

# **Forelesning IN1900 – 26 September 2023**

**Ole Christian Lingjærde  
Institutt for Informatikk, Universitetet i Oslo**

**Uke: 25 September - 1 Oktober, 2023**

# Ukens agenda

I dag:

- Kort repetisjon av arrayer og plotting
- Mer om animering
- Plenumsøvelser 5.29, 5.39

Onsdag:

- Differenslikninger
- Plenumsøvelser A.1 og A.4

# Tupler, lister eller arrayer?

Alle tre brukes til å lagre mange verdier.

## **Lister:**

- Veldig fleksible (kan endres)
- Operasjoner gjøres ett element av gangen

## **Arrayer:**

- Relativt fleksible (kan endres, men kun én datatype)
- Vektoriserte operasjoner

## **Tupler:**

- Lite fleksible (kan ikke endres)
- Operasjoner gjøres ett element av gangen

# Hvordan bruke numpy-funksjoner

## Uskreven regel:

Bruk `import numpy as np` og referer til numpy-funksjoner som `np.linspace(..)`, `np.zeros(..)`

## Unntak:

For matematiske funksjoner (`sin`, `cos`, `log`, ...) kan du bruke `from numpy import sin, cos` og referere til dem som `sin(..)`, `cos(..)`, etc.

For flere detaljer, se læreboka (5th ed.), side 235 og 243.

# Vektorisering

Anta at vi skal regne ut  $\sin(x)$  for  $x=0, 0.1, 0.2, \dots, 1$ .

## A) Bruk av lister (ikke-vektorisert):

```
from math import sin
x = [k/10 for k in range(11)]
y = [sin(e) for e in x]
```

# Vektorisering

Anta at vi skal regne ut  $\sin(x)$  for  $x=0, 0.1, 0.2, \dots, 1$ .

## A) Bruk av lister (ikke-vektorisert):

```
from math import sin
x = [k/10 for k in range(11)]
y = [sin(e) for e in x]
```

## B) Bruk av numpy-arrays (ikke-vektorisert):

```
from math import sin
import numpy as np
x = np.linspace(0,1,11)
y = np.zeros(11)
for k in range(11):
    y[k] = sin(x[k])
```

# Vektorisering

Anta at vi skal regne ut  $\sin(x)$  for  $x=0, 0.1, 0.2, \dots, 1$ .

## A) Bruk av lister (ikke-vektorisert):

```
from math import sin
x = [k/10 for k in range(11)]
y = [sin(e) for e in x]
```

## B) Bruk av numpy-arrays (ikke-vektorisert):

```
from math import sin
import numpy as np
x = np.linspace(0,1,11)
y = np.zeros(11)
for k in range(11):
    y[k] = sin(x[k])
```

## C) Bruk av numpy-arrays (vektorisert)

```
import numpy as np
x = np.linspace(0,1,11)
y = np.sin(x)
```

# Hvor rask er vektorisert kode?

## Ikke-vektorisert kode:

```
def f_list(N):  
    import math  
    x = [0]*N; y = [0]*N; z = [0]*N  
    for i in range(N):  
        x[i] = 1 + i**2  
    for i in range(N):  
        y[i] = 1 + i * x[i] - math.tanh(x[i])  
    for i in range(N):  
        z[i] = abs(y[i])  
    return z
```



# Hvor rask er vektorisert kode?

## Ikke-vektorisert kode:

```
def f_list(N):
    import math
    x = [0]*N; y = [0]*N; z = [0]*N
    for i in range(N):
        x[i] = 1 + i**2
    for i in range(N):
        y[i] = 1 + i * x[i] - math.tanh(x[i])
    for i in range(N):
        z[i] = abs(y[i])
    return z
```

## Vektorisert kode:

```
def f_array(N):
    import numpy as np
    x = 1 + np.arange(N)**2
    y = 1 + np.arange(N) * x - np.tanh(x)
    z = np.abs(y)
    return z
```

## Sammenlikning av CPU-tid:

```
import time

t0 = time.clock()
f_list(10**7)
t1 = time.clock()
print(f'Nonvectorized: {t1-t0:.2f} seconds')

t0 = time.clock()
f_array(10**7)
t1 = time.clock()
print(f'Vectorized: {t1-t0:.2f} seconds')
```

```
Terminal> python compare_time.py
Nonvectorized: 6.67 seconds
Vectorized: 0.29 seconds
```

