Introduktion

I denna datorövning ska vi undersöka den centrala gränsvärdessatsen, samt generera en del slumpässiga tal. Den centrala gränsvärdessatsen är viktig eftersom den svarar på frågor som angår, t.ex., fördelningen man får om man adderar upp många oberoende observationer med olika mätfel i ett experiment. I denna övning ska du undersöka summor av slumpmässiga tal - på vilket sätt är det annorlunda om du forändrar antalet termer i summan? Vad är fördelningen av varje slumpässigt tal i summan?

Nyttiga funktioner

```
import scipy.stats as sps
#Some random distributions in scipy.stats are:
#sps.uniform #uniform between 0 and 1
#sps.norm#   #normal distribution, mean=0, sigma=1
#sps.cauchy  #Cauchy distribution- pdf = 1/(\pi*(1+x^2))

#You can generate a long vector of random values like this:
nVals=10000000;
r = sps.uniform.rvs(size=nVals);

#Alternatively you can use numpy.random like this:
import numpy.random as nr
#nr.normal #normal distribution, mean=0, sigma=1
#nr.uniform #uniform between 0 and 1
#nr.standard_cauchy #Cauchy distribution with mode 0
#and so on
```

Ta en titt på dokumentationerna för flera `scipy.stats` och `numpy.random` fördelningar.

Obligatoriska uppgifter

- Plotta ett histogram av 10000 summor av n slumpmässiga tal från en uniform fördelning för $n =$:
  - 2
  - 5
  - 12
- Hur förändrar histogrammet om du byter till några andra fördelningar, t.ex.:
  - Normalfördelning?
  - Poissonfördelning?
  - Cauchyfördelning?
- Räkna ut medelvärdet och standardavvikelsen för din summa, och plotta en normalfördelning över varje. Är någon av fördelningarna annorlunda[1]?

[1] Ledtråd: Vad får du ifrån `sps.cauchy.var()`? Kan man tillamäpa CGS när man har med många oberoende Cauchy fördelningar att göra

- Många matematiska modeller, till exempel för aktiepriser, antar att förändringar i priser över tid är normalfördelade. Vilka antaganden måste du göra för att det ska hålla?
- Vad händer om många slumpmässiga, positiva tal multipliceras? Kommer produkten att konvergera mot någon fördelning?

Vi förväntar oss att ni kan lösa övningarna hittills under datorövningen. Vi förväntar oss dessutom att de som hinner försöker lösa de följande uppgifterna.

Uppgifterna för de som hinner

Ta en titt på dokumentationen av `scipy.stats.norm` på denna länk. Det finns många nittiga funktioner som ni kan tänka använda er av, t.ex.:

- `scipy.stats.norm.cdf`
- `scipy.stats.norm.sf`
- `scipy.stats.norm.ppf`
- `scipy.stats.norm.isf`

`scipy.stats.norm.cdf` har med fördelningsfunktionen ("cumulative distribution function") $F(x)$ att göra, dvs, $F(x)$ räknar ut sannolikheten att slumpvariabeln (X) antar värden som är mindre än eller lika med värdet som betraktas (x). På samma sätt har `scipy.stats.norm.sf` med "survival function" $1 - F(x)$ att göra, dvs sannolikheten att slumpvariabeln (X) antar värden som är större än eller lika med värdet som betraktas (x). Bekanta er själva med de här funktionerna och försöka ta reda på vad `scipy.stats.norm.isf` och `scipy.stats.norm.ppf` är.

- Om ni vill räkna ut α-kvantilen av normalfördelningen, dvs, lösningen till $F(x) = 1 - \alpha$, vilka av de ovanstående funktioner är det enklaste att använda?
- Om ni vill räkna ut $100 \cdot r$%-percentilen i stället, dvs, lösningen till $F(x) = r$?

Ni ha kanske hört tala om 1.96, som är inte bara Zlatan's höjd i meter, utan 97.5-percentilen av normalfördelningen. Eftersom normalfördelningen är symmetrisk, innebär det att 95% av ytan under en normalfördelning finns mellan -1.96 och 1.96 standardavvikelser.

- Använda er av två av de ovanstående funktioner (`scipy.stats.norm.xxx`) för att härleda värdet 1.96.
- Nu ska ni använda er av Monte-Carlo metoder (dvs att använda datorgenererade slumptal för att lösa problem) för att härleda värdet 1.96. Ni kan göra det så här:
  - Generera ett stort antal slumpmässiga tal från en standard normalfördelning
  - Beräkna för vilket värde x det gäller att andelen tal större än x är lika med 2.5% av det totala antalet genererade slumptal. (det kan hjälpa om ni använder er av `.sort()`-metoden)
  - Jämföra med värdet som ni får om ni använder er av `np.percentile` i stället för metoden ovan.