tma4240-jarle-2026h

September 18, 2026

1 TMA4240 Høst 2026

1.1 Uke 3

1.1.1 Simulering av pseudotilfeldig tall

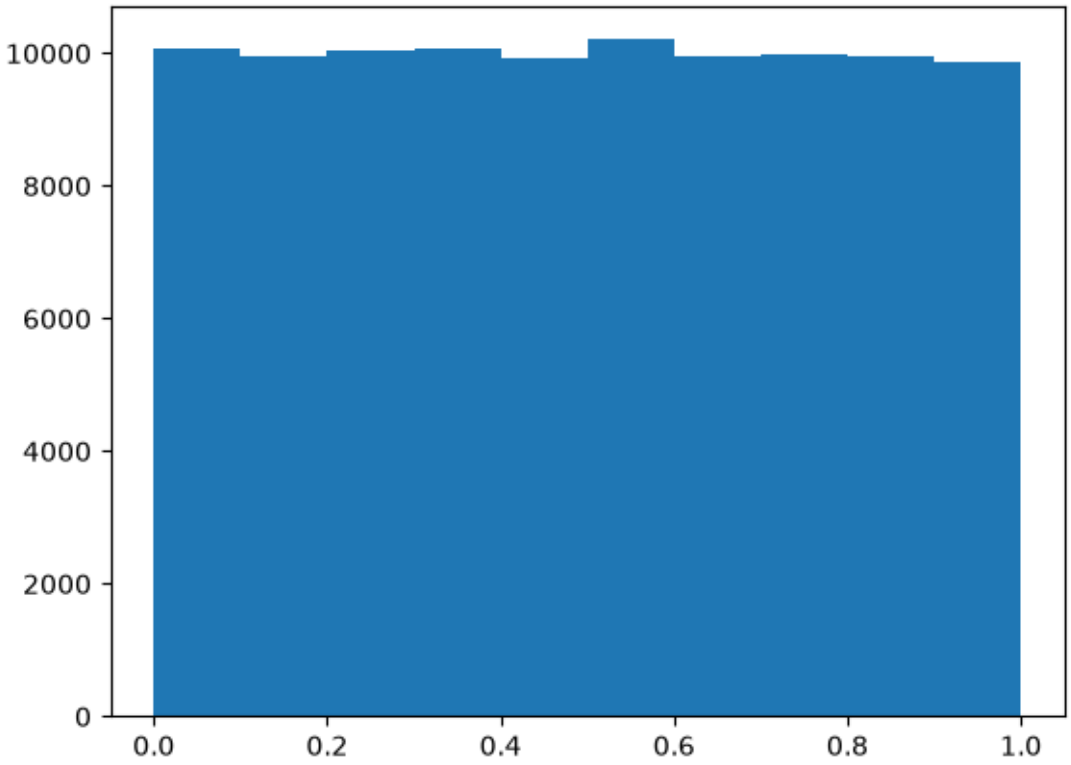
`numpy.random.uniform` genererer pseduotilfeldige flyttall med tetthet $f(u) = 1$ for $0 < u < 1$. Ved sette tallgeneratoren i en viss tilstand med `numpy.random.seed` kan samme sekvens gerereres på nytt, slik at en simulering kan gjøres *reproduserbar* (se https://en.wikipedia.org/wiki/Reproducibility).

[105]:
```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(7)
u = np.random.uniform(0,1,100000)
print(u[1:50])
```

```
[0.77991879 0.43840923 0.72346518 0.97798951 0.53849587 0.50112046
 0.07205113 0.26843898 0.4998825  0.67923    0.80373904 0.38094113
 0.06593635 0.2881456  0.90959353 0.21338535 0.45212396 0.93120602
 0.02489923 0.60054892 0.9501295  0.23030288 0.54848992 0.90912837
 0.13316945 0.52341258 0.75040986 0.66901324 0.46775286 0.20484909
 0.49076589 0.37238469 0.47740115 0.36589039 0.83791799 0.76864751
 0.31399468 0.57262533 0.27604905 0.45284293 0.35297837 0.65739946
 0.37035108 0.45909298 0.71932412 0.41299183 0.90642327 0.18045162
 0.74111887]
```

[99]:
```
plt.hist(u, bins=10)
plt.plot()
```

[99]:
```
[]
```



Lineær kongruensgenerator Vi kan lage vår egne pseudotilfeldige tallgenerator ved å iterere rekursjonen

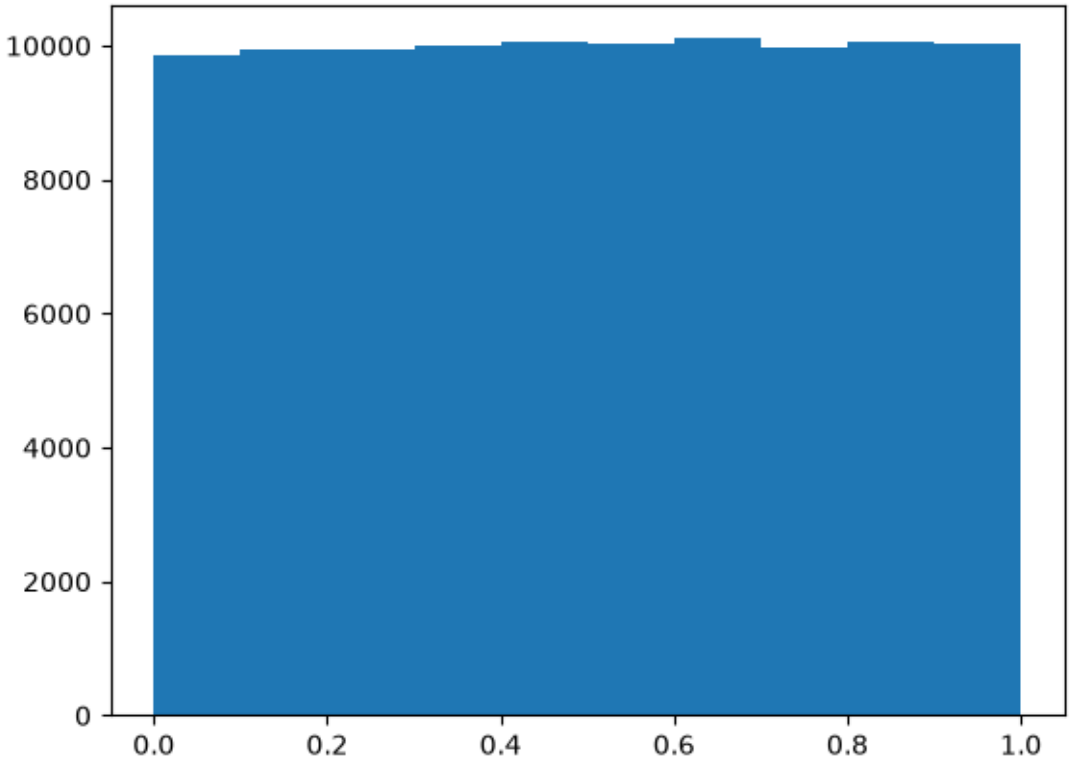
$$x_{t+1} = (ax_t + c) \mod b$$

for passende positive heltall a, b og m.

[107]:
```
def lcg(seed, a=1664525, c=1013904223, m=2**32, n=10):
    """Generate n pseudo-random numbers using a linear congruential generator.
    """
    numbers = []
    x = seed
    for _ in range(n):
        x = (a*x + c) % m
        numbers.append(x / m)  # Normalize to [0, 1)
    return numbers

# Example usage
u = lcg(seed=42, n=100000)
```

[108]:
```
plt.hist(u, bins=10)
plt.plot()
```

[108]: []



Ekte tilfeldige tall (TRNG) Ekte tilfeldig tall er noe mer kostbart å generere men å foretrekke i noen sammenhenger (vinlotterier, kryptografi). Drevet fram av behovet for sikker kryptert kommunikasjon over internett begynte de fleste CPUer fra omkring 2012-2015 å inneholde hardware som kan levere tilfeldig tall basert på esktrahering av entropi fra fysisk mikroskopisk støy i prosessoren. Se https://en.wikipedia.org/wiki/Hardware_random_number_generator. Følgende kode gir tilgang til TRNG generatoren i Python:

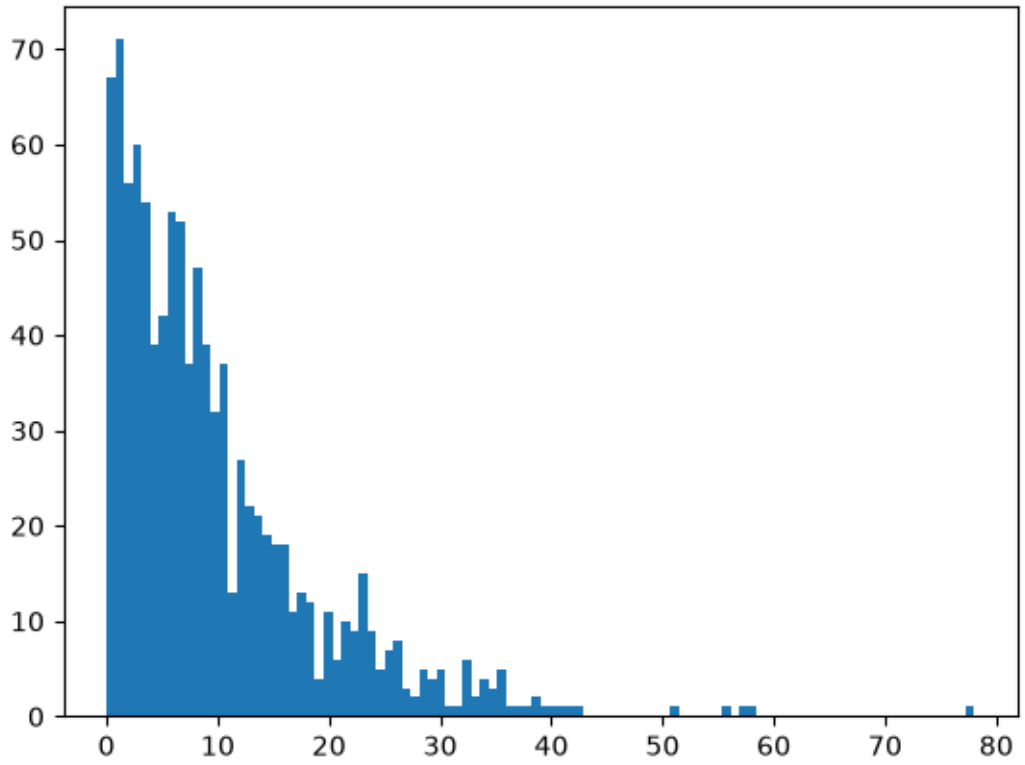
[67]:
```
import random
hardware_rand = random.SystemRandom()
u = hardware_rand.uniform(0, 1)
print(u)
```

```
0.2628641407281267
```