**Den vektoriserte (numpy) løsningen er 23 ganger raskere!**

# Konvertere mellom array og liste

*Fra liste til array:*

```
x = [1,2,3]
a = np.array(x)
```

*Fra array til liste:*

```
a = np.linspace(0, 5, 100)
x = list(a)
```

Liste	Array
<code>x = [1,2,3,4]</code>	<code>x = np.array([1,2,3,4])</code>
<code>x = [0]*n</code>	<code>x = np.zeros(n)</code>
<code>x = [1]*n</code>	<code>x = np.ones(n)</code>
<code>x = range(n)</code>	<code>x = np.arange(n)</code>
<code>y = x[:]</code>	<code>y = x.copy()</code>
<code>y = x + z</code>	<code>y = np.append(x,z)</code>
<code>y = [e+f for e,f in zip(x,z)]</code>	<code>y = x + z</code>
<code>h = (b-a)/(n-1)</code> <code>x = [a+i*h for i in range(n)]</code>	<code>x = np.linspace(a,b,n)</code>
<code>y = [e**2 + 2*e for e in x]</code>	<code>y = x**2 + 2*x</code>

Hva skrives ut?

```
import numpy as np

def delta(x):
    return x[1:]-x[:-1]

x = np.array([1, 2, 7, 9, 13, 63, 70])
print(delta(x))
```

Hva skrives ut?

```
import numpy as np

def delta(x):
    return x[1:]-x[:-1]

x = np.array([1, 2, 7, 9, 13, 63, 70])
print(delta(x))
```

Utskrift: [ 1 5 2 4 50 7]

Hva skriver dette programmet til fil?

```
import numpy as np
n = 11
x = np.linspace(0,1,n)
y = np.ones(n)
z = x + 2*y
with open('Fil.txt', 'w') as outfile:
    for e in z:
        outfile.write(f'{e}\n')
outfile.close()
```

2.0

2.1

2.2

2.3

2.4

2.5

2.6

2.7

2.8

2.9

3.0



## Quiz

Dette programmet skal skrive ut tallene

$\sin(0)$ ,  $\sin(0.1)$ ,  $\sin(0.2)$ , ...,  $\sin(10.0)$

på separate linjer i filen "Fil.txt". Finn feilene!

```
n = 100
x = np.linspace(0,10,n)
y = np.sin(x)
with open('Fil.txt') as outfile:
    for e in y:
        outfile.write(e)
outfile.close()
```

## Quiz

Dette programmet skal skrive ut tallene

$\sin(0)$ ,  $\sin(0.1)$ ,  $\sin(0.2)$ , ...,  $\sin(10.0)$

på separate linjer i filen "Fil.txt". Finn feilene!

```
n = 100 # Skal være n = 101
x = np.linspace(0,10,n)
y = np.sin(x)
with open('Fil.txt') as outfile:
    for e in y:
        outfile.write(e)
outfile.close()
```

## Quiz

Dette programmet skal skrive ut tallene

$\sin(0)$ ,  $\sin(0.1)$ ,  $\sin(0.2)$ , ...,  $\sin(10.0)$

på separate linjer i filen "Fil.txt". Finn feilene!

```
n = 100 # Skal være n = 101
x = np.linspace(0,10,n) # Vi har ikke importert numpy
y = np.sin(x)
with open('Fil.txt') as outfile:
    for e in y:
        outfile.write(e)
outfile.close()
```

# Quiz

Dette programmet skal skrive ut tallene

$\sin(0)$ ,  $\sin(0.1)$ ,  $\sin(0.2)$ , ...,  $\sin(10.0)$

på separate linjer i filen "Fil.txt". Finn feilene!

```
n = 100 # Skal være n = 101
x = np.linspace(0,10,n) # Vi har ikke importert numpy
y = np.sin(x) # Samme feil her
with open('Fil.txt') as outfile:
    for e in y:
        outfile.write(e)
outfile.close()
```

## Quiz

Dette programmet skal skrive ut tallene

$\sin(0)$ ,  $\sin(0.1)$ ,  $\sin(0.2)$ , ...,  $\sin(10.0)$

på separate linjer i filen "Fil.txt". Finn feilene!

```
n = 100 # Skal være n = 101
x = np.linspace(0,10,n) # Vi har ikke importert numpy
y = np.sin(x) # Samme feil her
with open('Fil.txt') as outfile: # Må ha med 'w'
    for e in y:
        outfile.write(e)
outfile.close()
```

