KRASJKURS I PYTHON FOR MATEMATIKK

"It is impossible to exaggerate the extent to which modern applied mathematics has been shaped and fueled by the general availability of fast computers with large memories. Their impact on mathematics, both applied and pure, is comparable to the role of the telescopes in astronomy and microscopes in biology."

— Peter Lax, Siam Rev. Vol. 31 No. 4

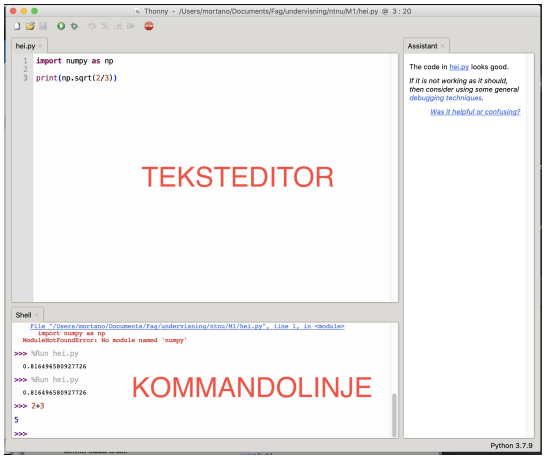
Det er kjekt å kunne minst ett programmeringsspråk når man skal drive med matematikk. Da jeg var student var det JAVA som var det store, og nå er det PYTHON. Når Aslak var student på 90-tallet, var det PASCAL. Lars Lundheim og Tante Siri lærte FORTRAN på 60- og 70-tallet. Om ti eller tjue år er det helt sikkert noe annet. Slik er livet med EDB. Det er noe nytt hver dag.

Naboen min Gudbrand er programmerer, og han sier at det finnes ikke ett språk som gjør alt. En programmerer må derfor kunne en haug med språk. Å lære seg et nytt programmeringsspråk er ikke så vanskelig dersom man allerede kan ett språk, så vi kan like gjerne begynne med python, selv om det er stor sjanse for at du trenger noe annet siden. Python er ikke det enkleste å bruke til matematikk, men det funker greit.

Komme igang

0 Kast kalkulatoren din. Du kommer aldri til å få bruk for den mer, med mindre den har solcellepanel og du skal ta den med på en øde øy.

1 Børge Haugset (koordinator i ITGK) sier at det enkleste er å installere THONNY. Hvis du gjør det, kan du hoppe over neste steg. Her er et bilde av thonny:



2 Hvis du vil gjøre det litt mer komplisert (det må du nok før eller siden, men det kan være noen år til):

Installer Python 3 etter oppskriften HER, og en valgfri IDE (Integrated Developer Environment). SPYDER, VS CODE, PYCHARM og THONNY fungerer alle sammen helt fint. Velg og vrak. Drift på IMF har en oppskrift på hvordan du går frem om du vil gå for ANACONDA. Per Kristian sier at det beste (hvis du vil holde det enkelt) er å installere SPYDER separat (altså ikke via Anaconda).

De barskeste installerer aldri noen IDE, men skriver kodene i en tilfeldig valgt tekststeditor og kjører dem i terminalen på laptopen sin. Men dette handler mer om barskhet enn om ferdighetsnivå. Jeg er selv en elendig programmerer, men bruker tekststeditor og terminal fordi det får meg til å føle meg helt utrolig barsk.

3 Husk til slutt at det er millioner av nerder som driver med programmering og internett, så all informasjon du noensinne kommer til å trenge om EDB, er bare et tastetrykk unna. Gudbrand sier det er en kunst å google riktig.

Gudbrands regel

GÅ RETT PÅ GOOGLE MED EN GANG DU STÅR FAST!

Hvis du ønsker å bruke et annet språk enn python i dette kurset er det helt greit. Alt er lov. Jeg liker best R. Dette er et språk som brukes mye til statistikk. Siden jeg var vokste opp med MATLAB, syntes jeg det var enda litt lettere å komme i gang med enn python, men Børge sier Python er like lett. Du finner oppskrift på å komme igang HER. Husk til slutt:

To empiriske EDB-lover

1. Hvis du lurer på noe om datamaskinen din, må du spørre noen som er eldre enn deg.
2. Hvis du lurer på noe om mobiltelefonen din, må du spørre noen som er yngre enn deg.