1.1.2 Inversjonsmetoden: Simulering fra vilkårlig fordeling

Eksempel, "eksponentiell" fordeling Anta at X er kontinuerlig fordelt med tetthet $f(x) = \frac{1}{10}e^{-x/10}$. Da er $F(x) = 1 - e^{-x/10}$ og $F^{-1}(u) = -10\ln(1-u)$. Realisasjoner av X kan da generes ved inversjonsmetoden som følger:

[110]:
```
u = np.random.uniform(0,1,1000)
x = -10*np.log(1 - u)
plt.hist(x, bins=100);
```



Eksempel, diskret fordeling med kjent F^{-1}, antall øyne på en terning X er antall øyne slik at $F(x) = \frac{\lfloor x \rfloor}{6}$. Da er kvantilfunksjonen gitt ved

$$F^{-1}(u) = \inf\{x : F(x) \geq u\} = \lceil 6u \rceil$$

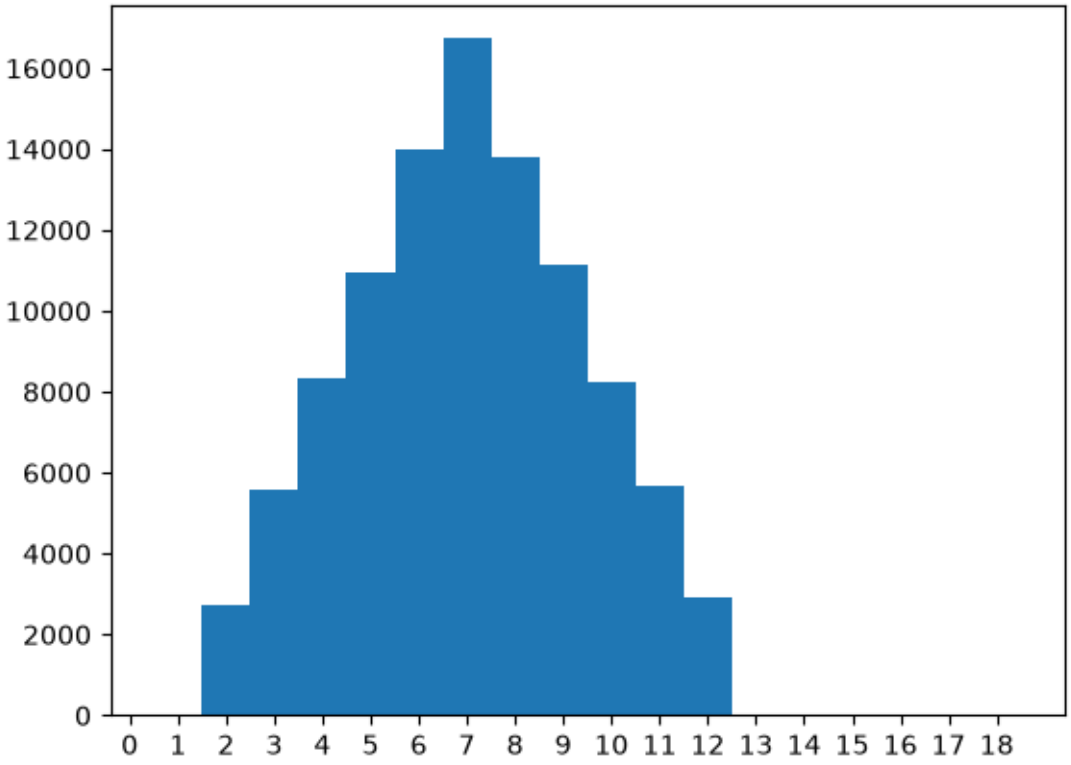
og realisasjoner av X kan simuleres med følgende kode:

[113]:
```
u = np.random.uniform(0,1,100)
x = np.ceil(6*u)
print(x)
```

```
[1. 2. 4. 1. 2. 6. 1. 3. 3. 3. 2. 6. 6. 1. 4. 5. 6. 1. 5. 4. 6. 2. 2. 5.
 1. 6. 6. 6. 3. 4. 6. 3. 3. 2. 5. 3. 5. 3. 4. 4. 2. 6. 4. 4. 6. 1. 3. 1.
 4. 4. 6. 3. 3. 6. 2. 3. 3. 5. 3. 6. 5. 2. 3. 5. 4. 1. 2. 2. 3. 2. 4. 1.
 2. 2. 1. 2. 5. 4. 1. 2. 3. 6. 1. 1. 1. 3. 4. 6. 4. 4. 4. 2. 1. 4. 6. 3.
 4. 4. 2. 4.]
```

Simulering av summer av stokastiske variable La X, Y, Z være antall øyne på tre terninger og la $W = X + Y + Z$. Realisasjoner av W kan da simuleres på følgende måte:

[115]:
```
u = np.random.uniform(0,1,100000)
x = np.ceil(6*u)
u = np.random.uniform(0,1,100000)
y = np.ceil(6*u)
u = np.random.uniform(0,1,100000)
z = np.ceil(6*u)
w = x + y
plt.hist(w,bins=np.arange(0.5, 19.5, 1))
plt.xticks(range(0, 18 + 1))
plt.show()
```



Eksempel, diskret fordeling med kjent F^{-1}, antall terningkast før 6 på terningen X er antall terningkast t.o.m. første sekser. Som vist i forelesningen er kumulativ fordeling da gitt ved

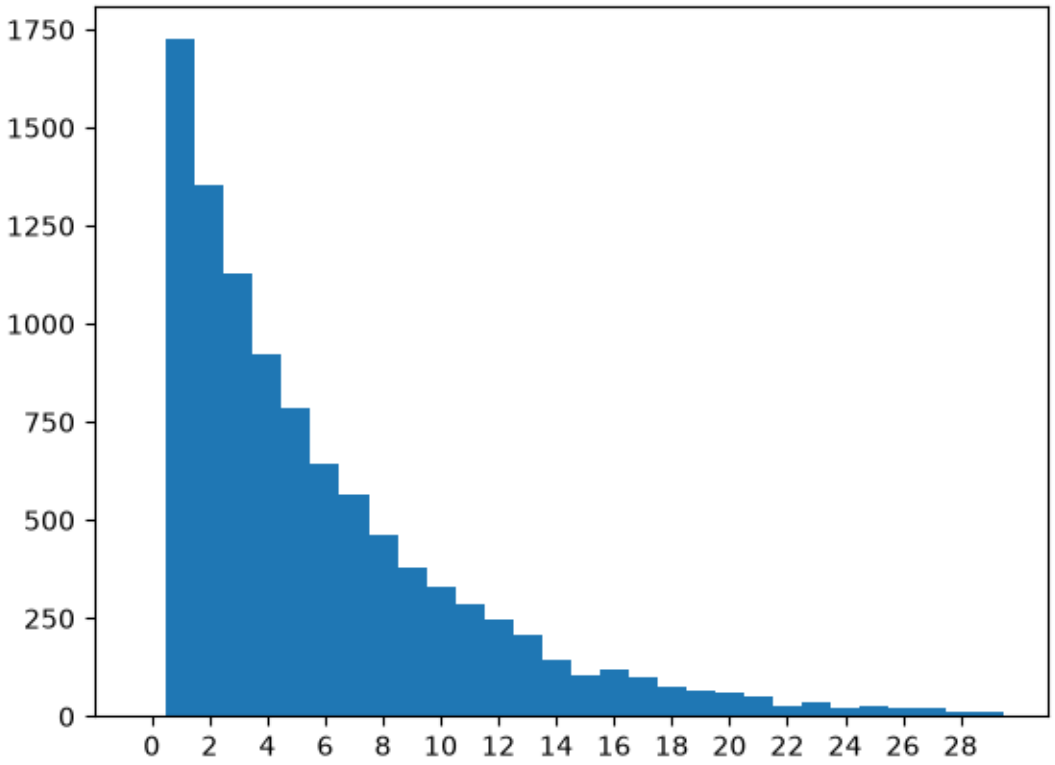
$$F(x) = 1 - \left(\frac{5}{6}\right)^{\lfloor x \rfloor}$$

og kvantilfunksjonen ("invers" til F) ved

$$F^{-1}(u) = \inf\{x : F(x) \geq u\} = \lceil \ln(1-u)/\ln(5/6) \rceil$$

og vi kan generere realisasjoner av X med inversjonsmetoden som følger:

[116]:
```
u = np.random.uniform(0,1,10000)
x = np.ceil(np.log(1 - u)/np.log(5/6))
plt.hist(x, bins=np.arange(-0.5, 30.5, 1))
plt.xticks(range(0, 30, 2))
plt.show()
```