# Quiz

Dette programmet skal skrive ut tallene

$\sin(0)$ ,  $\sin(0.1)$ ,  $\sin(0.2)$ , ...,  $\sin(10.0)$

på separate linjer i filen "Fil.txt". Finn feilene!

```
n = 100 # Skal være n = 101
x = np.linspace(0,10,n) # Vi har ikke importert numpy
y = np.sin(x) # Samme feil her
with open('Fil.txt') as outfile: # Må ha med 'w'
    for e in y:
        outfile.write(e) # e er tall, må være string
outfile.close()
```

## Quiz

Dette programmet skal skrive ut tallene

$\sin(0)$ ,  $\sin(0.1)$ ,  $\sin(0.2)$ , ...,  $\sin(10.0)$

på separate linjer i filen "Fil.txt". Finn feilene!

```
n = 100 # Skal være n = 101
x = np.linspace(0,10,n) # Vi har ikke importert numpy
y = np.sin(x) # Samme feil her
with open('Fil.txt') as outfile: # Må ha med 'w'
    for e in y:
        outfile.write(e) # e er tall, må være string
outfile.close() # Vi har glemt linjeskift
```

## Korrekt kode:

```
import numpy as np
n = 101
x = np.linspace(0,10,n)
y = np.sin(x)
with open('Fil.txt','w') as outfile:
    for e in y:
        outfile.write(f'{e}')      # outfile.write(str(e))
        outfile.write('\n')
outfile.close()
```



## Plotting (repetisjon)

1) Den store læreboka (Langtangen) nevner en rekke muligheter for grafplotting:

`matplotlib.pyplot`, `scitools.std`, `EasyViz`, `Mayavi`

Bare den første av disse benyttes i dette kurset.

2) Den anbefalte måten å bruke plottefunksjonene på er å skrive `import matplotlib.pyplot as plt` og så bruke `plt.plot(..)` osv.

3) Merk: når du bruker `plt.plot(x,y)` kan variablene `x` og `y` være enten arrayer eller lister.

## Plotte en enkelt kurve

Anta  $x$  og  $y$  er numeriske lister eller arrayer av samme lengde. Vi kan plotte dataene slik:

```
import matplotlib.pyplot as plt

plt.plot(x, y, 'ro')      # Røde punkter ('r-' for kurve)
plt.xlabel('x')          # Navn på x-aksen
plt.ylabel('y')          # Navn på y-aksen
plt.title('My plot')     # Overskrift
plt.axis([0,5,0,1])     # xmin,xmax,ymin,xmax
plt.show()
```

# Eksempel 1

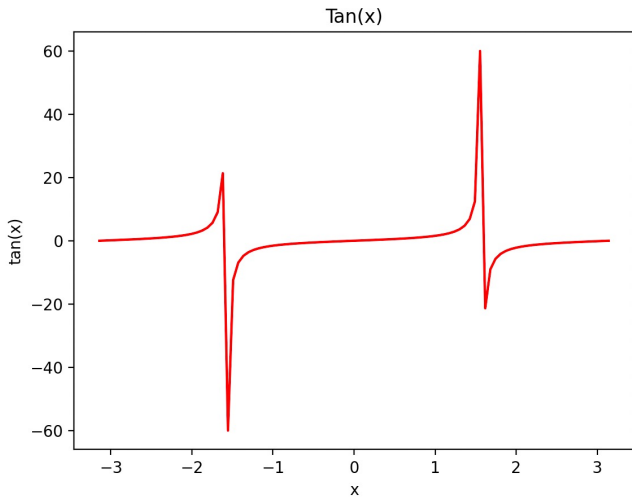
Tangentfunksjonen:

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

x = np.linspace(-3.14, 3.14, 100)
y = np.tan(x)

