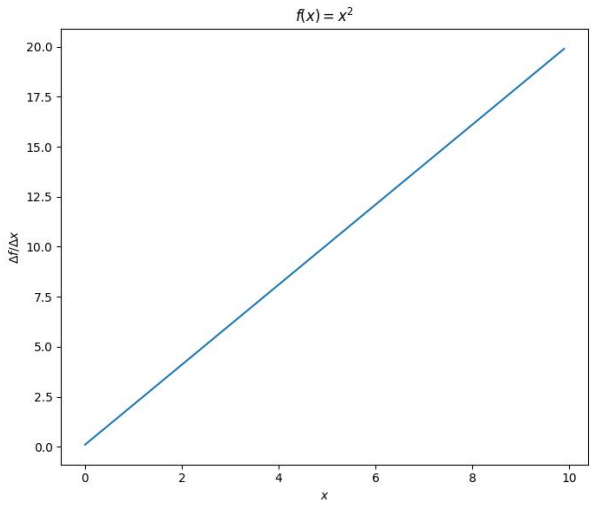
G 8.2 Numerisk derivasjon

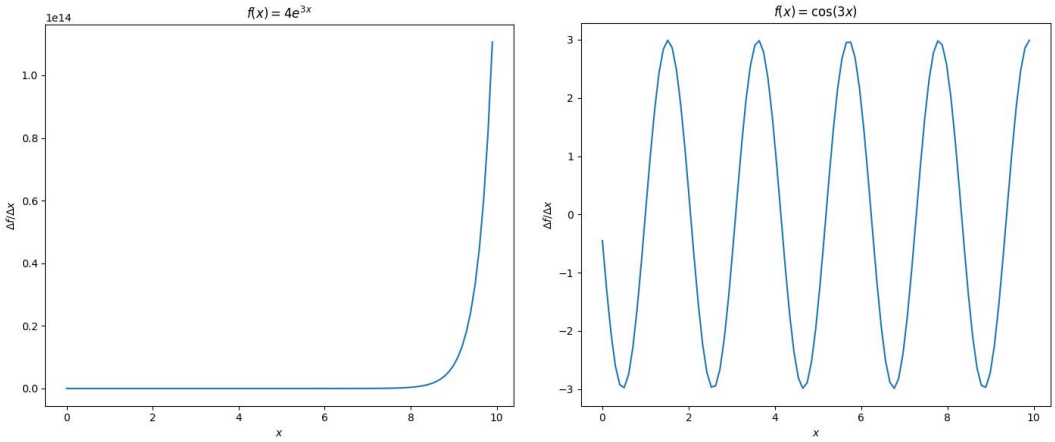
a) Lag et array $x \in [0, 10]$ med $n = 100$ punkter og deriver numerisk følgende funksjoner:

- $f(x) = x^2$
- $f(x) = 4e^{3x}$
- $f(x) = \cos 3x$

Plott resultatene. Stemmer det med det du forventet?

Løsning: Vi forventer er at den deriverte av den første funksjonen kommer til å bli lineær, siden vi deriverer et polynom av andre grad. Den deriverte av den andre funksjonen kommer til å bli en eksponentialfunksjon med større koeffisient enn den originale funksjonen, og den siste funksjonens deriverte kommer til å bli en sinusfunksjon med negativt fortegn.