1.1.3 Forventningsverdi til stokastiske variable $E(X)$

Store talls lov La $X_1, X_2, \ldots$, være sekvens av uavhengige stokastiske variable med sammme endelige forvetning $\mu = E(X)$. La $\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$. Når $n \to \infty$ vil $\bar{X}_n \to E(X) = \mu$ (i sannsynlighet "in probability" og nesten sikkert "almost surely")

Eksempel: Terningkast Hvis $f(x) = 1/6$ for $x = 1, 2, \ldots, 6$ er $F(x) = \lfloor \frac{x}{6} \rfloor$ og kvantilfunksjonen $F^{-1}(u) = \lceil 6u \rceil$. Kan dermed generere terningkast og beregne kumulative gjennomsnitt som følger:

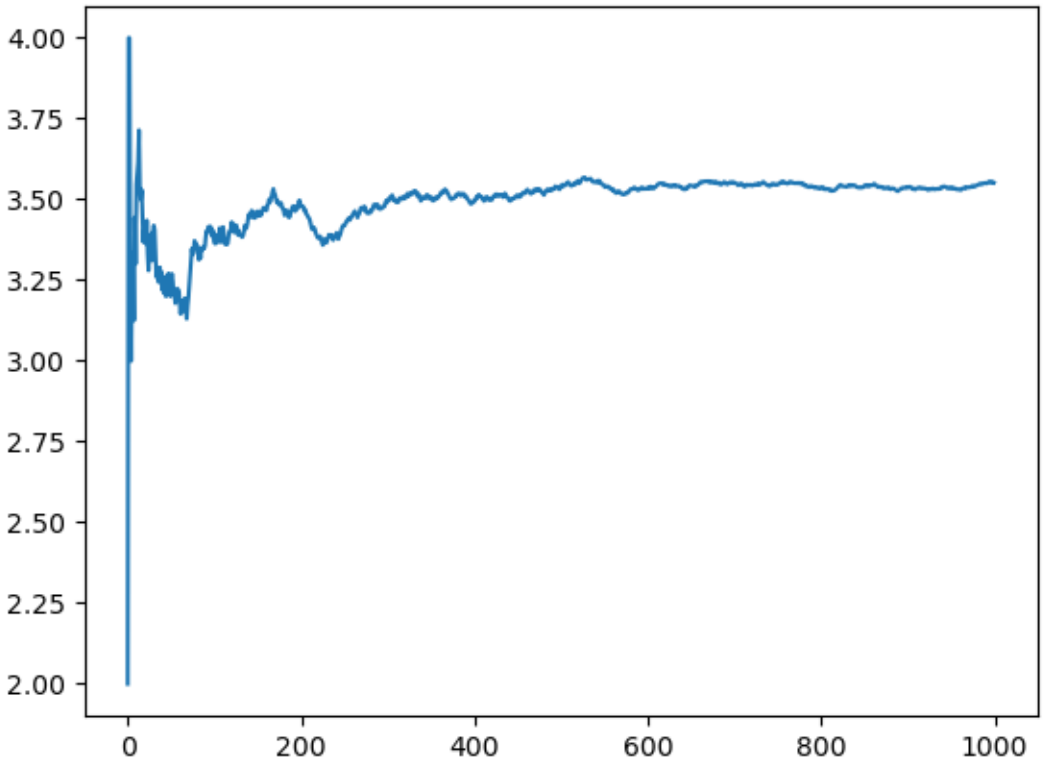
[112]:
```
u = np.random.uniform(0,1,1000)
x = np.ceil(6*u)
x[1:100]
```

[112]:
```
array([2., 5., 6., 4., 4., 3., 2., 4., 4., 5., 2., 3., 1., 4., 1., 1., 1.,
       4., 3., 3., 6., 3., 4., 4., 5., 3., 2., 1., 2., 5., 2., 3., 3., 2.,
       1., 6., 5., 1., 3., 2., 4., 2., 1., 3., 2., 3., 1., 1., 2., 4., 1.,
```

```
3., 6., 5., 1., 5., 4., 3., 1., 3., 4., 1., 6., 1., 1., 2., 2., 1.,
6., 2., 4., 4., 4., 2., 2., 5., 2., 6., 4., 1., 2., 6., 6., 5., 6.,
1., 1., 4., 4., 2., 1., 3., 5., 2., 3., 1., 1., 1., 3.])
```

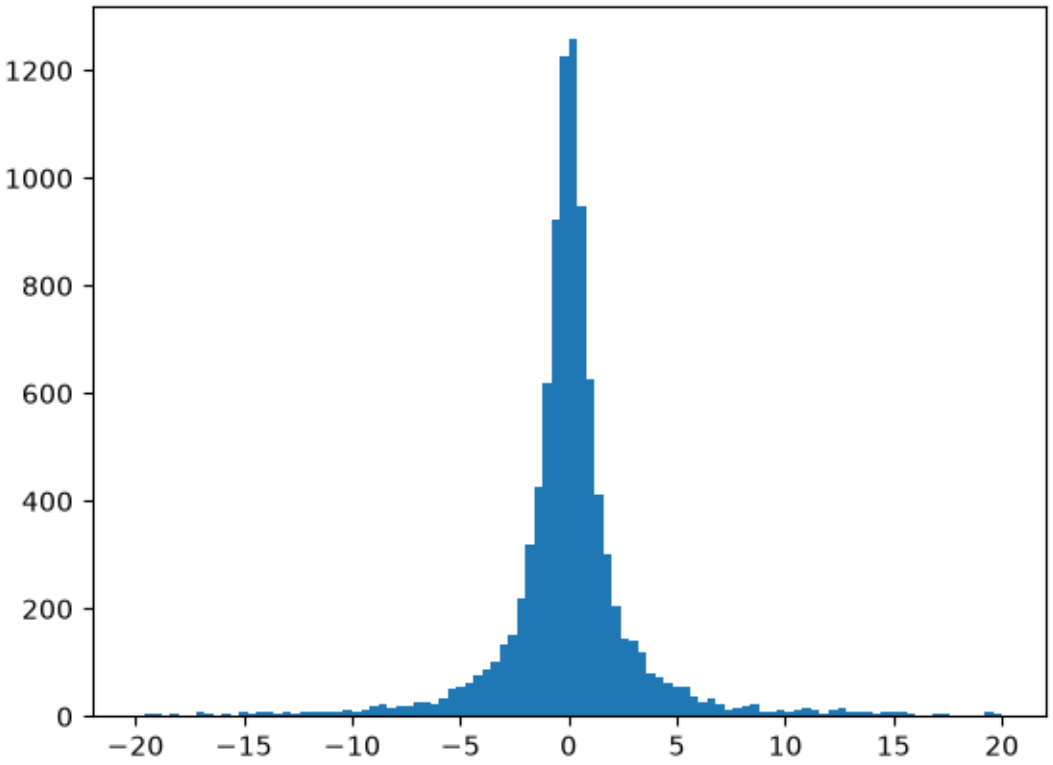
[200]:
```
cummean = np.cumsum(x)/np.arange(1,1001)
plt.plot(cummean)
```

[200]:
```
[<matplotlib.lines.Line2D at 0x17709f530>]
```



Eksempel: Ikke-endelig $E(X)$ Hva skjer hvis $E(X)$ ikke er endelig? La X ha tetthet $f(x) = \frac{1}{\pi(1+x^2)}$ (Cauchy-fordelingen). Kan da generere X ved å sette $X = F^{-1}(U) = \tan(\pi(U - 1/2))$.

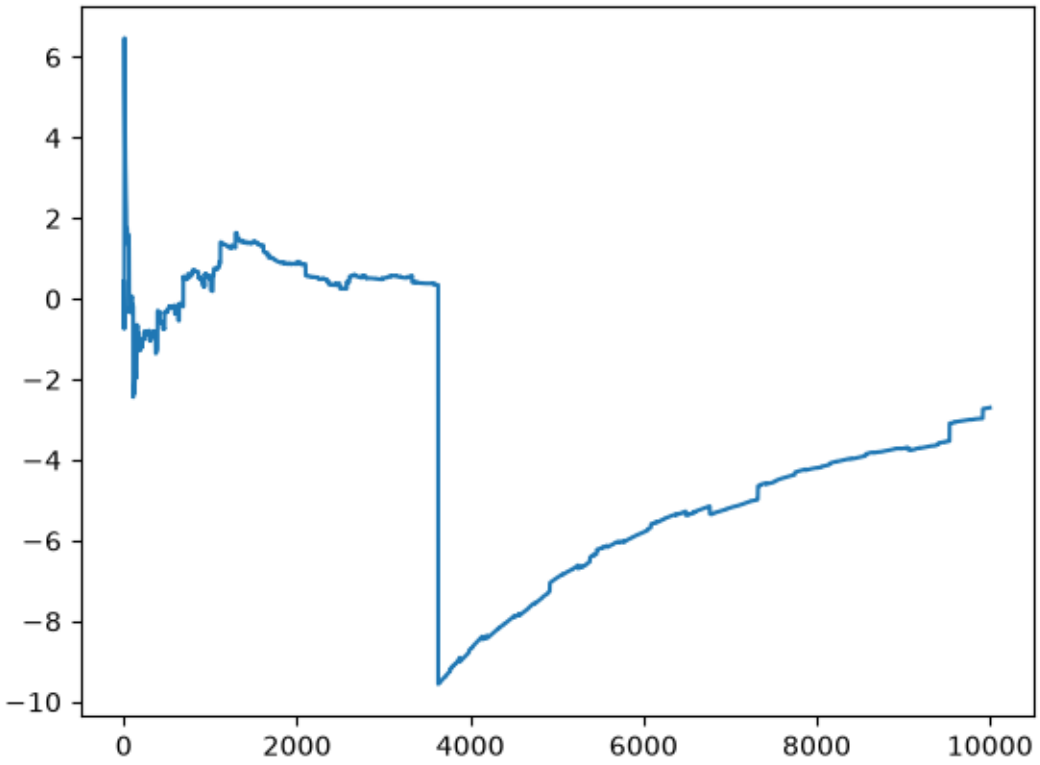
[111]:
```
u = np.random.uniform(0,1,10000)
x = np.tan(np.pi*(u-1/2))
plt.hist(x,range=(-20,20), bins=100);
```



Men integralet $E(X)$ ikke endelig (vist på tavlen). Så hva skjer med $\bar{X}_n$ når $n \to \infty$?

