Arrayer, plotting, animering og differenslikninger

Ole Christian Lingjærde, Dept of Informatics, UiO

27. september 2021

Ukens agenda

- Kort repetisjon av arrayer, plotting og animering
- Plenumsøvelser 5.29, 5.39, A.1 og A.4 fra Langtangens bok
- Differenslikninger

Mange valg å ta når man programmerer

Python byr på mange (store og små) valg:

- Ulike versjoner av Python
- Bruke lister eller arrayer?
- Bruke `matplotlib` eller `scitools` til å plotte?
- Skrive `np.linspace(..)` og `plt.plot(..)` eller bare `linspace(..)` og `plot(..)`?
- Bruke `from numpy import *` etc?
- Initiere lister med `a = [0]*n` eller bruke `a.append(..)`?
- ...osv osv

Tupler, lister eller arrayer?

Tupler, lister og arrayer kan alle brukes til å lagre mange verdier.

Tupler:

- Relativt fleksible (kan blande datatyper)
- Matematiske operasjoner gjøres ett element av gangen

Lister:

- Svært fleksible (kan blande datatyper, kan modifiseres)
- Matematiske operasjoner gjøres ett element av gangen

Arrayer:

- Mindre fleksible (bare én datatype)
- Vektoriserte matematiske operasjoner tilgjengelig
- Raskere å skrive, raskere å kjøre

Valget mellom liste og array

- **Arrayer**: spesielt nyttige for å håndtere numeriske vektorer og matriser i situasjoner hvor det gir mening å utføre matematiske operasjoner samtidig på alle elementene
- **Lister**: alltid en opsjon (hvis ikke det er krav om arrayer)
- **Fra array til liste**: `l = list(a)`
- **Fra liste til array**: `a = np.array(l)`

NB: konvertering mellom array og liste er ikke effektivt når det er veldig lange lister/arrayer.

Sammenlikning av lister og arrayer

Liste	Array
`x = [1,2,3,4]`	`x = np.array([1,2,3,4])`
`x = [0]*n`	`x = np.zeros(n)`
`x = [1]*n`	`x = np.ones(n)`
`x = range(n)`	`x = np.arange(n)`
`xnew = x`	`xnew = x`
`xnew = x[:]`	`xnew = x.copy()`
`xnew = x+x`	`xnew = np.append(x,x)`
`h = float(b-a)/(n-1)` `x = [a+i*h for i in range(n)]`	`x = np.linspace(a,b,n)`
`for elem in x:` `  print(elem)`	`for elem in x:` `  print(elem)`

`xnew = [0]*len(x)`
`for i in range(len(x)):`
`  xnew[i] = math.sin(x[i])`
	`xnew = np.sin(x)`

`xnew = [0]*len(x)`
`for i in range(len(x)):`
`  xnew[i] = x[i] + 2*x[i]**2`
	`xnew = x + 2*x**2`

Hvordan bruke numpy-funksjoner

- **Generell 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 (repetisjon)

Viktig å forstå begrepet *vektorisert beregning*. Eksempel på 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
```

Tilsvarende 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
```

Hvor mye raskere er vektorisert kode?

Sammenlikning av CPU-tid:

```
import time
N = 10**7

t0 = time.clock()
f_list(N)
t1 = time.clock() - t0
print('Nonvectorized: %4.2f seconds' % t1)

t0 = time.clock()
f_array(N)
t1 = time.clock() - t0
print('Vectorized: %4.2f seconds' % t1)

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

Den vektoriserte (numpy) løsningen er 23 ganger raskere!

Plotting (repetisjon)

- Boka nevner en rekke muligheter for grafplotting: `matplotlib.pyplot`, `scitools.std`, EasyViz, Mayavi. Bare den første av disse er nødvendig i dette kurset.
- Den anbefalte måten å bruke plottefunksjonene på er å skrive `import matplotlib.pyplot as plt` og så bruke `plt.plot(..)` etc (se side 243 i læreboka).
- 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.

Bare plotte kurve:

```
import matplotlib.pyplot as plt
plt.plot(x, y)                  # Create plot
plt.savefig('Figure1.pdf')      # Save plot as pdf
plt.show()                      # Show plot on screen
```

Plotte kurve med dekorering:

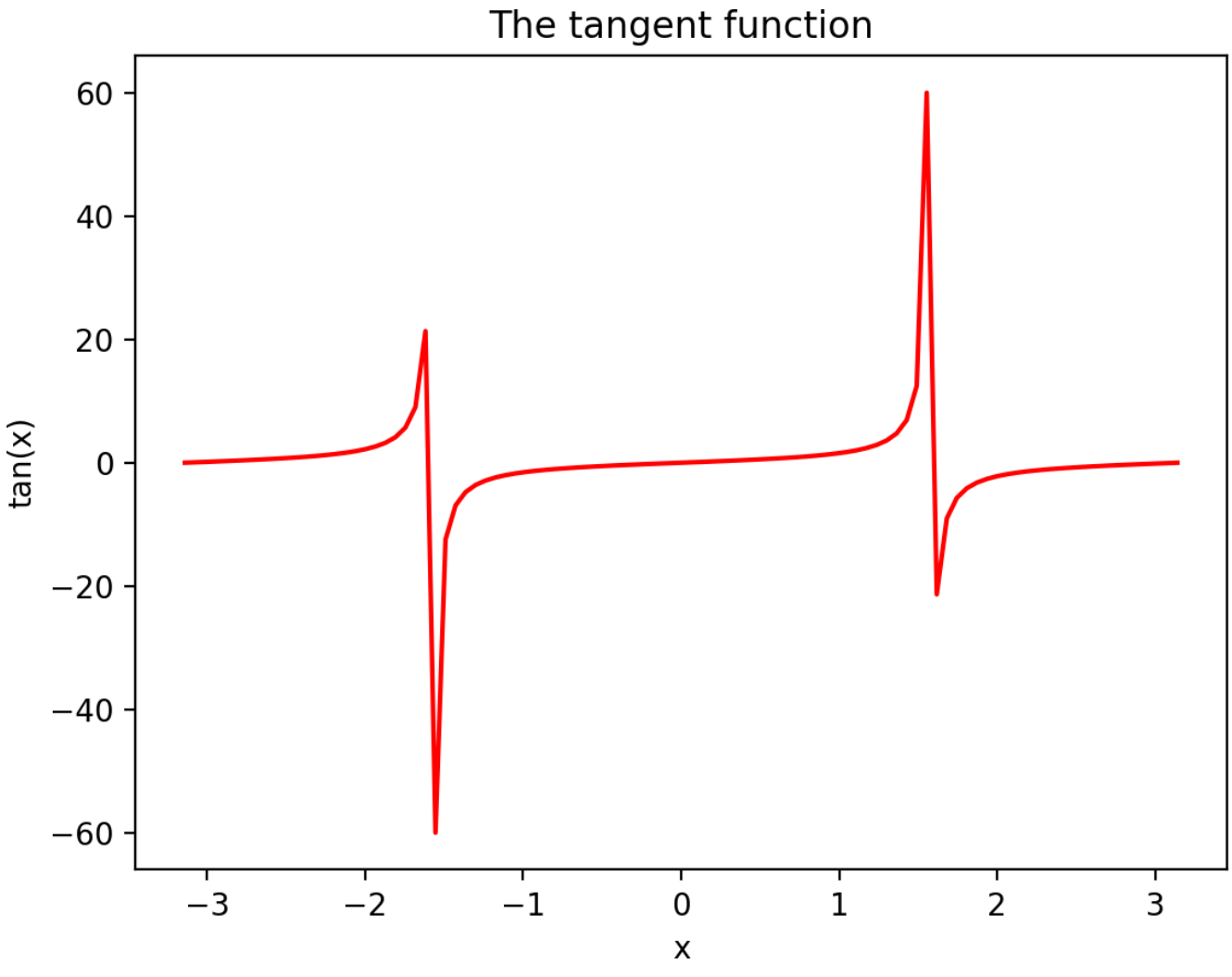
```
import matplotlib.pyplot as plt
plt.plot(x, y, 'r-') # Red line (use 'ro' for red circle)
plt.xlabel('x')       #Label on x-axis
plt.ylabel('y')       #Label on y-axis
plt.title('My plot') # Title on top of plot
plt.axis([0,5,0,1])  # Range on x-axis [0,5] and y-axis [0,1]
plt.show()
```

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-') #Red line (use 'ro' for red circle)
plt.xlabel('x')       #Label on x-axis
plt.ylabel('tan(x)') # Label on y-axis
plt.title('The tangent function')
plt.show()
```

