

Interaktive Elementer med Jupyter Widgets

I dette kurset har vi gitt en veldig kort innføring i Jupyter notebook, et format som lar oss integrere kode med andre elementer som tekst, matematikk og figurer.

Det finnes mye funksjonalitet i Jupyter som vi *ikke* har nevnt. Ettersom at Jupyter er implementert igjennom nettleseren finnes det en lang rekke muligheter for ekstra funksjonalitet, og det finnes også tilleggspakker man kan laste ned for å utvide Jupyter enda mer. For eksempel finnes det en tilleggspakke der man kan lage og vise frem slides som en presentasjon i Jupyter (Les mer her: <https://damianavila.github.io/RISE/>) (<https://damianavila.github.io/RISE/>).

Vi har aktivt valgt å ikke vise frem eller nevne for mange av disse mulighetene, fordi de kanskje kan være litt kompliserte for nybegynnere. Derimot har vi blitt spurt om å vise hvordan man kan lage såkalte *widgets* i Jupyter, så det skriver vi litt om her.

Den grunnleggende `interact`

Den letteste måten å lage en interaktiv widget på er å bruke funksjonen `interact`, denne lar deg interagere med en gitt funksjon som du har definert. La oss se på et eksempel.

Vi lager en funksjon `f(x)`, som tar et tall `x` inn, og returnerer kvadratet av funksjonen. Vi kan definere funksjonen på "vanlig" måte, altså som vi ellers ville definert en funksjon for å jobbe med den.

In [1]:

```
1 def f(x):  
2     return x**2
```

Når vi har definert en slik funksjon, så kan vi bruke `interact` for å lage en interaktiv widget hvor vi kan utforske funksjonen. Først må vi importere `interact`, og denne ligger i pakka `ipywidgets`

In [2]:

```
1 from ipywidgets import interact
```

For å bruke `interact` må vi sende selve funksjonen vi ønsker å interegere med som input. I tillegg gir vi startverdien til `x`. Så vi kaller på

```
interact(f, x=10);
```

Merk at vi sender selve funksjonen, altså bare `f`. Til slutt skriver vi et semicolon, dette er ikke strengt tatt nødvendig, det kan bare unngå litt ekstra utskrift i Jupyter av grunner vi ikke går inn på her.

Resultatet blir som følger:

In [3]:

```
1 interact(f, x=10);
```

x  10


100

Vi får en slider der vi kan endre verdien av variabelen `x`, verdien begynner på 10, ettersom at det var dette vi satt startverdien til. Merk at man også kan trykke på tallverdien og skrive inn et konkret tall, om man syns det er knotete å treffe nøyaktig med slideren.

Ettersom at vi satt startverdien til et heltall, så vil slideren automatisk gå i hele steg. Om vi istedet hadde valgt et desimaltall som startverdi, så hadde vi kunne gått til andre desimaltall:

In [4]:

```
1 interact(f, x=10.0);
```

x  10.00

100.0

Merk at grensene vi kan gå til med variabelen `x` er satt automatisk av `interact`. Dette gjør den så funksjonen skal være så enkel som mulig å bruke, men vi kan også sette grenser selv - dette ser vi nærmere på snart.

Interact og output

I widgeten vi lagde nå får vi rett og slett skrevet ut returverdien til funksjonen. Dette er vel og bra, men av og til ønsker man å ha mer kontroll over hva som skrives ut. Da kan vi istedet for å lage en funksjon som returnerer en gitt verdi, heller printer ut noe direkte.

For eksempel kunne vi definert funksjonen slik her istedet:

In [5]:

```
1 def g(x):
2     print(f"{x}^2 = {x**2}")
```

Nå vil `interact` gi følgende resultat:

In [6]:

```
1 interact(g, x=10);
```

x  10

$10^2 = 100$

På denne måten kan vi også lage widgets som skriver ut informasjon på flere linjer for eksempel.