plt.plot(x, y, 'r-')
plt.xlabel('x')
plt.ylabel('tan(x)')
plt.title('Tan(x)')
plt.show()
```

# Resultat



## Eksempel 2

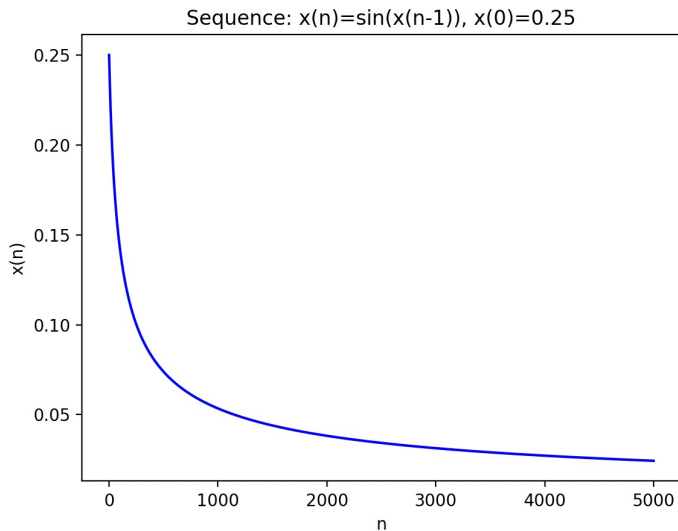
Følgen  $0.25, \sin(0.25), \sin(\sin(0.25)), \dots$  :

```
import matplotlib.pyplot as plt
import math

N = 5000
x = [0]*N
x[0] = 0.25
for i in range(1,N):
    x[i] = math.sin(x[i-1])

plt.plot(range(N), x, 'b-')
plt.xlabel('n')
plt.ylabel('x(n)')
plt.title('Sequence: x(n)=sin(x(n-1)), x(0)=0.25')
plt.show()
```

# Resultat



## Plotte flere kurver oppå hverandre

Anta at  $x_1$  og  $y_1$  er numeriske lister eller arrayer av samme lengde, og tilsvarende for  $x_2$  og  $y_2$ :

```
import matplotlib.pyplot as plt

plt.plot(x1, y1, 'r-')
plt.plot(x2, y2, 'b-')

plt.legend(['y1', 'y2'])

plt.show()
```

## Eksempel 1: Polynomier

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

def p(x,n):
    return x**n

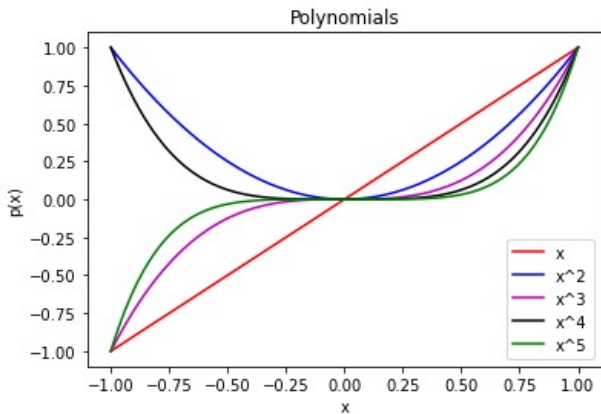
col = ['r-', 'b-', 'm-', 'k-', 'g-']
x = np.linspace(-1, 1, 100)

for n in range(1,6):
    plt.plot(x, p(x,n), col[n-1])

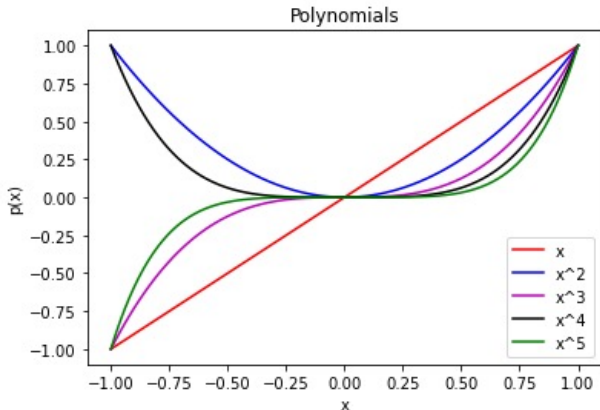
plt.xlabel('x')
plt.ylabel('p(x)')
plt.legend(['x', 'x^2', 'x^3', 'x^4', 'x^5'])
plt.title('Polynomials')
plt.show()
```



# Resultat



# Resultat



```
col = ['r-', 'b-', 'm-', 'k-', 'g-']
for n in range(1,6):
    plt.plot(x, p(x,n), col[n-1])
plt.legend(['x', 'x^2', 'x^3', 'x^4', 'x^5'])
```

## Avansert eksempel: Julia-mengde

Vi starter med å definere en funksjon  $f(x,y)$ :

```
def f(x,y):  
    k = 0  
    while abs(x) + abs(y) < 100 and k < 100:  
        x = x**2 - y**2 - 0.75  
        y = 2 * x * y  
        k = k + 1  
    if k < 100:  
        return 1  
    else:  
        return 0
```