[85]:
```
cummean = np.cumsum(x)/np.arange(1,10001)
plt.plot(cummean)
```

[85]:
```
[<matplotlib.lines.Line2D at 0x1243c1be0>]
```



Eksempel: Forventet absoluttavstand mellom to uniformt fordelte variable Hvis X og Y er uavhengig $U(0,1)$ er $E\underbrace{|X-Y|}_{g(X,Y)} = \int_0^1 \int_0^1 \underbrace{|x-y|}_{g(x,y)} f_{X,Y}(x,y)dxdy = 1/3$ (vist på tavlen). Sanity sjekk:

[117]:
```
x = np.random.uniform(0,1,10000)
y = np.random.uniform(0,1,10000)
absoluttavstand = np.abs(x-y)
np.mean(absoluttavstand)
```

[117]:
```
np.float64(0.3342537163967207)
```

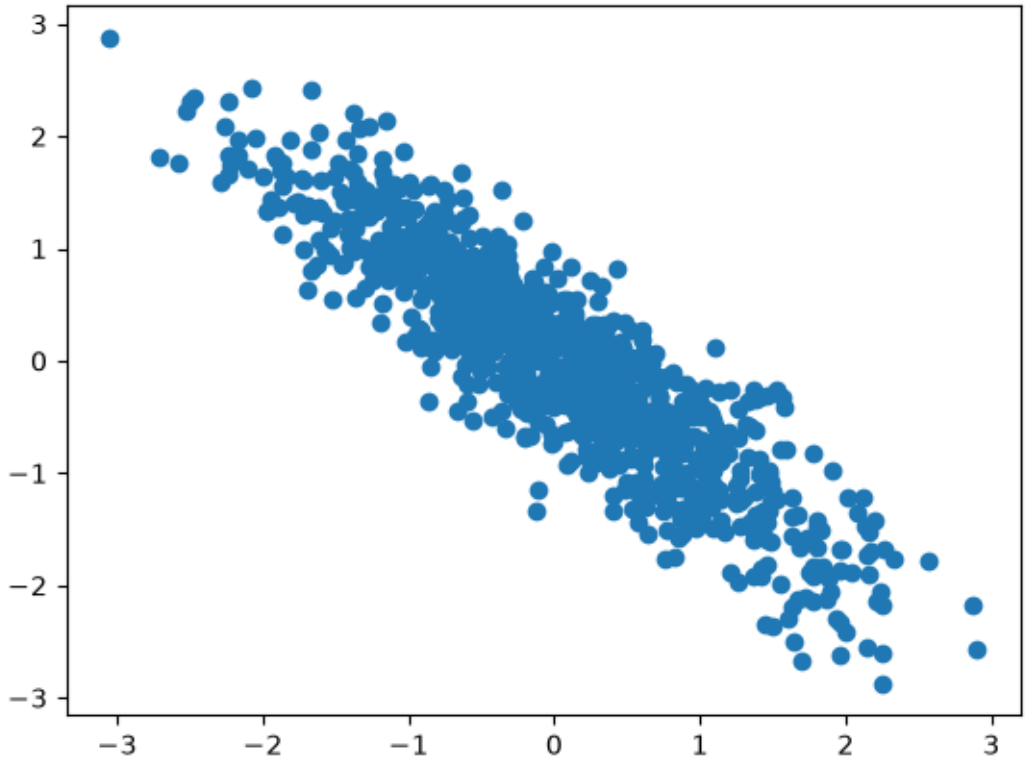
1.1.4

1.2 Uke 4

Eksempel på korrelerte variable (binormalfordeling):

[135]:
```
rho = -.9
x = np.random.normal(0,1, 1000)
y = rho*x + np.sqrt(1-rho**2)*np.random.normal(0,1,1000)
plt.scatter(x,y)
```

[135]: <matplotlib.collections.PathCollection at 0x117b77b10>



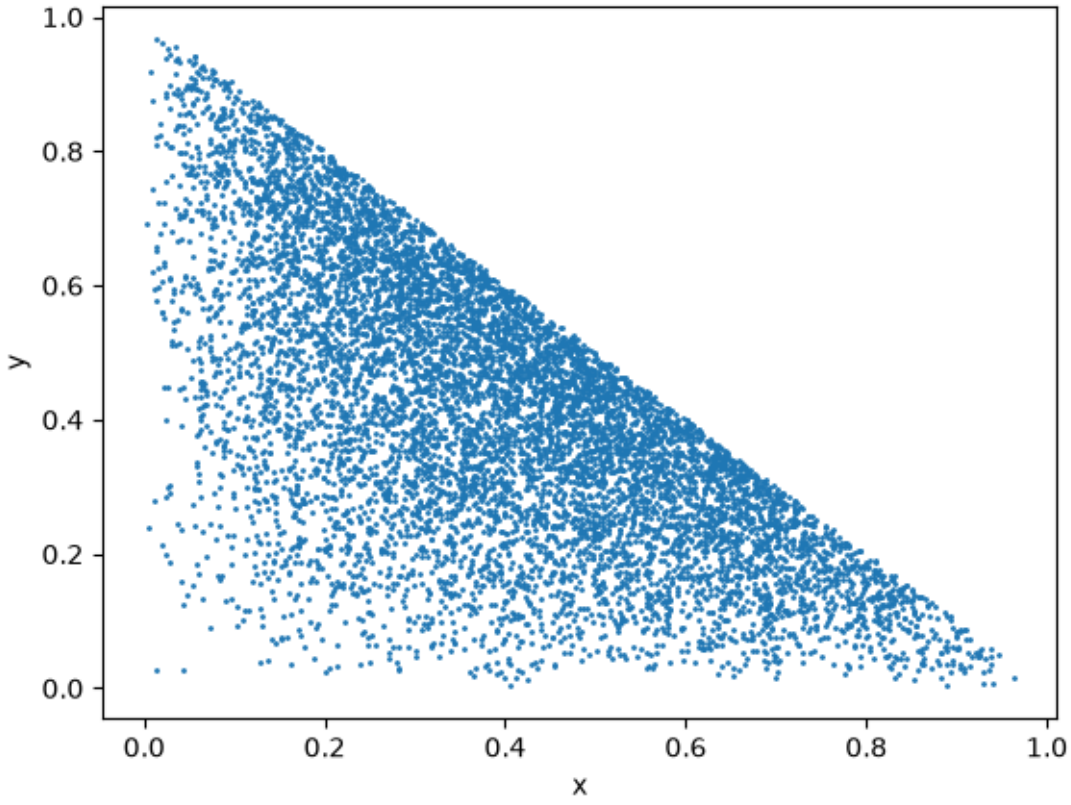
For to variable X og Y må korrelasjonen $\rho_{XY} = \text{corr}(X, Y)$ oppfylle $-1 \leq \rho_{XY} \leq 1$.

Digresjon: For tre variable X, Y, Z må $(\rho_{XY}, \rho_{XZ}, \rho_{YZ}) \in \mathcal{E}_3$ der $\mathcal{E}_3$ er en elliptope definert ved $\mathcal{E}_3 = \{(x, y, z) \in [-1, 1]^3 \mid 1 - x^2 - y^2 - z^2 + 2xyz \geq 0\}$, se https://stats.stackexchange.com/a/557645/77222 som er det samme som at korrelasjonsmatrisen er positiv semi-definitt.

Eksempel, kovarians, sjekk av løsning ved å simulere Anta at X og Y har tetthet $f_{X,Y}(x, y) = 24xy$ for $x > 0, y > 0, x + y < 1$. Fant $\text{corr}(X, Y) = -2/3$ (på tavlen). Sjekk ved å simulering (rejection sampling brukt under er ikke pensum):

[123]:
```
n = 50000

# Generate uniform random variables and accept with probability proportional to␣
 ↪the target density
x = np.random.uniform(0, 1, n)
y = np.random.uniform(0, 1, n)
u = np.random.uniform(0, 1, n)
accept_prob = (4 * x * y * (x + y < 1))
accept = u < accept_prob
x = x[accept]
```

```
y = y[accept]

# Plot the accepted sample points
plt.scatter(x, y, s=1)
plt.xlabel('x')
plt.ylabel('y')
plt.show()

# Sample statistics vs. theoretical values
print("E[X]:", np.mean(x), "| Theoretical:", 2 / 5)
print("Var(X):", np.var(x, ddof=1), "| Theoretical:", 1 / 25)
print("Cov(X,Y):", np.cov(x, y)[0, 1], "| Theoretical:", -2 / 75)
print("Cor(X,Y):", np.corrcoef(x, y)[0, 1], "|Theoretical:", -2/3)

x = np.random.uniform(0,1,20000)
y = np.random.uniform(0,1,20000)
u = np.random.uniform(0,1,20000)
alpha = (1 + x*y)/2 # Aksepter x,y med sannsynlighet alpha proposjonal med␣
 ↪tettheten f_X,Y(x,y)
accept = u < alpha
x = x[accept]
y = y[accept]
len(x) # antall aksepterte samples av totalt n
```