Tekst-input

Det er ikke bare matematiske funksjoner vi kan interagere med gjennom `interact`. Ta for eksempel et av funksjonene vi lagde i DNA-opplegget vårt. Der sender vi inn en DNA-sekvens som tekst, og kan for eksempel komplementere denne. Funksjonen kunne skrives som:

In [7]:

```
1 def g(x):  
2     return x**2
```

In [8]:

```
1 interact(g, x=10);
```

x  10

100

In [9]:

```
1 def komplementer(DNA):  
2     komplement = ""  
3     for base in DNA:  
4         if base == 'T':  
5             komplement += 'A'  
6         elif base == 'A':  
7             komplement += 'T'  
8         elif base == 'C':  
9             komplement += 'G'  
10        elif base == 'G':  
11            komplement += 'C'  
12        else:  
13            komplement += '?'  
14  
15    print(f"Input tråd: {DNA}")  
16    print(f"Komplement: {komplement}")
```

Bruker vi `interact` på denne funksjonen får vi nå en tekstboks vi kan skrive input-strengen inn i:

In [10]:

```
1 interact(komplementer, DNA="ATGC");
```

DNA

Input tråd: ATGC

Komplement: TACG

Annen input

Widgets kan også ta boolsk input (True/False). Da får vi en liten boks man kan huke av:

Eller vi kan lage en liste, og da får vi en dropdown-boks hvor man kan velge ett av elementene.

Interact og plotting

Om vi ønsker å lage en widget der vi interagerer med et plot, er vi nødt til å lage en funksjon som produserer hele plottet for oss. Dette er ikke så avansert, vi må bare legge koden vi vanligvis bruker for å lage plottet på innsiden av en funksjon.

La oss se på et eksempel. La oss si vi ønsker å utforske hvordan formelen for en rett linje fungerer. Formelen for en slik linje er

$$y = ax + b,$$

der a og b kan justeres. La oss lage en widget, der vi kan oppdatere disse dynamisk:

In [11]:

```
1 from pylab import *
2
3 def tegn_rett_linje(a, b):
4     # Regn ut x- og y-verdier
5     x = linspace(-10, 10, 101)
6     y = a*x + b
7
8     # Plot kurve
9     plot(x, y, linewidth=2.0)
10
11     # Sett navn på akser og figur
12     xlabel("x")
13     ylabel("y")
14     title(f"y = {a}x + {b}")
15
16     # Lås aksene
17     xlim(-10, 10)
18     ylim(-10, 10)
19
20     # Tegn aksekors
21     grid()
22     axvline(0, color='black')
23     axhline(0, color='black')
24     show()
```

Her har vi definert funksjonen `tegn_rett_linje`. Denne tar de to parametrene `a` og `b` inn. Deretter plottes vi $y = ax + b$ for $x \in [-10, 10]$. Funksjonen ser litt voldsom ut, men det meste er bare å pynte på plottet.

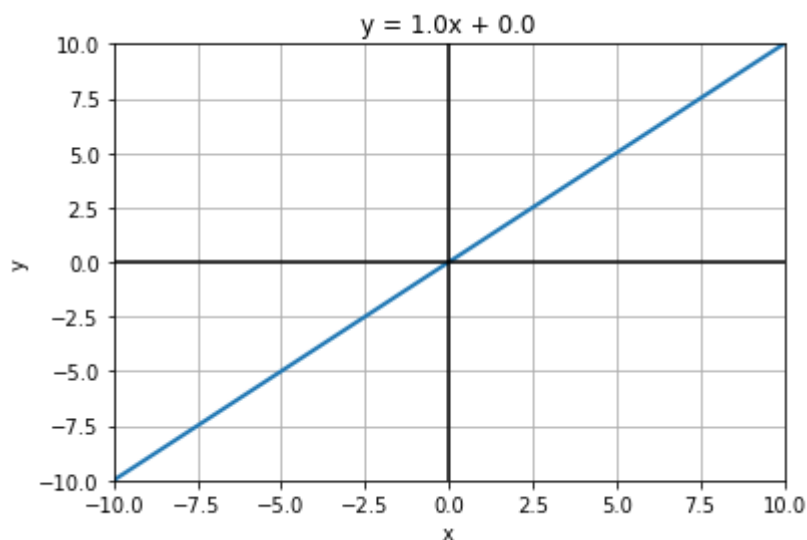
Merk derimot at når vi lager et interaktivt plot er det veldig lurt å låse aksene, ellers vil aksene endre skala når man endrer parametrene. Gjør dette med `xlim` og `ylim`, eller `axis()`.