## Avansert eksempel: Julia-mengde

Vi starter med å definere en funksjon  $f(x,y)$ :

```
def f(x,y):  
    k = 0  
    while abs(x) + abs(y) < 100 and k < 100:  
        x = x**2 - y**2 - 0.75  
        y = 2 * x * y  
        k = k + 1  
    if k < 100:  
        return 1  
    else:  
        return 0
```

Løkke som oppdaterer (x,y)

## Avansert eksempel: Julia-mengde

Vi starter med å definere en funksjon  $f(x,y)$ :

```
def f(x,y):  
    k = 0  
    while abs(x) + abs(y) < 100 and k < 100:  
        x = x**2 - y**2 - 0.75  
        y = 2 * x * y  
        k = k + 1  
    if k < 100:  
        return 1  
    else:  
        return 0
```

Løkken terminerer dersom  $|x|+|y|$  blir for stor eller antall iterasjoner overstiger 100

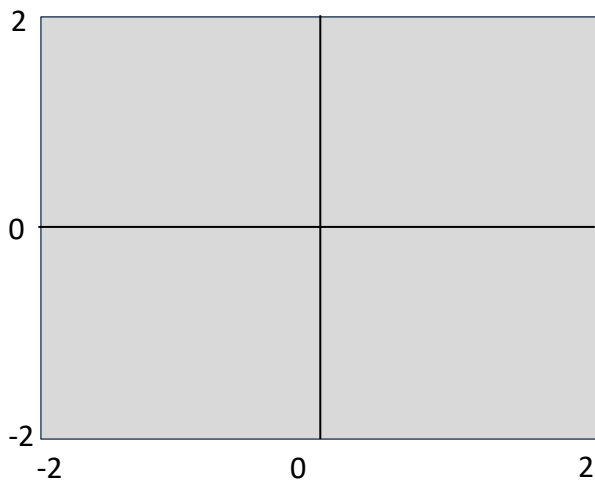
## Avansert eksempel: Julia-mengde

Vi starter med å definere en funksjon  $f(x,y)$ :

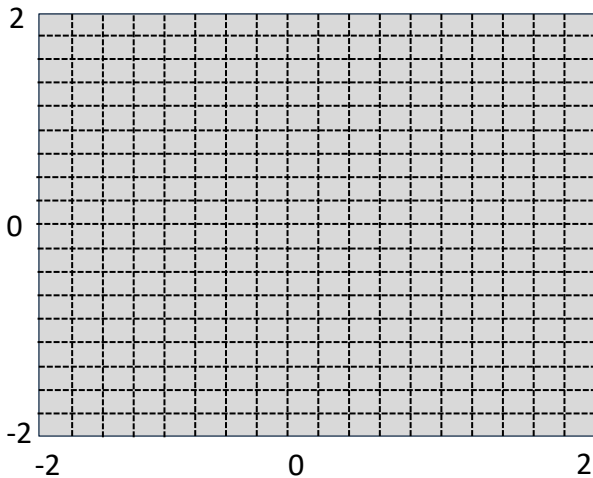
```
def f(x,y):  
    k = 0  
    while abs(x) + abs(y) < 100 and k < 100:  
        x = x**2 - y**2 - 0.75  
        y = 2 * x * y  
        k = k + 1  
    if k < 100:  
        return 1  
    else:  
        return 0
```