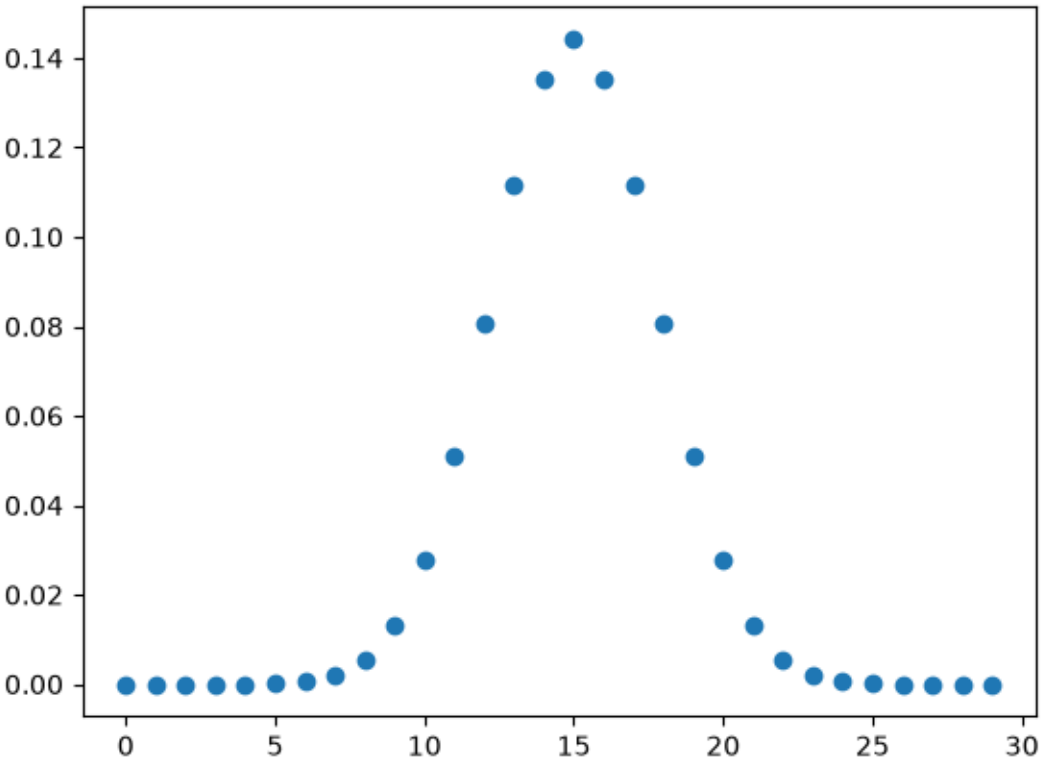
```
E[X]: 0.39519922914989997 | Theoretical: 0.4
Var(X): 0.03971193614866834 | Theoretical: 0.04
Cov(X,Y): -0.026611971199165776 | Theoretical: -0.02666666666666667
Cor(X,Y): -0.6637089833924263 |Theoretical: -0.6666666666666666
```

[123]:
```
12565
```

1.2.1 Bruk av Python som statistisk tabell

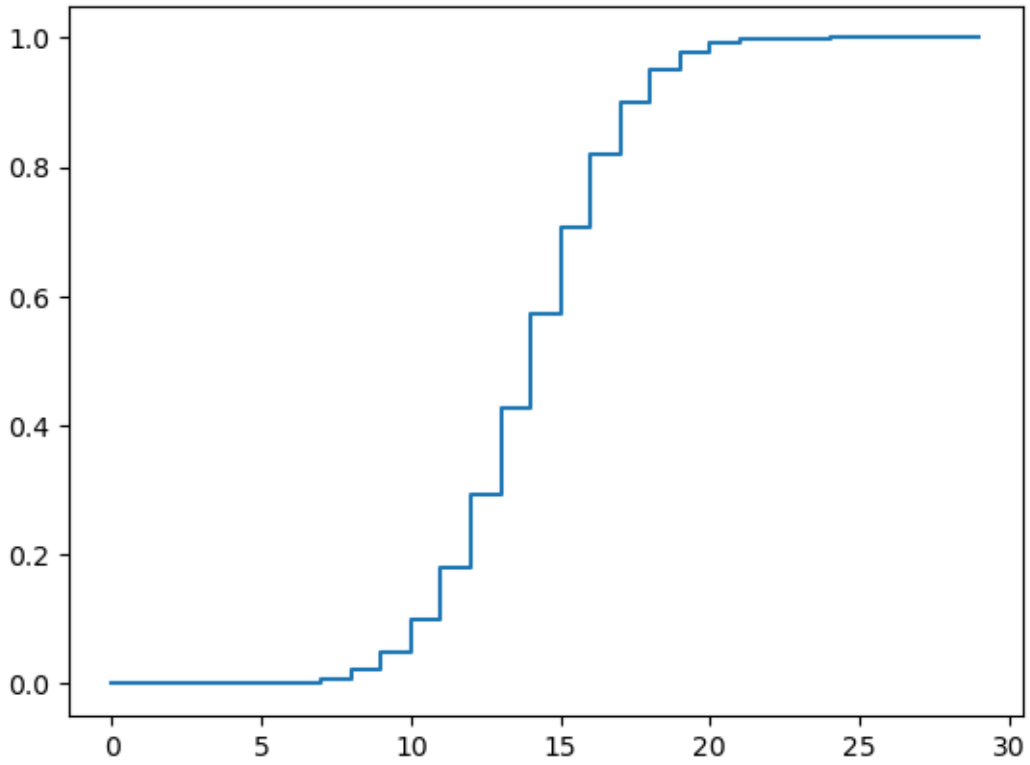
Punktsannsynlighet $f_X(x) = P(X = x)$ når $X \sim \text{bin}(n, p)$ for $n = 30$, $p = 1/2$.

[146]:
```
import scipy.stats as sp
x = np.arange(0,30)
p_X = sp.binom.pmf(x,30,.5)
plt.scatter(x,p_X);
```



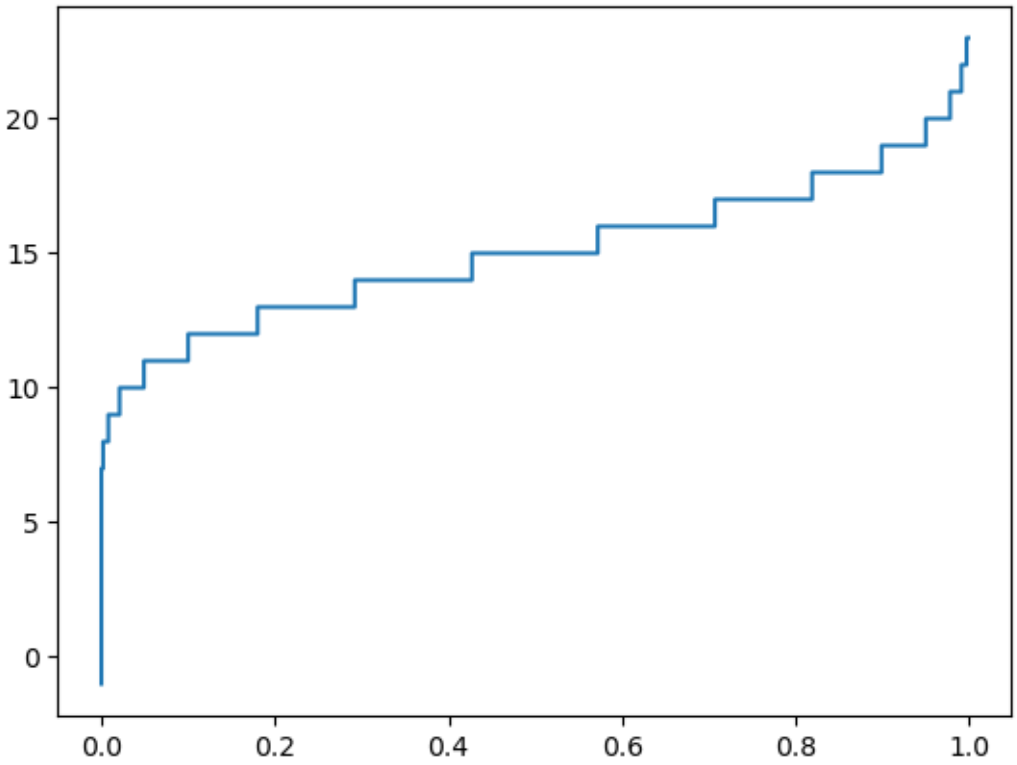
Kumulativ fordeling $F_X(x) = P(X \leq x)$

[58]:
```
F_X = sp.binom.cdf(x,30,.5)
plt.step(x,F_X);
```



Kvantilfunksjonen definert ved $Q(\alpha) = F^{-1}(\alpha) = \min\{x : F_X(x) \geq \alpha\}$ (omtalt som "percent point function" in scipy)

[67]:
```
alpha = np.arange(0,1,.001)
q = sp.binom.ppf(alpha, 30, .5)
plt.step(alpha,q);
```



Simulering

[72]:
```
x = sp.binom.rvs(30,.5,size=100)
x
```

[72]:
```
array([15, 16, 22, 14, 15, 14, 16, 11, 15, 18, 14, 14, 13, 15, 21, 15, 13,
       16, 17, 16, 20, 10, 12, 23, 15, 13, 10, 12, 16, 11, 12, 15, 15, 15,
       16, 15, 12, 17, 12, 16, 15, 17, 16, 16, 17, 16, 14, 15, 19, 17, 16,
       14, 15, 16, 12, 14, 16, 16, 15,  9, 15, 16, 11, 10, 18, 13, 15, 17,
       17, 15,  9, 13, 14, 17, 15, 10, 16, 17, 16, 14, 19, 11, 12, 16, 18,
       14, 16, 12, 12, 13, 14, 15, 15, 17, 11, 14, 12, 14, 15, 17])
```

Eksempel fra folkeopplysningen NRK I hvilken grad er det overraskende at en healer gjetter riktig minst $X = 17$ av $n = 30$ enkeltforsøk dersom sannsynligheten for å gjette riktig er $p = 1/2$ i forsøket vist på https://tv.nrk.no/serie/folkeopplysningen/sesong/2012/episode/KMTE61006110, altså sannsynligheten $P(X \geq 17) = 1 - P(X < 17) = 1 - P(X \leq 16)$ når $X \sim \text{bin}(30, 1/2)$

[138]:
```
1 - sp.binom.cdf(16, 30, .5)
```

[138]:
```
np.float64(0.292332355864346)
```

Hadde resultatet i stedet vært f.eks. $X = 25$ hadde vært vanskelige å forklare dette som et resultat av tilfeldigheter:

[142]:
```
1 - sp.binom.cdf(24, 30, .5)
```

[142]:
```
np.float64(0.00016245711594820023)
```

Sannsynlighetene beregnet over er eksempel på p-verdier som vi kommer tilbake til i uke 43.