Resultat

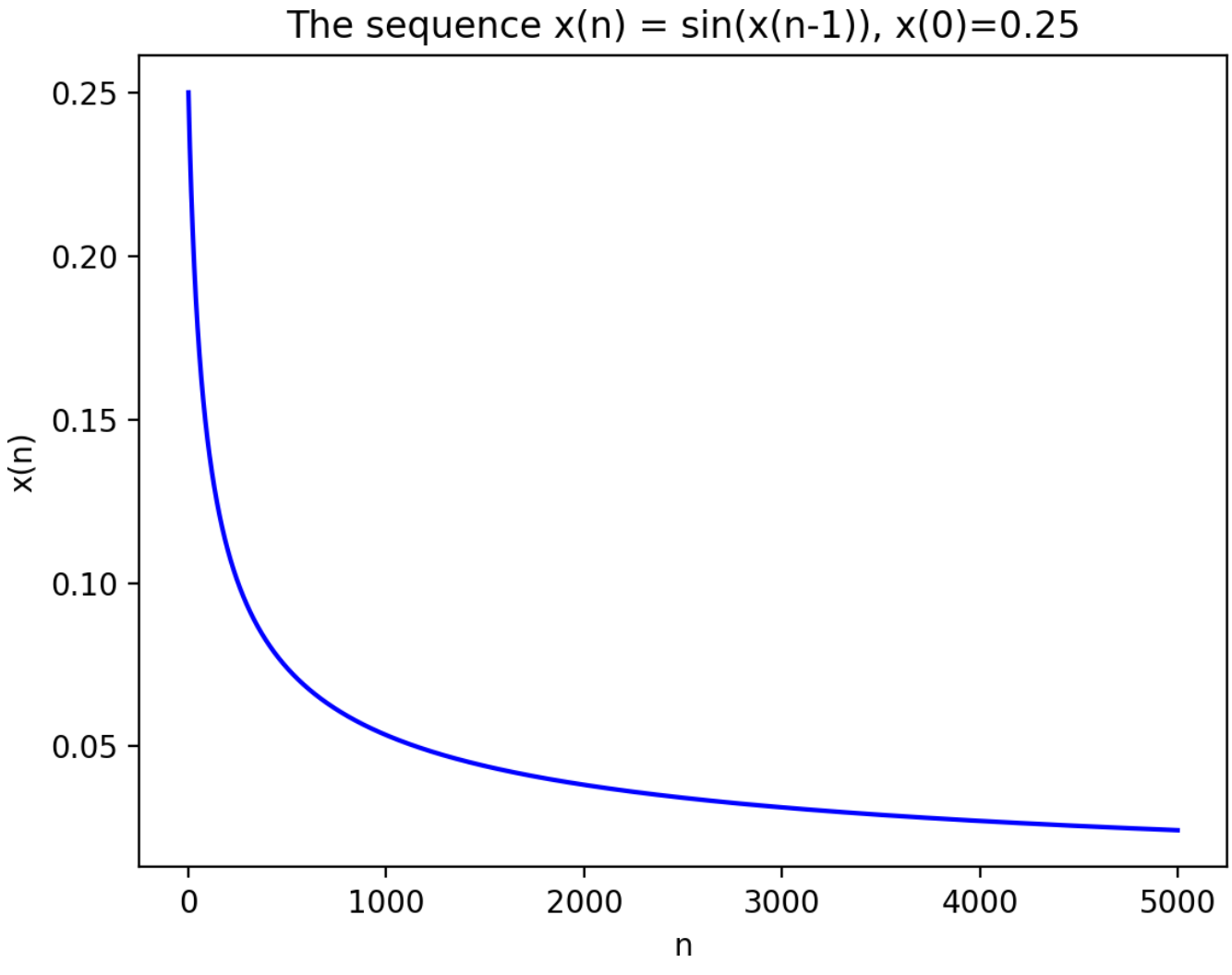


Eksempel 2

Følgen 0.25, sin(0.25), sin(sin(0.25)),...:

```
import matplotlib.pyplot as plt
import math
N = 5000
y = [0]*N
y[0] = 0.25
for i in range(1,N):
    y[i] = math.sin(y[i-1])
plt.plot(range(N), y, 'b-')  # Blue line
plt.xlabel('n')               #Label on x-axis
plt.ylabel('x(n)')          # Label on y-axis