Returnerer alltid 0 eller 1

## Avansert eksempel: Julia-mengde



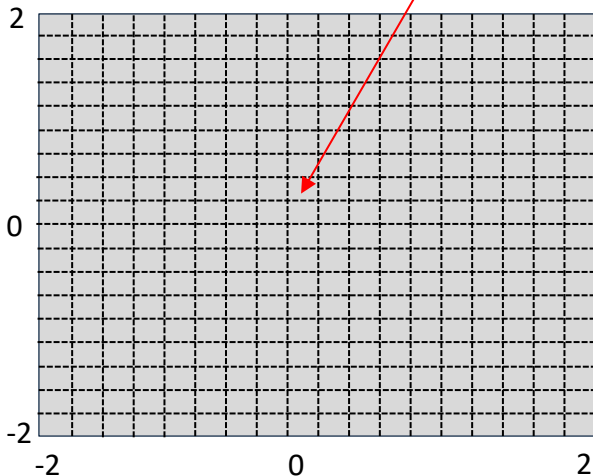
## Avansert eksempel: Julia-mengde





## Avansert eksempel: Julia-mengde

Fargelegg  $(x,y)$  i blått hvis  $f(x,y) = 1$   
og i hvitt hvis  $f(x,y) = 0$



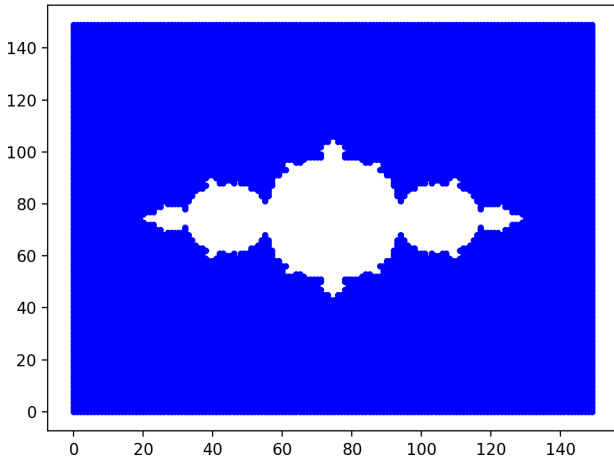
## Avansert eksempel: Julia-mengde

Komplett program for å fargelegge rutene i firkanten:

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

n = 150
x = np.linspace(-2, 2, n)
z1 = z2 = 0
for i in range(n):
    for j in range(n):
        z1, z2, k = x[i], x[j], 0
        while abs(z1)+abs(z2) < 100 and k < 100:
            z1,z2 = z1**2-z2**2-0.75, 2*z1*z2
            k = k+1
        if k < 100: # Svaret gikk mot uendelig
            plt.plot(i, j, 'b.')
plt.show()
```

# Resultat



## Multipanel plott

Anta at  $x_1$  og  $y_1$  er numeriske lister eller arrayer av samme lengde, og tilsvarende for  $x_2$  and  $y_2$ .

Kurver med titler:

```
import matplotlib.pyplot as plt

plt.subplot(1,2,1)
plt.plot(x1, y1, 'r-')
plt.title('Title for left panel')

plt.subplot(1,2,2)
plt.plot(x2, y2, 'b-')
plt.title('Title for right panel')

plt.show()
```

## Eksempel: polynomer

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

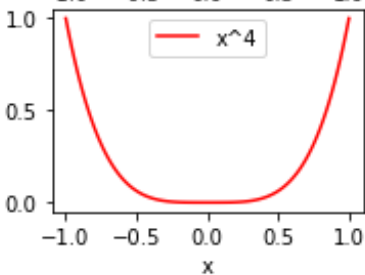
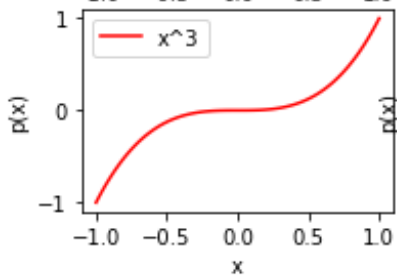
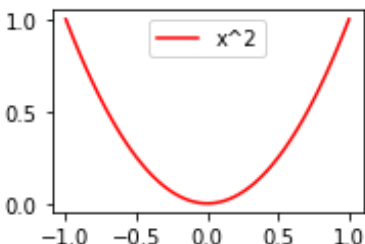
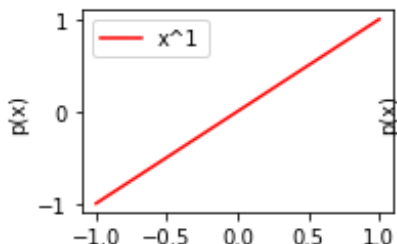
def p(x,n):
    return x**n

x = np.linspace(-1, 1, 100)

for n in range(1,5):
    plt.subplot(2,2,n)
    plt.plot(x, p(x,n), 'r-')
    plt.xlabel('x')
    plt.ylabel('p(x)')
    plt.legend([f'x^{n}'])

plt.show()
```

# Resultat



## Animering (repetisjon)

Mange mulige fremgangsmåter for å lage en "film" av en sekvens med plott. Tre mulige teknikker:

**Metode A:** Vis plottene mens programmet kjører

**Metode B:** Lagre plottene som nummererte bildefiler

**Metode C:** Bruke `FuncAnimation` i `matplotlib.animation`

Se forrige ukes forelesning for detaljer.

**Oppgave:** Lag en animasjon som fortløpende viser grafene til polynomene

$$x^1, x^2, x^3, \dots$$

for  $x \in [-2, 2]$ .