Eksempel: multinomisk fordeling Antall 1'ere, 2'ere, …, 6'ere ved kasting av 5 terninger.

[128]:
```
sp.multinomial.rvs(n=5, p=[1/6,1/6,1/6,1/6,1/6,1/6], size=20)
```

[128]:
```
array([[2, 0, 0, 1, 1, 1],
       [0, 0, 2, 0, 1, 2],
       [1, 0, 2, 2, 0, 0],
       [0, 0, 2, 1, 0, 2],
       [1, 0, 0, 0, 1, 3],
       [1, 1, 2, 0, 0, 1],
       [1, 1, 1, 0, 2, 0],
       [2, 1, 0, 1, 1, 0],
       [0, 0, 1, 1, 2, 1],
       [1, 0, 1, 0, 0, 3],
       [0, 1, 1, 1, 2, 0],
       [1, 1, 0, 0, 1, 2],
       [0, 1, 2, 1, 1, 0],
       [0, 0, 3, 0, 1, 1],
       [2, 0, 0, 2, 1, 0],
       [1, 0, 2, 0, 2, 0],
       [0, 2, 1, 0, 0, 2],
       [0, 1, 0, 0, 2, 2],
       [0, 2, 1, 1, 1, 0],
       [0, 0, 2, 1, 1, 1]])
```

1.3 Uke 5

1.3.1 Eksamen August 2026, oppgave 4

Norsk regnesentrals modell for fotballvm bygger på antakelsen om at antall scoringer til hvert lag er Poissonfordelt med parametere bestemt av styrkeforholdet mellom lagene. Anta at lag A scorer $X \sim \text{Pois}(1.2)$ og at lag B scorer $Y \sim \text{Pois}(0.4)$. Sannsynlighetet for resultatet 2-1, $P(X = 2 \cap Y = 1)$:

[7]:
```
import scipy.stats as sp
import numpy as np
sp.poisson.pmf(2, 1.2)*sp.poisson.pmf(1,0.4)
```

[7]:
```
np.float64(0.05814619718246076)
```

Anslag av $P(\text{Seier til lag A}) = P(X > Y)$ ved å simulere:

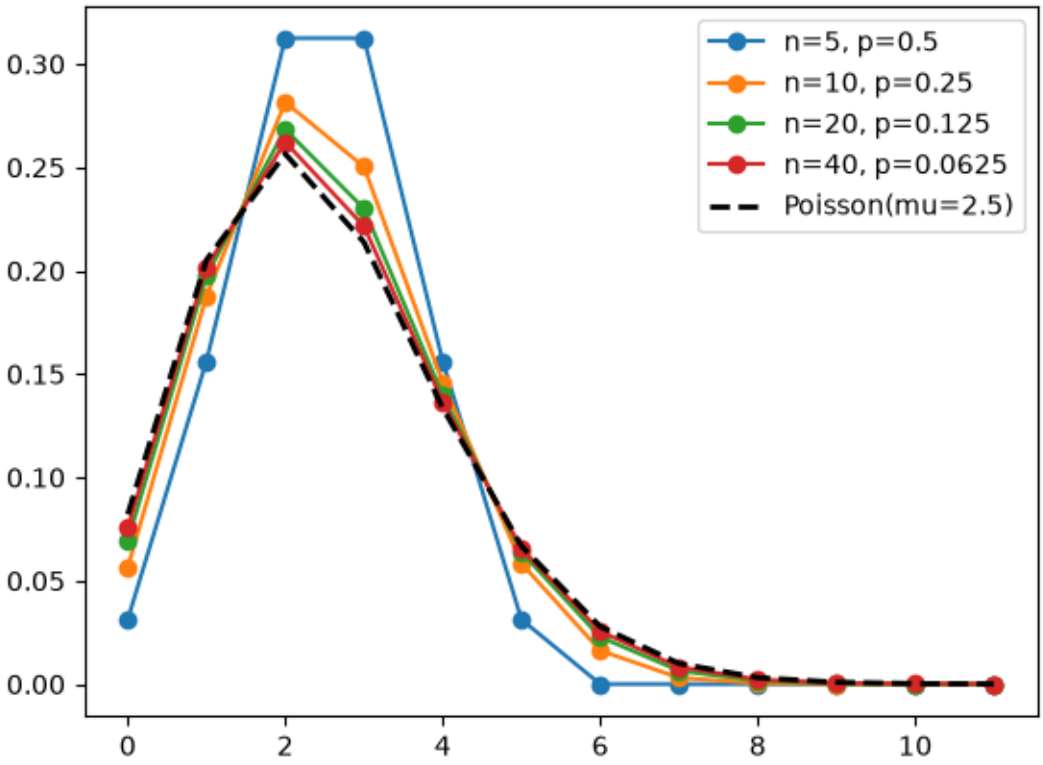
[42]:
```
x = sp.poisson.rvs(1.2, size=10000)
y = sp.poisson.rvs(0.4, size=10000)
np.sum(x>y)
np.sum(x>y)/10000
```

[42]:
```
np.float64(0.5688)
```

1.3.2 Poissonfordeling som grense av binomisk

Lar vi $p \to 0$ org $n \to \infty$ samtidig som $\mu = np$ holdes konstant går punktsannsynlighetene for en binomisk mot punktsannsynlighetene til en Poissonfordeling med parameter $\mu = np$. Illustrasjon:

[21]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import binom, poisson

k = np.arange(12)

for n, p in zip([5, 10, 20, 40], [0.5, 0.25, 0.125, 0.0625]):
    plt.plot(k, binom.pmf(k, n, p), "o-", label=f"n={n}, p={p}")