In [12]:

```
1 interact(tegn_rett_linje, a=1.0, b=0.0);
```

a 1.00

b 0.00



Selv om funksjonen over så litt komplisert ut, vil den være stort sett lik også om vi ser på andre typer kurver. Her er for eksempel en lignende widget for en annengradsfunksjon, merk at tilnærmet all koden er lik:

In [13]:

```
1 from pylab import *
2
3 def tegn_annengrads(a, b, c):
4     # Regn ut x og y arrays
5     x = linspace(-10, 10, 101)
6     y = a*x**2 + b*x + c
7
8     # Plot kurve
9     plot(x, y, linewidth=2.0)
10
11     # Navn på akser og plott
12     xlabel("x")
13     ylabel("y")
14     title(f"y = {a}x² + {b}x + {c}")
15
16     # Lås aksene
17     xlim(-10, 10)
18     ylim(-10, 10)
19
20     # Tegn aksekors
21     grid()
22     axvline(0, color='black')
23     axhline(0, color='black')
24     show()
```

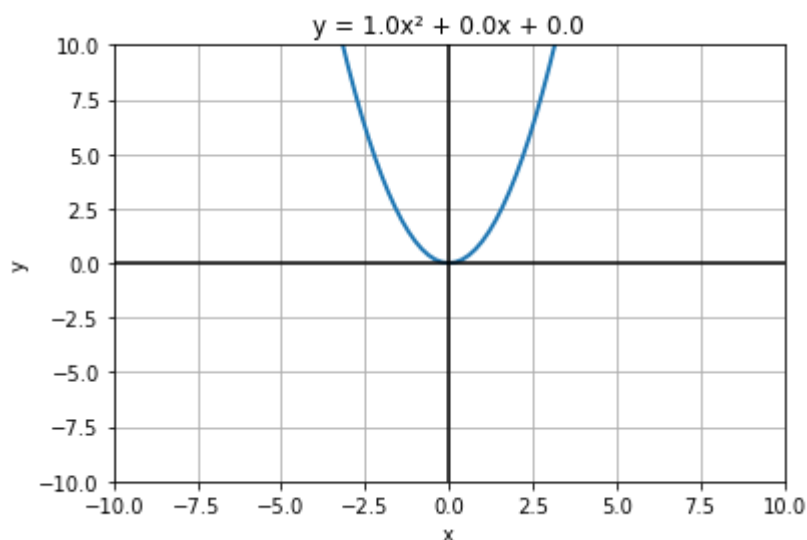
In [14]:

```
1 interact(tegn_annengrads, a=1.0, b=0.0, c=0.0);
```

a 1.00

b 0.00

c 0.00




Mer kontroll på mulig input


Så langt har vi bare satt startverdier, og latt `interact` bestemme hvilke intervaller som er fornuftig å lage på sliderene. Derimot er det ofte vi ønsker å selv sette disse grensene. For eksempel er det ofte `interact` vil gi verdier som er urimelige, eller ikke fungerer med funksjonen vi jobber med, eller det er interessante verdier vi gjerne vil kunne bruke som faller utenfor intervallet.

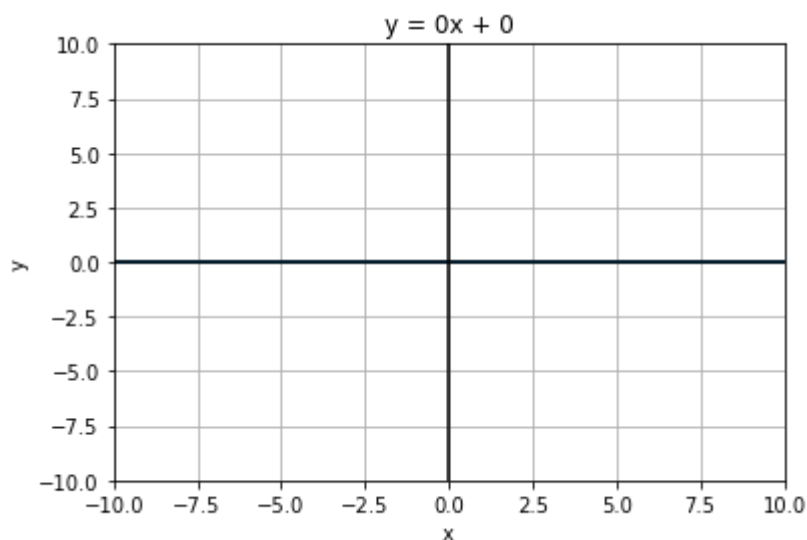
Vi kan få finkontroll på sliderne ved å gi flere verdier til en parameter når vi bruker `interact`. Ta for eksempel den rette linja vår. La oss si vi ønsker å kunne bruke verdier for a mellom -5 og 5, og verdier for a mellom -10 og 10. Da kan vi skrive som følger