plt.title('The sequence x(n) = sin(x(n-1)), x(0)=0.25')
plt.show()
```

Resultat


The sequence x(n) = sin(x(n-1)), x(0)=0.25
0.25
0.20
0.15
0.10
0.05
x(n)
0
1000
2000
3000
4000
5000
n

Plotte flere kurver oppå hverandre

Anta at `x1` og `y1` er numeriske lister eller arrayer av samme lengde, og tilsvarende for `x2` og `y2`.

Bare plotte kurvene:

```
import matplotlib.pyplot as plt
plt.plot(x1, y1, 'r-')
plt.plot(x2, y2, 'b-')
plt.show()
```

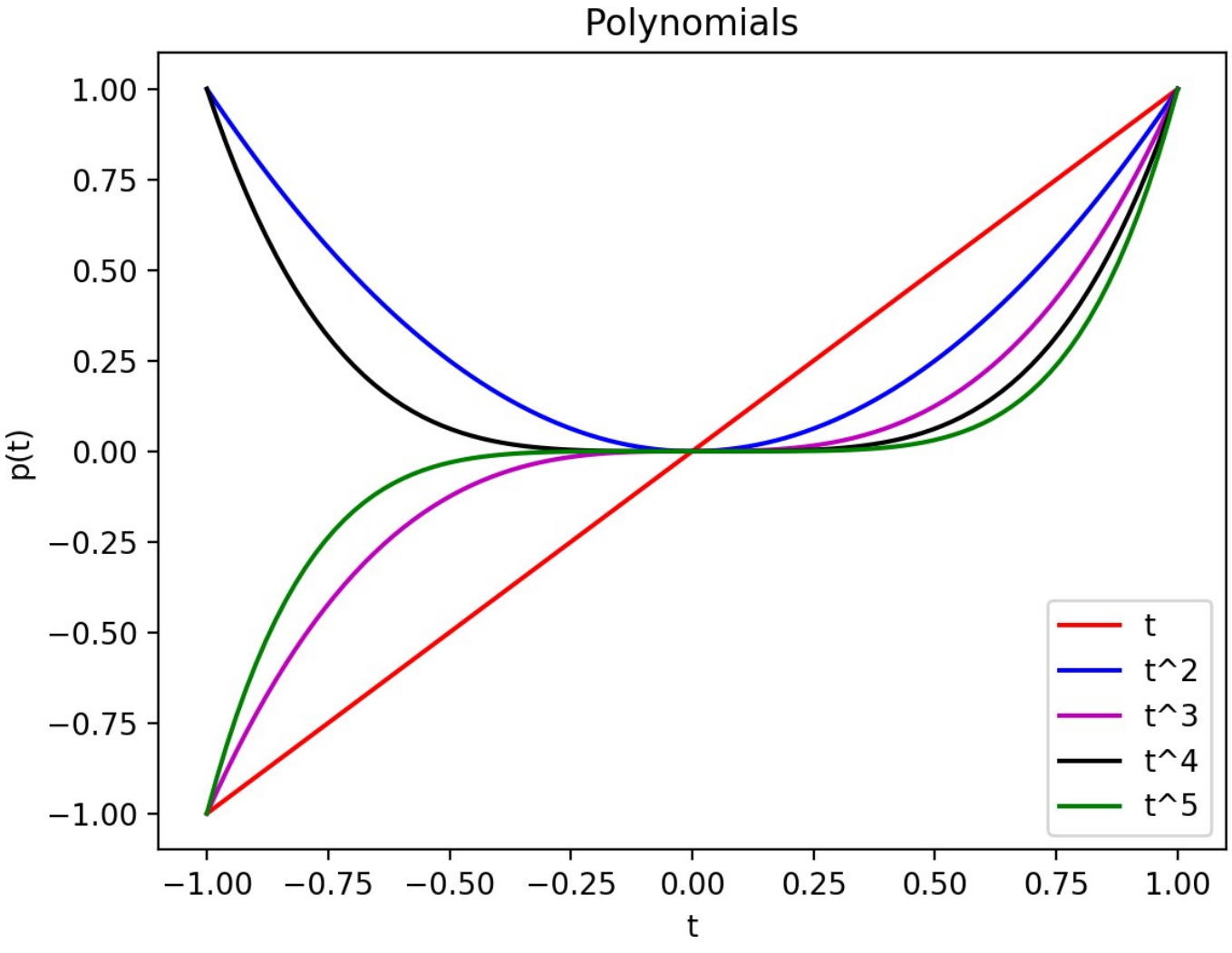
Plotte kurvene med dekorering:

```
import matplotlib.pyplot as plt
plt.plot(x1, y1, 'r-')
plt.plot(x2, y2, 'b-')
plt.legend(['y1', 'y2'])
plt.xlabel('x')
plt.ylabel('y')
plt.title('My multiplot')
plt.axis([0,7,0,7])
plt.show()
```