plt.plot(k, poisson.pmf(k, 2.5), "k--", lw=2, label="Poisson(mu=2.5)")
plt.legend()
plt.show()
```



1.3.3 Normalfordeling

Eksempel, maksimalt antall personer i heis Antar at $X_1, X_2, \ldots, X_n$ er uavhengige og $N(\mu, \sigma^2)$ dere $\mu = 85$kg og $\sigma = 15$kg. Hvor mange personer kan gå inn i heisen vi vi krever at $P(W = \sum_{i=1}^{n} X_i < k) \leq \alpha = 0.01$, $k = 1000$kg?

Fant at n kan være høyst lik 10 (på tavlen).

Sanitycheck i Python:

[41]:
```
for n in range(8,15):
  print("n =",n,", P(W > 1000) =",1 - sp.norm.cdf(1000, 85*n, 15*np.sqrt(n)))
```

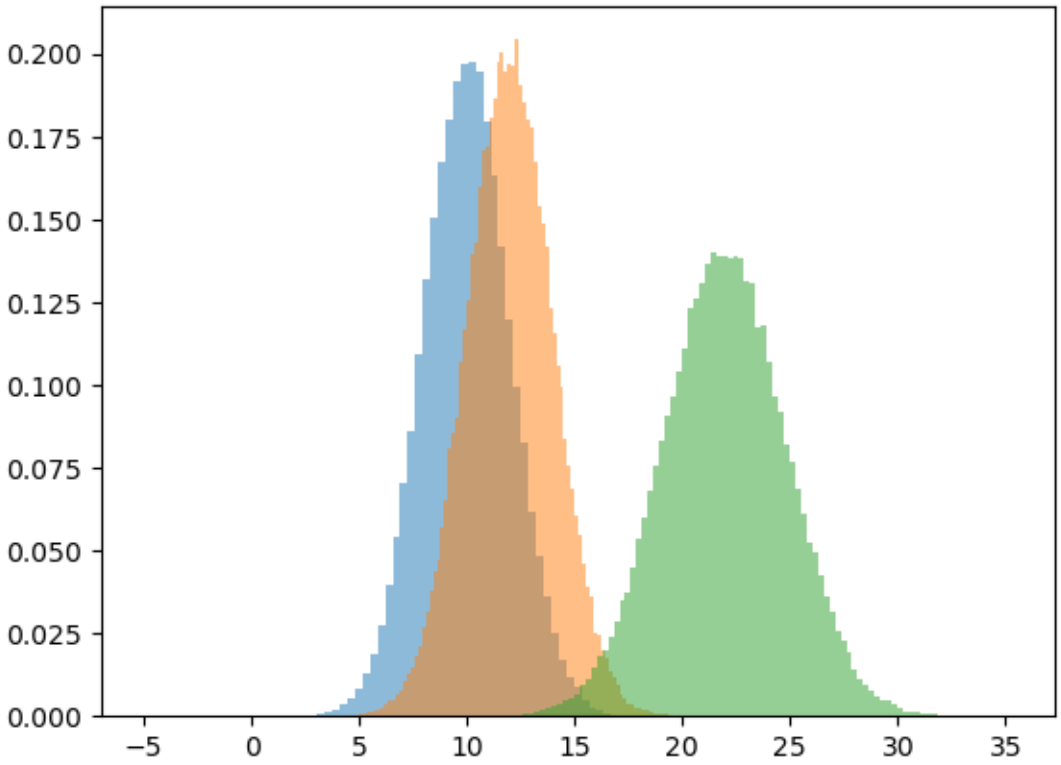
```
n = 8 , P(W > 1000) = 2.3092638912203256e-14
n = 9 , P(W > 1000) = 8.839431620000227e-08
n = 10 , P(W > 1000) = 0.0007827011290012509
n = 11 , P(W > 1000) = 0.09568293639285841
n = 12 , P(W > 1000) = 0.6498443135315549
n = 13 , P(W > 1000) = 0.9738981823293427
n = 14 , P(W > 1000) = 0.9996445092340235
```

1.4 Uke 7

1.4.1 Momentgenererende funksjoner

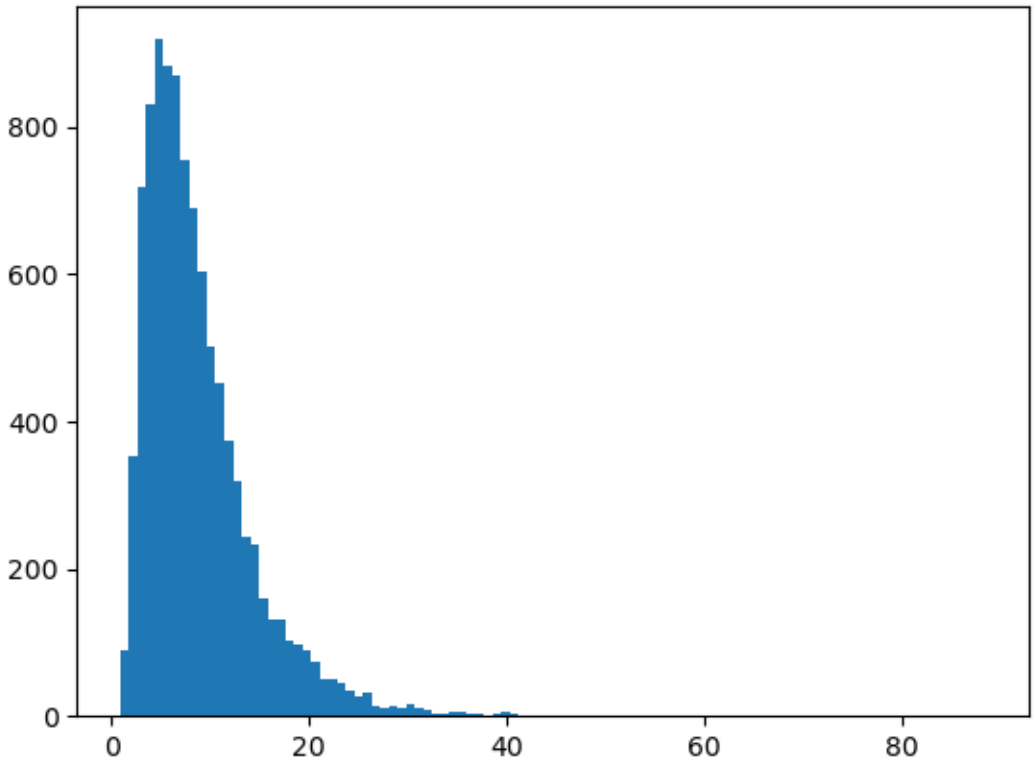
Hvilken fordeling har $X + Y$ hvis X og Y er uavhengig normalfordelte variable?

[403]:
```
x = np.random.normal(10, 2, 100000)
y = np.random.normal(12, 2, 100000) # Normalfordelt Y via inversjonsmetode

plt.hist(x, range=(-5,30), alpha=0.5, bins=100, density=True);
plt.hist(y, alpha=0.5, bins=100, density=True);
plt.hist(x + y, alpha=0.5, bins=100, density=True);
```



Eksempel: Momenter til log-normalfordelingen Sjekk om analytisk løsning $EY = e^{\mu+\sigma^2/2}$ er riktig:

[277]:
```
mu = 2
sigma = .6
x = np.random.normal(mu, sigma, 10000)
y = np.exp(x)
plt.hist(y, bins=100);
```



[250]:
```
# Estimert forventning basert på simulerte verdier
np.mean(y)
```

[250]: 8.857606551633221

[260]:
```
# Teoretisk forventingsverdi
np.exp(mu + sigma*sigma/2)
```

[260]: 8.846306258720883

[254]:
```
# Estimert median
np.median(y)
```

[254]: 7.469786657163913

[262]:
```
# Teoretisk median (50%-kvantilen)
np.exp(2)
```

[262]: 7.38905609893065

1.4.2 Sentralgrenseteoremet

La $X_1, X_2, ...$ være uavhengig identisk fordelt stokastiske variable med forventning μ og varians σ^2 og la $\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$. Vi har da at

$$Z_n = \frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} \xrightarrow{d} N(0,1),$$

når $n \to \infty$, vi sier at Z_n konvergerer i fordeling mot standardnormalfordelingen når n går mot uendelig. Det betyr (definisjon av konvergens i fordeling) at

$$\lim_{n\to\infty} P(Z_n \leq z) = \Phi(z)$$

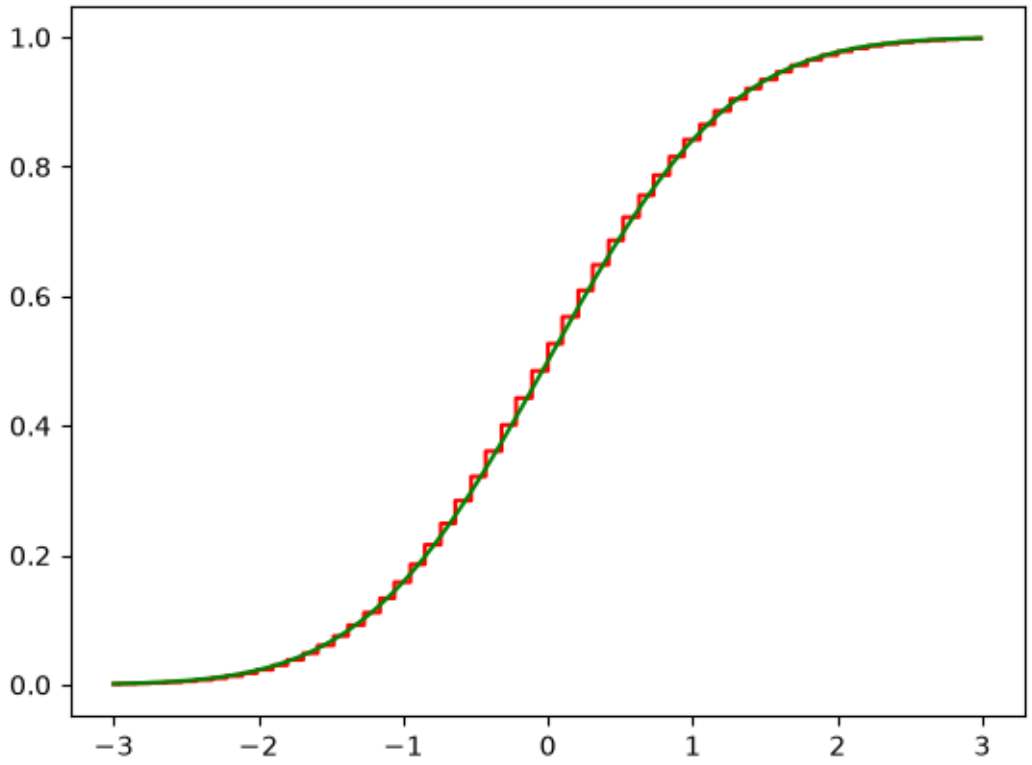
for alle $z \in \mathbb{R}$ der $\Phi(z)$ er kontinuerlig.

Eksempel: Binomisk fordeling Hvis $X_1, X_2, ...$ er iid Bernoulli med parameter p er $Y_n = \sum_{i=1}^{n} X_i \sim \text{bin}(n,p)$. Lar vi $F_{n,p}(y) = P(Y_n \leq y)$ betegne kumulativ fordeling til Y_n, er eksakt kumulativ fordeling til Z_n gitt ved

$$\begin{aligned}
F_{Z_n}(z) &= P(Z_n \leq z) && (1)\\
&= P(\frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} \leq z) && (2)\\
&= P(\frac{Y_n - np}{\sqrt{np(1-p)}} \leq z) && (3)\\
&= P(Y_n \leq np + \sqrt{np(1-p)}z) && (4)\\
&= F_{n,p}(np + \sqrt{np(1-p)}z) && (5)
\end{aligned}$$

Plotter vi dette for økende verdier av n ser vi at $F_{Z_n}(z)$ (rød kurve i figur) ser ut til konvergere mot $\Phi(z)$ (grønn kurve i figur) i alle z slik sentralgrenseteoremet sier.

[23]:
```
import scipy.stats as sp
import matplotlib.pyplot as plt
import numpy as np
z = np.arange(-3, 3, .01)
n = 1000
p = .1
F_Z = sp.binom.cdf(n*p + np.sqrt(n*p*(1-p))*z, n, p)
plt.step(z,F_Z, 'r-');
Phi = sp.norm.cdf(z)
plt.plot(z, Phi, 'g-');
```



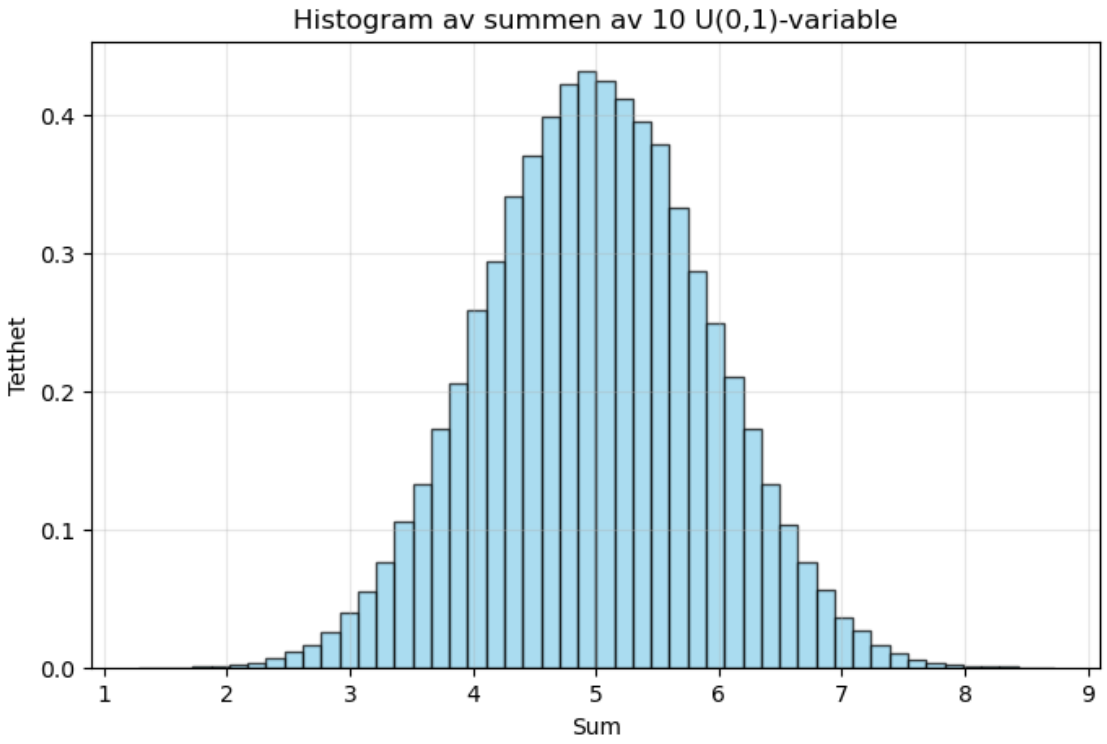
Eksempel: Sum av n uniformt fordelte variable Anta at $U_1, U_2, \ldots, U_n$ er iid uniformt fordelt å intervallet fra 0 til 1. Da er $E(U_i) = 1/2$ og $\operatorname{Var} U_i = 1/12$. Fordelingen til

$$\frac{\bar{U} - 1/2}{\sqrt{1/12n}}$$

kan simuleres som følger

[460]:
```
nsim = 100000 # Antall simuleringer
n = 10