Python som kalkulator

Det første vi må gjøre, er å finne ut hvordan vi kan bruke python som kalkulator, slik at det blir trygt å kaste den i bosset. Det kommer her. Alt som står i en slik ramme:

```
2 / 3
```

er kode du skal skrive i kommandolinjen og trykke på enter. Vi begynner med de vanlige regnereglene.
Addisjon:

```
2 + 3
```

Subtraksjon:

```
2 - 3
```

Multiplikasjon:

```
2 * 3
```

Divisjon:

```
2 / 3
```

Potenser:

```
2**3
```

Kvadratrøtter:

```
2**.5
```

I matematikk dropper vi av og til gangetegnet, og skriver

$$2(2+3).$$

Dette går ikke i python. Du må skrive

```
2*(2 + 3)
```

Variable

Nå kan du gå videre til å lime koden inn i teksteditoren over kommandolinjen og så kjøre koden ved å trykke på F5, eller 'Run current script' i menyen. Vi lagrer variable slik:

```
a=2*(2 + 3)
b=7
```

Vil du se hva variabelen a er for tiden, skriver du

```
print(a)
```

og så får du en utskrift av verdien lagret i a. Du kan herje med variable akkurat som med tall

```
print(a+b)
```

Hvis du er lei av verdien $a = 12$, kan du overskrive den

```
a=2
```

Numpy

Du må fortelle python nøyaktig hva du har tenkt å gjøre, ellers skjønner python ingenting. Vanligvis importerer man en pakke med matematikkfunksjoner ved å skrive

```
import numpy as np
```

på første linje i filen. Første gang du bruker thonny kan det tenkes du må installere numpy. Dette gjør du ved å gå på Tools>Manage Packages og så lete opp numpy og trykke på install.
Python kan brukes til mange ting, og matematikk er bare en av dem. Python er ikke født med kunnskap om hva akkurat du skal bruke python til, så derfor må du importere denne pakken. Nå har man tilgang på de vanligste matematiske regneoperasjonene. Men ikke for eksempel plotting, da må du skrive noe mer greier. Vi kommer tilbake til det.

Lineæralgebra

Vi begynner med å lage en matrise

```
A=[[2,3,4],[3,4,5],[4,5,7]]
```

og en søylevektor

```
b=[[4],[5],[3]]
```

Dersom vi ikke gidder å gausseliminere, kan python løse likningssystemet

$$A\mathrm{x} = \mathrm{b}$$

for oss:

```
x=np.linalg.solve(A,b)
```

Dette er en typisk ting som gjør Python litt mer komplisert å lære, om du skal programmere matematikk. Du må stadig fortelle Python at du driver på med matematikk, og ikke programmerer internettsider eller dataspill. Spesielt lineæralgebra er håpløst kløynete. (MATLAB er mye bedre, og gratis for ntnu-studenter.) Prøv å skrive

```
b+b
```

og se hva som skjer. Det er mulig å få Python til å følge vanlige regnereglene for vektorer, men du må vite hva kommandoene heter:

```
np.add(b,b)
```

De vanligste matematiske operasjonene du har lært, er implementert i Numpy. Dette biblioteket importerte vi til å begynne med som 'np', og lurer du på hvordan en funksjon fungerer, googler du 'numpy operasjonen du lurer på'. Matrise-vektorproduktet er implementert i en en funksjon som heter np.dot.

```
np.dot(A,b)
```

Skal du ha lengden til vektoren b, skriver du

```
np.linalg.norm(b)
```

Igjen, det finnes enklere språk for lineæralgebra (for eksempel matlab), men akkurat nå er det python som er på moten, så vi får lære oss det.

Plotting

Nå kan vi nok til å plotte funksjoner. Det første du må gjøre er å importere matplotlib

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

Sjekk for øvrig matplotlib.org.
Her er en ting som er viktig å forstå: På kalkulatoren din er det mulig å skrive inn et matematisk funskjonsuttrykk med tilhørende definisjonsmengde, og så få ut et plot. Dette kan ikke Python. Python plotter to vektorer mot hverandre. La oss illustrere med å plotte funksjonen $f(x) = x \sin x$ på intervallet $[0, 2\pi]$. Man lager først en vektor med de verdiene man vil ha på x-aksen:

```
np.linspace(0, 2*np.pi, num=1000)
```

Denne kommandoen lager et gitter med 1000 ekvidistante verdier fra 0 til 2π. Nå lager vi en (like lang) vektor med funksjonsverdier

```
y=np.multiply(x,sin(x))
```

Vi skriver nå

```
plt.plot(x,y)
```

for å lage figuren, og

```
fig.savefig("en_flott_figur.png")
```

for å lagre den. Python tegner en rett strek mellom punktene i plottet, men hvis det er tett nok mellom verdiene på x-aksen vil plottet fremstå som en glatt kurve.
Hvis du har lyst til å plotte to funksjoner i samme figur, for eksempel $f(x) = x \sin x$ og $g(x) = x^2 \sin x$, skriver du

```
np.linspace(0, 2*np.pi, num=1000)
y=np.multiply(x,sin(x))
z=np.multiply(x**2,sin(x))
plt.plot(x,y)
plt.plot(x,z)
plt.show()
```

For og if

En for løkke er en måte å få datamaskinen til å gjøre samme ting om igjen, men med en parameter som endres for hver gang:

```
for i in range(5):
  print(i)
```

If kan brukes til å gjøre unntak:

```
for i in range(5):
  if i == 3:
    print(i)
```

Python bruker indentering for å skjønne hva som er inne i for-løkken, og hva som er utenfor. Bruk tab-tasten for korrekt indentering, hvis ikke teksteditoren din får til sånt av seg selv. Du må også fortelle python at løkken er ferdig ved å kutte et knepp i indenteringen i første linje under løkken.