Eksempel 1: Polynomer

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

def p(t,k):
  return t**(k+1)

col = ['r-', 'b-', 'm-', 'k-', 'g-']

t = np.linspace(-1, 1, 100)
for k in range(5):
  plt.plot(t, p(t,k), col[k])  # Plot t^1, t^2, ..., t^5

plt.xlabel('t')
plt.ylabel('p(t)')
plt.legend(['t', 't^2', 't^3', 't^4', 't^5'])
plt.title('Polynomials')
plt.show()
```

Resultat

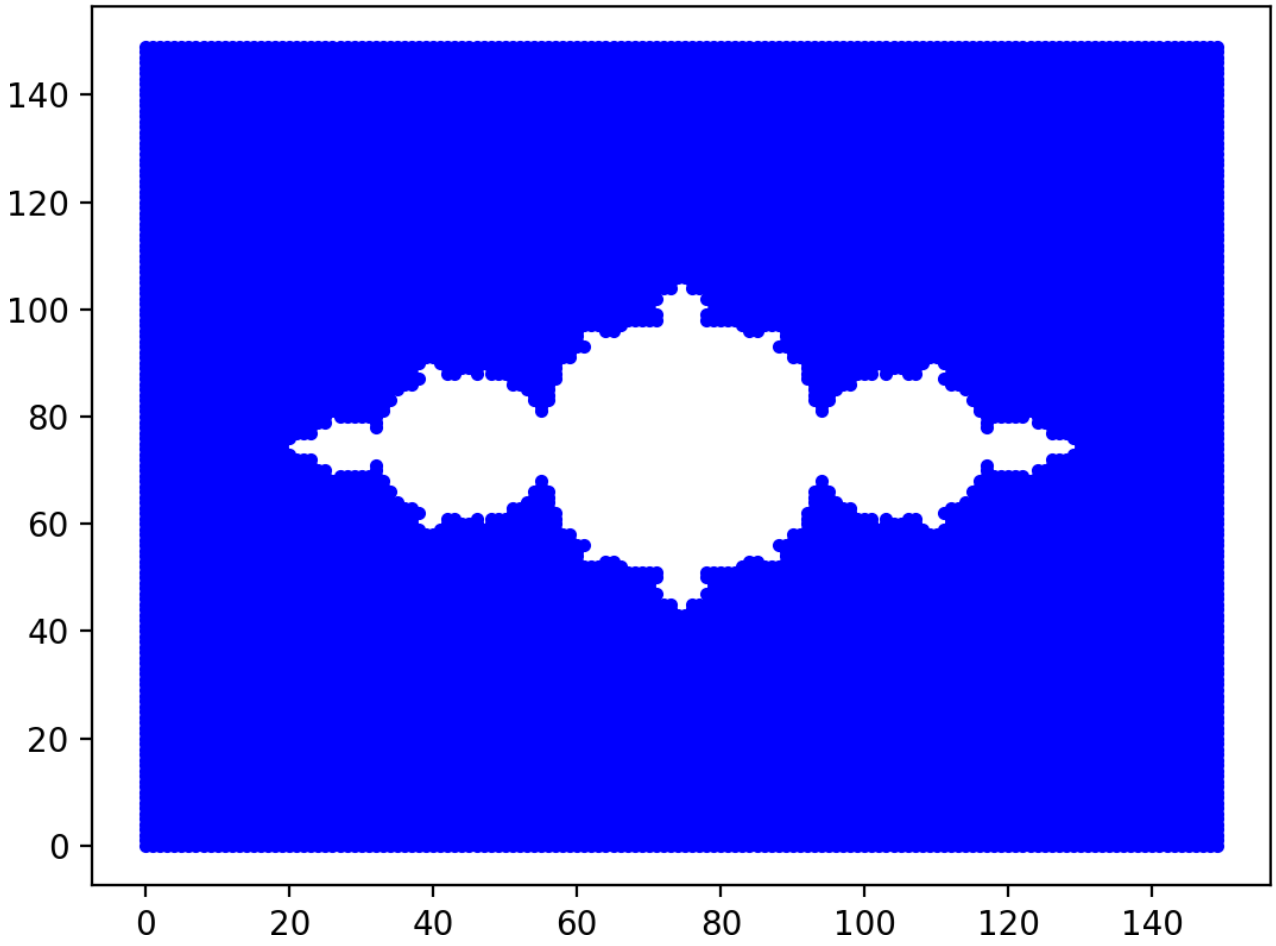

Polynomials
p(t)
t
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
t
t^2
t^3
t^4
t^5

Eksempel 2: Julia set

Vi kan plotte mer enn grafer i Python! Kan du se hva som plottes her? Hint: z representerer et komplekst tall.