In [15]:

```
1 interact(tegn_rett_linje, a=(-5, 5), b=(-10, 10));
```

a  0


b  0




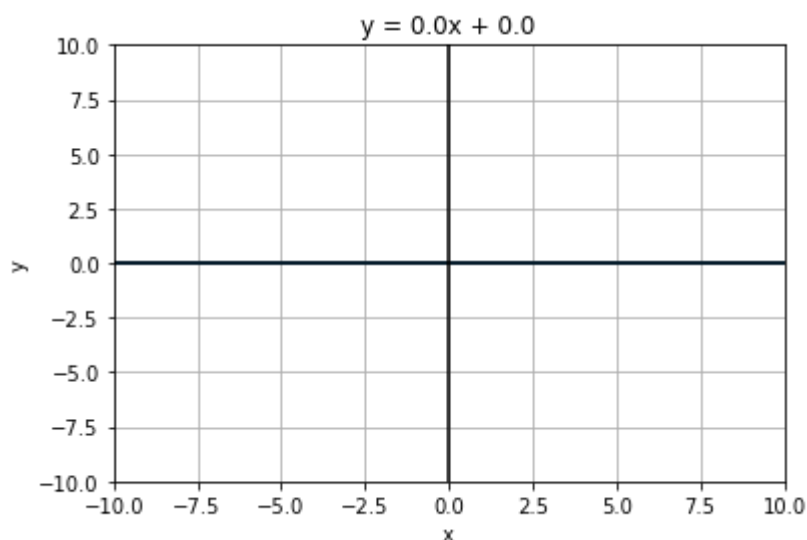
I tillegg kan vi spesifisere hvor store skritt slideren skal kunne bruke. Om vi vil ha finkontroll kan vi f.eks skrive $a = (-5, 5, 0.01)$. Da vil slideren endre seg i intervaller på 0.01. Om vi vil ha grovere kontroll kan vi f.eks skrive $a = (-5, 5, 0.5)$. På den måten er det langt enklere å lande på heltallsverdier for a :

In [16]:

```
1 interact(tegn_rett_linje, a=(-5, 5, 0.5), b=(-10, 10, 0.5));
```

a  0.00

b  0.00



Den eneste "ulempen" med denne måten å finjustere inputten på er at vi ikke lenger for lov til å sette startverdien selv, istedet legget `interact` den til midten av input intervallet. I dette tilfellet blir f.eks `a` og `b` satt til 0 ved start. Dette er sjeldent et stort problem i praksis, men kan være litt irriterende. Den letteste måten å fikse det på er å sette startverdien i selve `tegn_rett_linje` funksjonen, med

```
def tegn_rett_linje(a=1.0, b=0.0):  
    ...
```

Da vil vi kunne sette intervallet i `interact` og startverdien kommer fra selve funksjonen vi interagerer med.

Forsinket oppdatering for tynge beregninger

Så langt har vi sett på helt dynamiske widgets, når man trekker på slideren oppdaterer plottet seg "live" for hver minste endring man gjør. Dette er vel og bra, men for å få til dette må Python jobbe i kulissene med å gjenta beregninger og replotte hver gang en slider flytter seg et lite hakk. For en enkel beregning som en rett linje går dette kjapt nok til at det føles ganske jevnt og fint, men for tynge beregninger vil det ikke være like bra.