Vi ser at plottene av de deriverte funksjonene oppfører seg som vi forventet.

```
# Gruppeoppgaver uke 08, v2023
# Basert på versjon fra h2022
# Ann-Cecilie Larsen, 09 March 2023
# a.c.larsen@fys.uio.no
# Last version: 09 March 2023
import numpy as np
import matplotlib.pyplot as plt

# En Python-funksjon for numerisk derivasjon.
# Den tar som input to arrays med verdier:
# funksjonen x(t) og argumentet t.
# Disse kunne også  kalles f(x) og x;
# dette er vilkaarlige inputfunksjoner og argumenter.
def differentiation(x,t):
    dxdt = np.zeros(len(x)-1) # lage array med nuller,  riktig lengde
    for i in range(len(x)-1): # her kommer loopen med derivasjonen
        dt = t[i+1]-t[i]     # regne ut tidsintervallet i hvert steg
        dxdt[i] = (x[i+1]-x[i])/dt  # regne ut den deriverte i punkt [i]
        # returnerer den deriverte for det gitte inputargumentet,
        # og  riktig lengde på argument-array
    return dxdt , t[:len(dxdt)]

# Den foerste funksjonen er f(x) = x**2
n = 100 # Antall x-punkter vi vil bruke
x = np.linspace(0, 10, n) # array med verdier for x
f = x**2    # funksjonen som skal deriveres

dfdx, new_x = differentiation(f,x)

# Lage plott
fig, ax = plt.subplots(nrows=1, figsize=(7, 6))
plt.title("$f(x) = x^2$")
```

```
ax.plot(new_x, dfdx)
ax.set_xlabel("$x$")
ax.set_ylabel("$\Delta f/\Delta x$")
fig.tight_layout()
fig.savefig("diff_2a1.jpg")

# Den andre funksjonen er f(x) = 4e^{3x}
x_2 = np.linspace(0, 10, n) # array med verdier for x
f_2 = 4*np.exp(3*x_2)   # funksjonen som skal deriveres

dfdx_2, new_x_2 = differentiation(f_2,x_2)

# Lage plott
fig2, ax2 = plt.subplots(nrows=1, figsize=(7, 6))
plt.title("$f(x) = 4e^{3x}$")
ax2.plot(new_x_2, dfdx_2)
ax2.set_xlabel("$x$")
ax2.set_ylabel("$\Delta f/\Delta x$")
fig2.tight_layout()
fig2.savefig("diff_2a2.jpg")

# Den tredje funksjonen er f(x) = cos(3x)
x_3 = np.linspace(0, 10, n) # array med verdier for x
f_3 = np.cos(3*x)   # funksjonen som skal deriveres

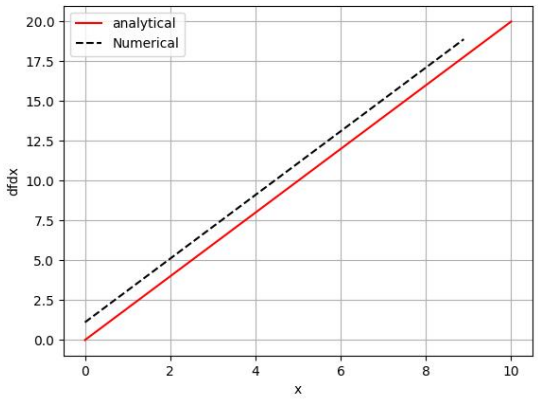
dfdx_3, new_x_3 = differentiation(f_3,x_3)

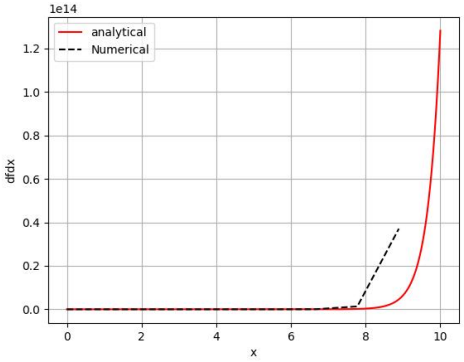
# Lage plott
fig3, ax3 = plt.subplots(nrows=1, figsize=(7, 6))
plt.title("$f(x) = \cos(3x)$")
ax3.plot(new_x_3, dfdx_3)
ax3.set_xlabel("$x$")
ax3.set_ylabel("$\Delta f/\Delta x$")
fig3.tight_layout()
fig3.savefig("diff_2a3.jpg")

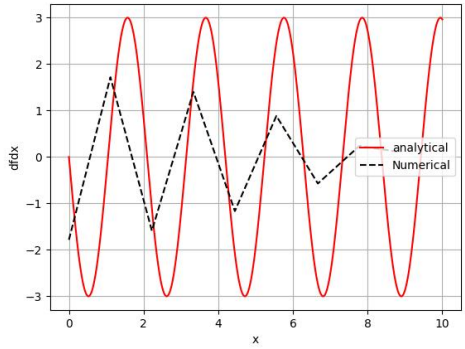
plt.show()
```

b) Gjør oppgave a) på nytt, men endre antallet punkter i arrayet x til 10, og deretter 10000. Ser du en forskjell i resultatene hvis du sammenligner resultatet med de analytisk deriverte av funksjonene?

Løsning: Vi ser først på løsningene med $n = 10$.