# Metode A: Vis film mens programmet kjører

Ide:

## Trinn 1:

Plott  $y = x$

Ta en kort pause



## Trinn 2:

Bytt ut kurven med  $y = x^2$

Ta en kort pause



## Trinn 3:

Bytt ut kurven med  $y = x^3$

Ta en kort pause



.... OSV....

# Metode A: Implementasjon

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

# Sett opp plott
plt.axis([-2, 2, -20, 20])      # xmin,xmax,ymin,ymax
x = np.linspace(-2, 2, 100)

# Plott y=x
lines = plt.plot(x, x)
plt.title("y = x")
plt.draw()
plt.pause(0.5)

# Plott  $y=x^2$ ,  $y=x^3$ , ...,  $y=x^{20}$ 
for k in range(2,21):
    lines[0].set_ydata(x**k)    # Bytt ut kurven, behold akser
    plt.title(f"y = x^{k}")
    plt.draw()                  # Tegn den nye kurven
    plt.pause(0.5)
```

# Hvis du bruker Spyder

Med standardinnstillingene i nyere versjoner av Spyder fungerer ikke eksemplet på forrige slide slik det skal.



For å fikse dette problemet, se innlegget under (det ligger under Beskjeder på nettsiden til kurset):

<https://www.uio.no/studier/emner/matnat/ifi/IN1900/h22/beskjeder/kjente-problemer-med-spyder.html>

# Metode B: Lag nummererte bildefiler

Ide:

## Trinn 1:

Plott  $y = x$

Lagre til fil

→ Figure001.png

## Trinn 2:

Bytt ut kurven med  $y = x^2$

Lagre til fil

→ Figure002.png

## Trinn 3:

Bytt ut kurven med  $y = x^3$

Lagre til fil

→ Figure003.png

.... OSV....

## Metode B: Implementasjon

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

# Sett opp plott
plt.axis([-2, 2, -20, 20])
x = np.linspace(-2, 2, 100)

# Plott y=x
lines = plt.plot(x, x)
plt.title("y = x")
plt.savefig("Figure01.png")

# Plott  $y=x^2$ ,  $y=x^3$ , ...,  $y=x^{20}$ 
for k in range(2,21):
    lines[0].set_ydata(x**k)
    plt.title(f"y = x^{k}")
    plt.draw()
    plt.savefig(f"Figure{k:02d}.png")
```

# Metode B: Fra bildefiler til film

Det er mulig å kombinere de nummererte bildefilene til en film ved å bruke ekstern programvare som *ImageMagick*.

Ikke nødvendig for dette kurset, men for de som har lyst følger her en kort bruksanvisning for mac (nb: tar litt tid og krever plass):

## TRINN 1: Lag figurfiler (fig1.png, fig2.png, ...) i Python:

```
> python animate.py
```

(her er animate.py ditt program)

## TRINN 2: Installer Homebrew:

```
> /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

## TRINN 3: Installer Imagemagick:

```
> brew install imagemagick
```

## TRINN 4: Konverter bildefiler til film:

```
> convert -delay 30 tmp_*.png movie.gif
```

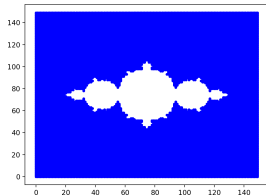
## TRINN 5: Vis filmen:

Åpne en nettleser og åpne filen movie.gif fra denne.

# Eksempel: Julia-mengden

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

n = 150
x = np.linspace(-2, 2, n)
z1 = z2 = 0
for i in range(n):
    for j in range(n):
        z1, z2, k = x[i], x[j], 0
        while abs(z1)+abs(z2) < 100 and k < 100:
            z1, z2 = z1**2-z2**2-0.75, 2*z1*z2
            k = k+1
        if k < 100: # Svaret gikk mot uendelig
            plt.plot(i, j, 'b.')
```



```

import matplotlib.pyplot as plt
import numpy as np

def julia(xmin,xmax,ymin,ymax):
    n = 150
    x = np.linspace(xmin, xmax, n)
    y = np.linspace(ymin, ymax, n)
    z1 = z2 = 0
    for i in range(n):
        for j in range(n):
            z1, z2, k = x[i], y[j], 0
            while abs(z1)+abs(z2) < 50 and k < 200:
                z1,z2 = z1**2-z2**2-0.75, 2*z1*z2
                k = k+1
            if k < 200:
                plt.plot(i, j, 'b.')
plt.axis([0,n,0,n])

a = np.array([-2,2,-2,2])
b = np.array([-0.45, -0.39, 0.37, 0.38])

u = np.linspace(0,1,200)
u = np.sqrt(u)
for k in range(len(u)):
    v = u[k]*b + (1-u[k])*a
    julia(v[0],v[1],v[2],v[3])
plt.savefig(f"Julia{k:03d}.png")
plt.clf()