Ta for eksempel tilfeller der vi løser differensialligninger med Euler's metode, sånn som i Fysikk eksempelet vårt. Da tar selve beregningen og plottingen kanskje så mye som 10 sekunder. Da vil en slider rett og slett ikke fungere. For å fikse dette bør du istedenfor å bruke `interact`, bruke `interact_manual`. Denne fungerer helt likt, men vi får en tilleggsknapp man trykker på når man er klar for å kjøre på nytt.

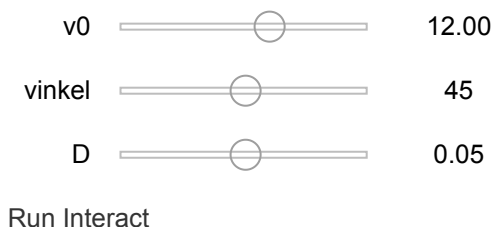
Som eksempel ser vi på et tilfelle der vi kaster en ball skrått og ser på hvordan ballen flyr avhengig av styrken på luftmotstanden og vinkelen vi kaster i

In [17]:

```
1  from pylab import *
2
3  def skråkast_med_luftmotstand(v0=12, vinkel=45, D=0.05):
4      # Sett parametere som ikke er justerbare
5      m = 0.25
6      g = 9.81
7
8      # Regn om til radianer
9      theta = vinkel*pi/180
10
11     # Tidsparametere
12     T = 4 # s
13     dt = 0.001 # s
14     N = int(T/dt) # Antall tidssteg
15
16     # Lag tomme arrays
17     x = np.zeros(N+1)
18     y = np.zeros(N+1)
19     vx = np.zeros(N+1)
20     vy = np.zeros(N+1)
21
22     # Startverdier
23     x[0] = 0 # m
24     y[0] = 1.5 # m
25     vx[0] = cos(theta)*v0
26     vy[0] = sin(theta)*v0
27
28     # Regne oss stegvis fremover i tid (Euler's Metode)
29     for i in range(N):
30         Vabs = sqrt(vx[i]**2 + vy[i]**2)
31         ax = -D*Vabs*vx[i]/m
32         ay = -g -D*Vabs*vy[i]/m
33
34         vx[i+1] = vx[i] + ax*dt
35         vy[i+1] = vy[i] + ay*dt
36         x[i+1] = x[i] + vx[i]*dt + 0.5*ax*dt**2
37         y[i+1] = y[i] + vy[i]*dt + 0.5*ay*dt**2
38
39     # Plot endelig løsning
40     plot(x, y)
41     axis((0, 20, 0, 10))
42     xlabel('x')
43     ylabel('y')
44     title('Skrått kast med luftmotstand')
45     show()
```

In [18]:

```
1 from ipywidgets import interact_manual
2
3 interact_manual(skråkast_med_luftmotstand, v0=(0, 20, 0.5), vinkel=(0, 90, 5),
```



Widgets fra fil

Så langt har vi lagd og kjørt widgets direkte i notebooken, men av og til vil man lage en widget på forhånd man kun vil at elever skal bruke, ikke nødvendigvis se eller endre koden bak. En måte å få til dette er å legge koden som definerer og kjører widgeten i en fil, og deretter importerer den rett inn i notebooken. Denne filen kan du for eksempel skrive i Spyder. Det må være en Pythonfil, altså med `.py`-endelse, og filen må ligge i samme mappe som notebooken.

Nå har vi for eksempel laget en fil som heter `primtall_widget.py`, som ligger i samme mappe som denne notebooken, da kan vi importere og se på denne:

In [19]:

```
1 import primtall_widget
```

n

1 er ikke et primtall

Mer om Widgets

Nå har vi vist noen eksempler av hvordan man kan lage og bruke widgets. For mer info kan du for eksempel se her:

- [https://ipywidgets.readthedocs.io/en/stable/examples/Using%20Interact.html#\(https://ipywidgets.readthedocs.io/en/stable/examples/Using%20Interact.html\)](https://ipywidgets.readthedocs.io/en/stable/examples/Using%20Interact.html#(https://ipywidgets.readthedocs.io/en/stable/examples/Using%20Interact.html))