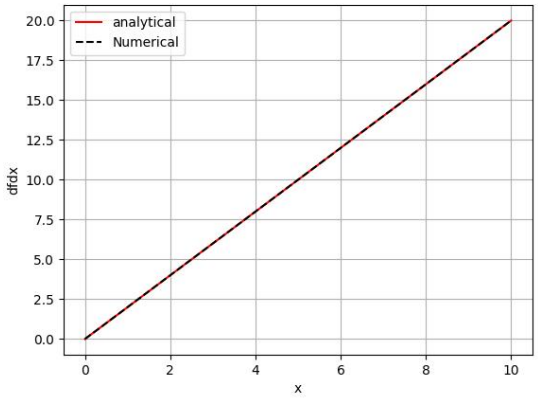


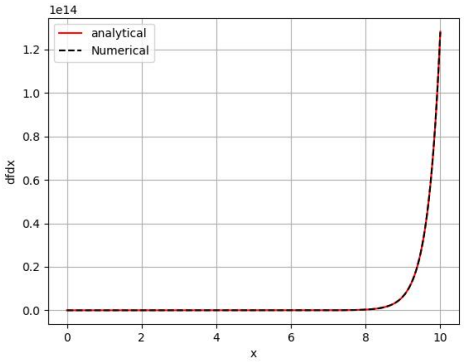


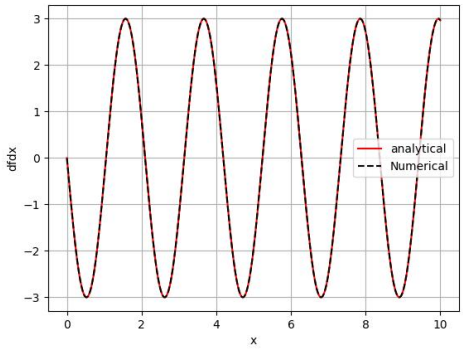


Vi ser at for lave antall datapunkter så er deriveringen svært unøyaktig, spesielt for harmoniske funksjoner (svingninger).

Deretter ser vi på løsninger for $n = 10000$.







For store antall n blir feilen til den numeriske derivasjonen mindre. Dette er fordi trunkeringsfeilen minimeres uten at den numeriske feilen som kommer fra avrundinger får et stort bidrag.

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


def differentiation(x,t):
    dxdt = np.zeros(len(x)-1)
    for i in range(len(x)-1):
        dt = t[i+1]-t[i]
        dxdt[i] = (x[i+1]-x[i])/dt
    return dxdt , t[:len(dxdt)]

def diffplot(n,f):
    x = np.linspace(0, 10, n)
    func = f(x)
    dfdx, x2 = differentiation(func, x)
    plt.plot(x2, dfdx,"--k", label ="Numerical")
    plt.ylabel("dfdx")
    plt.xlabel("x")
    plt.legend()
    plt.grid()


# --- Task c ---
for j in [10,10000]:
    n = 1000
    x = np.linspace(0, 10, n)


    def f(x):
        return x ** 2




    plt.plot(x, 2 * x,"r",label = "analytical")

```

```
37      diffplot(j, f)
38      plt.savefig(f"diff_1c1_n{j}.jpg")
39      plt.show()
40
41
42      def f(x):
43          return 4 * np.exp(3 * x)
44
45
46
47
48      plt.plot(x, 12 * np.exp(3 * x),"r",label = "analytical")
49      diffplot(j, f)
50      plt.savefig(f"diff_1c2_n{j}.jpg")
51      plt.show()
52
53
54      def f(x):
55          return np.cos(3 * x)
56
57
58
59
60      plt.plot(x, -3 * np.sin(3 * x),"r",label = "analytical")
61      diffplot(j, f)
62      plt.savefig(f"diff_1c3_n{j}.jpg")
63      plt.show()
```

G 8.3 Trapesmetoden

Nå skal vi bruke trapesmetoden for numerisk integrasjon, hvor integralet I kan regnes ut numerisk ved

$$I = \sum_{i=1}^{n} \frac{f(x_i) + f(x_{i-1})}{2} \Delta x \tag{1}$$

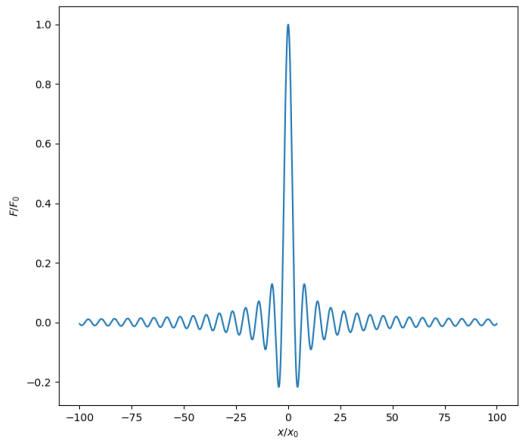
Vi ser på en partikkel som er påvirket av posisjonsavhengig kraft som kan beskrives ved:

$$F(x) = F_0 \frac{\sin x/x_0}{x/x_0},$$

hvor F_0 er en konstant med enhet N og x en lengde med enhet m.

a) Plott kraften $F(x)/F_0$ i intervallet $x \in [-100x_0, 100x_0]$. Du kan gjerne sette $F_0 = 1.00$ N og $x_0 = 1.00$ m.

Løsning:

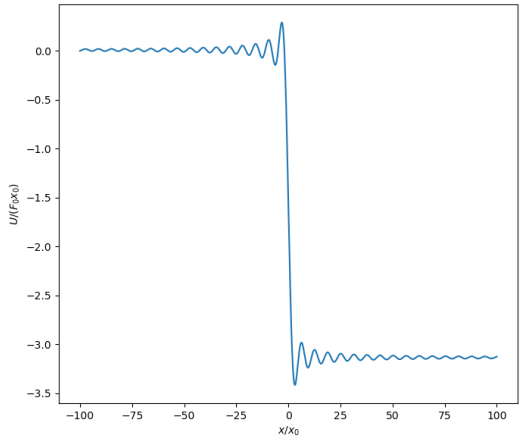


b) Finn arbeidet som kraften gjør når partikkelen beveger seg fra $x = -100x_0$ til $x = 100x_0$. (Tips: Du kan skrive din egen Python-funksjon som beregner arbeidet eller bruke scipy-funksjonen `trapz`).

Løsning: $W = 3.12$ J.

c) Vi definerer nullpunktet i potensiell energi som $U(-100x_0) = 0$ J. Plott den potensielle energien til partikkelen i intervallet $x \in [-100x_0, 100x_0]$. (Tips: Du kan skrive din egen Python-funksjon som beregner den potensielle energien eller bruke scipy-funksjonen `cumulative_trapezoid`.)

Løsning:



Løsning: Kode:

```python
# Python-script
# Group exercise G3,  24-28 February 2020
# A particle with a position-dependent force
# acting on it: F(x) = F0 * sin(x/x0)/(x/x0)
# F0 is a constant in units of N and x0 is a constant length in units of m.
# We want to plot the force normalized by the constant F0: F/F0 versus x/x0
# Cecilie, Feb 26, 2020
# Last update: Mar 06, 2026 cumtrapz->cumulative_trapezoid
# a.c.larsen@fys.uio

# Import Python libraries
import numpy as np
#from scipy import integrate # The cumtrapz function does not work for newer Scipy libraries...
from scipy.integrate import cumulative_trapezoid
import matplotlib.pyplot as plt

# Define constants.
# We have not been given numbers for x0 and F0, so I choose them to be 1.
n    = 100000 # number of steps
x0   = 1. # defining the constant length as 1.0 m
F0   = 1. # defining the force constant F0 as 1.0 N

# Define the arrays we need for the position x and the potential energy U
x = (1./x0)*np.linspace(-100,100,n)
U = np.zeros(n)

# Define the function for the force, this is now actually F/F0
F = (1./F0)*np.sin(x)/x

# Now we plot the force as a function of the position x
# See https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplots.html
# This is part a.
fig, ax = plt.subplots(nrows=1, figsize=(7, 6))
ax.plot(x[0:n-1], F[0:n-1])
ax.set_xlabel("$x/x_{0}$")
ax.set_ylabel("$F/F_{0}$")
fig.tight_layout()
fig.savefig("kraft.png")

# Part b.
# Here we are asked to find the work done by the force over the distance
# x/x0 = [-100, 100] -> conveniently the range we have defined the vector x :)
# We assume this is the only force acting on the particle.
# Because this is a position-dependent force only, we can use that
```

```
45 # W_{0,1} = \int F(x)dx where the integral goes over x = [-100x0, 100x0]
46 # Here we use the function trapz from scipy to integrate and get the total work done:
47 # https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.trapz.html
48 # https://en.wikipedia.org/wiki/Trapezoidal_rule
49 W01 = np.trapz(F,x)
50 print('Work (in J): W = ',W01)
51
52 # Part c.
53 # Here we will look at the potential energy as a function of position.
54 # We use the definition
55 # of the potential energy for a position-dependent, conservative force:
56 # U(x) = U(x0) - \int_{x0}^{x} F(x) dx
57 # We choose that U(x0) = U(-100) = 0 J
58 # => U(x) = -  \int_{x0 = -100}^{x=100} F(x) dx
59 # => U(x) = - \int_{x0 = -100}^{x=100} F0*sin(x/x0)/(x/x0) dx
60 # For this, we use the cumtrapz scipy function:
61 # https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.cumtrapz.html
62 # https://en.wikipedia.org/wiki/Trapezoidal_rule
63 # This gives us the integral for each small trapezoid for each interval (step dx) of x
64 #U = integrate.cumtrapz(-F,x) # THis does not work anymore for newer Scipy releases
65
66 U = cumulative_trapezoid(-F,x)
67
68 fig2, ax2 = plt.subplots(nrows=1, figsize=(7, 6))
69 ax2.plot(x[0:n-1], U[0:n-1])
70 ax2.set_xlabel("$x/x_{0}$")
71 ax2.set_ylabel("$U/(F_{0}x_{0})$")
72 fig2.tight_layout()
73 fig2.savefig("potensiell_energi.png")
74
75 plt.show()
```

Fasit til gruppeoppgaver

G 8.3

b) $W = 3.12$ J.