```



## Exercise 5.29: Judge a plot

Assume you have the following program for plotting a parabola:

```
import numpy as np
x = np.linspace(0, 2, 20)
y = x*(2 - x)
import matplotlib.pyplot as plt
plt.plot(x, y)
plt.show()
```

Then you switch to the function  $\cos(18\pi x)$  by altering the computation of  $y$  to  $y = \cos(18\pi x)$ . Judge the resulting plot. Is it correct? Display the  $\cos(18\pi x)$  function with 1000 points in the same plot.

Filename: judge\_plot.

## Exercise 5.39: Animate the evolution of Taylor polynomials

A general series approximation (to a function) can be written as

$$S(x; M, N) = \sum_{k=M}^N f_k(x).$$

For example, the Taylor polynomial of degree  $N$  for  $e^x$  equals  $S(x; 0, N)$  with  $f_k(x) = x^k / k!$ . The purpose of the exercise is to make a movie of how  $S(x; M, N)$  develops and improves as an approximation as we add terms in the sum. That is, the frames in the movie correspond to plots of  $S(x; M, M)$ ,  $S(x; M, M + 1)$ ,  $S(x; M, M + 2)$ ,  $\dots$ ,  $S(x; M, N)$ .

## Exercise 5.39: Animate the evolution of Taylor polynomials

A general series approximation (to a function) can be written as

$$S(x; M, N) = \sum_{k=M}^N f_k(x).$$

For example, the Taylor polynomial of degree  $N$  for  $e^x$  equals  $S(x; 0, N)$  with  $f_k(x) = x^k/k!$ . The purpose of the exercise is to make a movie of how  $S(x; M, N)$  develops and improves as an approximation as we add terms in the sum. That is, the frames in the movie correspond to plots of  $S(x; M, M)$ ,  $S(x; M, M + 1)$ ,  $S(x; M, M + 2)$ ,  $\dots$ ,  $S(x; M, N)$ .

Eksempel:

Taylorpolynom for  $e^x$ :

$$S(x; M, N) = \frac{x^M}{M!} + \dots + \frac{x^N}{N!}$$

$$f_k(x) = x^k/k!$$

I Python: funksjon av to variabler:

```
def f(x,k):  
    return x**k/math.factorial(k)
```

a) Make a function

```
animate_series(fk, M, N, xmin, xmax, ymin, ymax, n, exact)
```

for creating such animations. The argument `fk` holds a Python function implementing the term  $f_k(x)$  in the sum, `M` and `N` are the summation limits, the next arguments are the minimum and maximum  $x$  and  $y$  values in the plot, `n` is the number of  $x$  points in the curves to be plotted, and `exact` holds the function that  $S(x)$  aims at approximating.

*Hint* Here is some more information on how to write the `animate_series` function. The function must accumulate the  $f_k(x)$  terms in a variable  $s$ , and for each  $k$  value,  $s$  is plotted against  $x$  together with a curve reflecting the exact function. Each plot must be saved in a file, say with names `tmp_0000.png`, `tmp_0001.png`, and so on (these filenames can be generated by `tmp_%04d.png`, using an appropriate counter). Use the `movie` function to combine all the plot files into a movie in a desired movie format.

In the beginning of the `animate_series` function, it is necessary to remove all old plot files of the form `tmp_*.png`. This can be done by the `glob` module and the `os.remove` function as exemplified in Sect. 5.3.4.

- b) Call the `animate_series` function for the Taylor series for  $\sin x$ , where  $f_k(x) = (-1)^k x^{2k+1}/(2k+1)!$ , and  $x \in [0, 13\pi]$ ,  $M = 0$ ,  $N = 40$ ,  $y \in [-2, 2]$ .
- c) Call the `animate_series` function for the Taylor series for  $e^{-x}$ , where  $f_k(x) = (-x)^k/k!$ , and  $x \in [0, 15]$ ,  $M = 0$ ,  $N = 30$ ,  $y \in [-0.5, 1.4]$ .

Filename: `animate_Taylor_series`.