```
import matplotlib.pyplot as plt
import numpy as np
n = 150
x = np.linspace(-2, 2, n); z = [0, 0]
for i in range(n):
    for j in range(n):
        z[0] = x[i]; z[1] = x[j]; k = 0
        while abs(z[0])+abs(z[1]) < 100 and k < 100:
            z = [z[0]**2-z[1]**2-0.75, 2*z[0]*z[1]]
            k = k+1
        if k < 100:
            plt.plot(i, j, 'b.')
plt.show()
```

Resultat: en fraktal



Multipanel plott

Anta at `x1` og `y1` er numeriske lister eller arrayer av samme lengde, og tilsvarende for `x2` and `y2`.

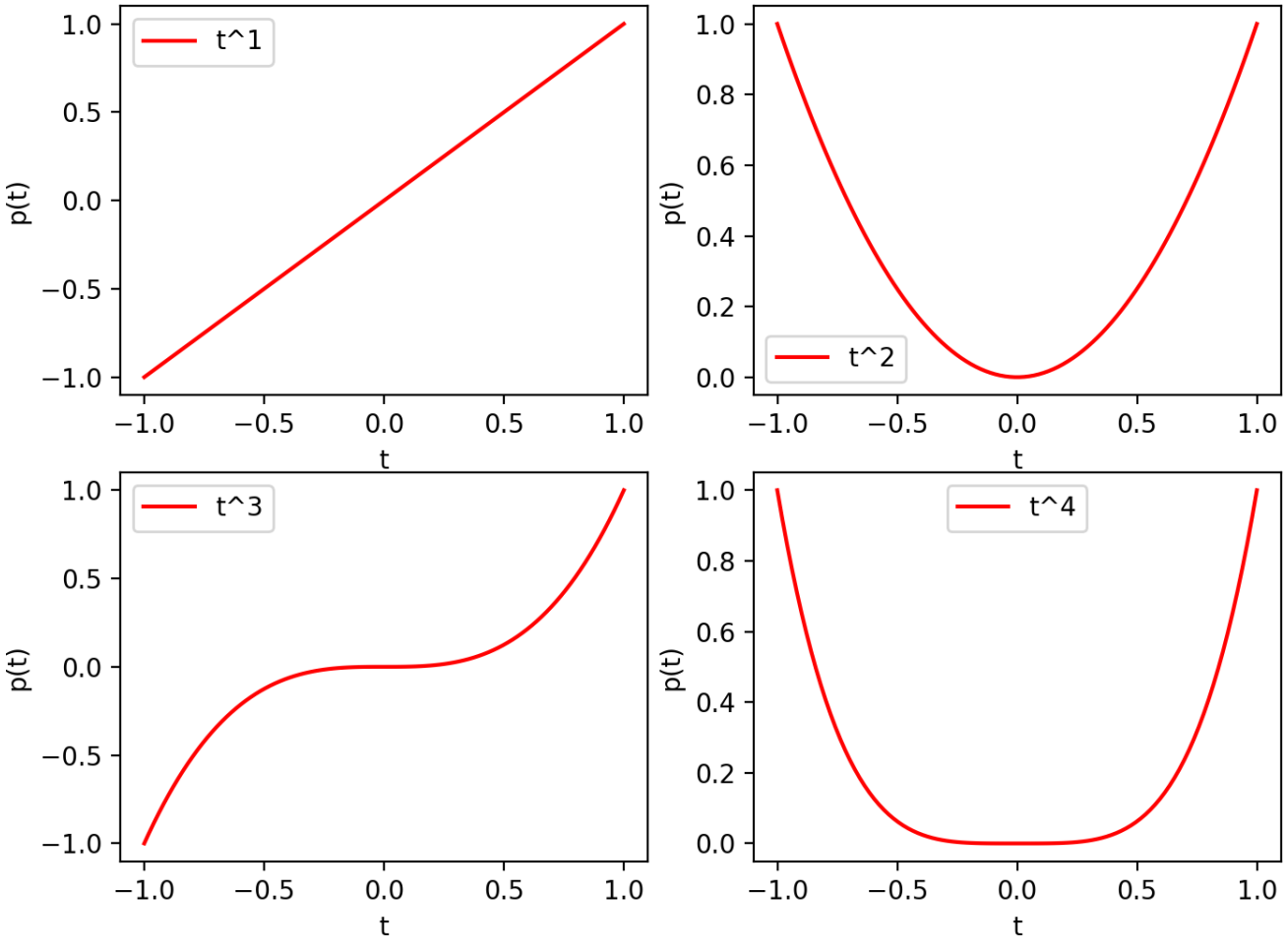
Kurver med titler:

```
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(t,k):
  return t**k

t = np.linspace(-1, 1, 100)
for k in range(1,5):
  plt.subplot(2,2,k)
  plt.plot(t, p(t,k), 'r-')
  plt.xlabel('t')
  plt.ylabel('p(t)')
  plt.legend(['t^%d' % k])
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.

Eksempel

Oppgave: Lag en animasjon som fortløpende viser grafene til polynomene

$$x^1,\ x^2\ \cdots x^{20}$$

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

Metode A: vis film under kjøring

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

# Sett opp plott
plt.axis([-2, 2, -20, 20]) # xmin, xmax, ymin, ymax

# Definer x-verdier
x = np.linspace(-2, 2, 100) # x-verdier til plotting

# 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) # Oppdater plottedata
    plt.title(f"y = x^{k}")
    plt.draw()                # Oppdater plottet
    plt.pause(0.5)              # Kort pause mellom hvert plott
```

Metode B: lag bildefiler og konvertér til film

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

# Sett opp plott
plt.axis([-2, 2, -20, 20]) # xmin, xmax, ymin, ymax

# Definer x-verdier
x = np.linspace(-2, 2, 100) # x-verdier til plotting

# Plott y=x
lines = plt.plot(x, x)
plt.savefig("figures/tmp_0000.png")

# Plott y=x^2, y=x^3,...., y=x^20
for k in range(2,21):
    lines[0].set_ydata(x**k)
    plt.draw()
    plt.savefig(f"figures/tmp_{k-1:04d}.png")

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

NB: Du trenger ekstern programvare som *ImageMagick* (brukt over) eller *ffmpeg* for å kombinere bildefilene til en film.

Følger og differenslikninger

Følge: $x_0, x_1, x_2, \ldots$

Eksempler:

$$\begin{aligned} x_n &= \frac{n(n+1)}{2}, & n &= 1, 2, \ldots \\ y_n &= n^2 - n, & n &= 1, 2, \ldots \\ z_n &= \log(1 + \tfrac{1}{n}), & n &= 1, 2, \ldots \end{aligned}$$

Differenslikning: formel som knytter sammen x_n med en eller flere tidligere ledd $x_{n-1}, x_{n-2}, \ldots$.

Eksempler:

$$\begin{aligned} x_n &= x_{n-1} + x_{n-2}, & n &= 2, 4, 5, \ldots \\ y_n &= 2y_{n-1} + 2^{-n}, & n &= 1, 2, 3, \ldots \\ z_n &= a_1 z_{n-1} + a_2 z_{n-2} + \cdots + a_r z_{n-r} \end{aligned}$$

Hvorfor er differenslikninger viktige?

(A) De opptrer naturlig i mange sammenhenger:

- Lån(t) = lån(t-1) + renter - avdrag
- Antall ulv(t) = antall ulv(t-1) + fødte - døde
- Antall bakterier(t) = 2*antall bakterier(t-1)
- Posisjon(t) = Posisjon(t-1) + forflytning

Eksempel:

Vi setter inn x_0 kroner i banken. Hvis det er p prosent rente pr år kan verdien etter n år uttrykkes som

$$x_n = \left(1 + \frac{p}{100}\right) x_{n-1}$$

Hvorfor er differenslikninger viktige?

(B) De kan spare oss beregningstid:

Eksempel 1:

Vi kan finne de første 1000 leddene av $x_n = n!$ raskt ved å sette $x_1 = 1$ og benytte formelen

$$x_n = nx_{n-1}, \quad n = 2, 3, \ldots$$

Eksempel 2:

Vi kan finne de første 1000 leddene av $x_n = n(n + 1)(2n + 1)/6$ raskere ved å sette $x_1 = 1$ og benytte formelen

$$x_n = x_{n-1} + n^2, \quad n = 2, 3, \ldots$$

Hvorfor er differenslikninger viktige?

(C) Ofte eneste alternativ:

Eksempel:

$$x_n = x_{n-1}(a - x_{n-1})(b - x_{n-2})$$

Her er det vanskelig å finne et eksplisitt uttrykk for det $n$te leddet.

Dvs vi klarer ikke å finne en (nyttig) formel for å regne ut x_n uten å referere til tidligere verdier i følgen.

Plenumsoppgaver

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`.

Plenumsoppgaver

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 $S(x; M, M)$, $S(x; M, M+1)$, $S(x; M, M+2)$, ..., $S(x; M, N)$.

Plenumsoppgaver

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 $S(x; M, M)$, $S(x; M, M+1)$, $S(x; M, M+2)$, ..., $S(x; M, N)$.

Eksempel:

Taylorpolynom for e^x:

$$S(x; M, N) = \frac{x^M}{M!} + \cdots + \frac{x^N}{N!} \qquad f_k(x) = x^k/k!$$

I Python: funksjon av to variabler:

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

Plenumsoppgaver

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.

Plenumsoppgaver

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`.