# Simuler summen
rng = np.random.default_rng(seed=42)
data = rng.uniform(0, 1, size=(nsim, n)).sum(axis=1)

# Lag histogram
plt.figure(figsize=(8, 5))
plt.hist(data, bins=50, density=True, color="skyblue", edgecolor="black",␣
↪alpha=0.7)
plt.title(f"Histogram av summen av {n} U(0,1)-variable")
plt.xlabel("Sum")
plt.ylabel("Tetthet")
plt.grid(alpha=0.3)
```

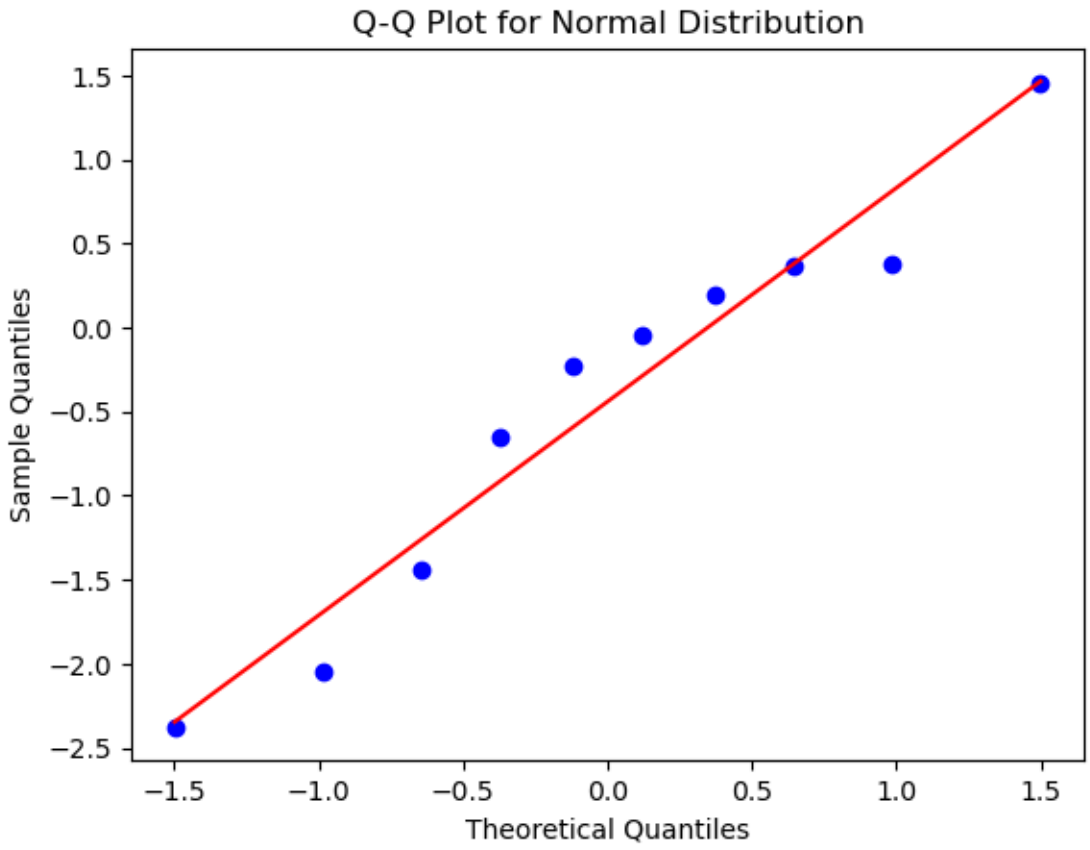
```
plt.show()
```



1.4.3 Uke 8

Normal kvantil-kvantil plot

[76]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

# Generate a sample of 1000 points from a normal distribution
sample = np.random.normal(0, 1, 10)

# Create a Q-Q plot comparing the sample to a theoretical normal distribution
stats.probplot(sample, dist="norm", plot=plt)

# Add title and labels
plt.title("Q-Q Plot for Normal Distribution")
plt.xlabel("Theoretical Quantiles")
plt.ylabel("Sample Quantiles");
```



1.4.4 Uke 9

Konfidensintervall Konfidensintervall for θ basert på tilfeldig utvalg $X_1, X_2, \ldots, X_n \sim \text{Unif}(0, \theta)$ med utgangspunkt i henholdsvis pivotalen $X_{(n)}/\theta$ der $X_{(n)} = \max(X_1, \ldots, X_n)$ og pivotal basert på normaltilnærming av $\bar{X}$ (utledet på forelesning). Sjekk av om dekning er korrekt og sammenligning av gjennomsnittlig intervallengde ved simulering:

[131]:
```
import numpy as np
import scipy.stats as sp

# --- Settings ---
n = 10          # sample size
R = 100000      # number of simulated samples
theta = 1       # true theta (change if you like; results scale accordingly)
alpha = 0.05    # for a 95% CI

# Quantiles for the pivotal Y = X_(n)/theta (Y ~ with CDF y^n on [0,1])
q_lower = (alpha/2)**(1/n)
q_upper = (1 - alpha/2)**(1/n)
```

```
rng = np.random.default_rng(0)  # reproducibility

# Simulate R samples of size n from Uniform(0, theta_true)
samples = rng.uniform(0, theta, size=(R, n))

# Sample maxima
xmax = samples.max(axis=1)

# Exact (1 - alpha) CI for theta using the pivotal x_max/theta
ci_lower = xmax / q_upper
ci_upper = xmax / q_lower

# Empirical coverage: how often the true theta is inside the CI
coverage = np.mean((ci_lower <= theta) & (theta <= ci_upper))

# Mean length
ci_length = np.mean(ci_upper - ci_lower)

# Show results
print(f"n={n}, R={R}, theta={theta}, alpha={alpha}")
print(f"q_lower={(q_lower):.6f}, q_upper={(q_upper):.6f}")
print(f"Empirical coverage (should be close to {1-alpha:.6f}): {coverage:.6f}")
print(f"Mean length = {ci_length:.3f}")

# Peek at the first few intervals
for i in range(5):
    print(f"Sample {i+1}: X_max={xmax[i]:.4f}, CI=[{ci_lower[i]:.4f},␣
 ↪{ci_upper[i]:.4f}]")
```

```
n=10, R=100000, theta=1, alpha=0.05
q_lower=0.691503, q_upper=0.997471
Empirical coverage (should be close to 0.950000): 0.949670
Mean length = 0.403
Sample 1: X_max=0.9351, CI=[0.9374, 1.3522]
Sample 2: X_max=0.8632, CI=[0.8654, 1.2483]
Sample 3: X_max=0.9972, CI=[0.9997, 1.4421]
Sample 4: X_max=0.9340, CI=[0.9364, 1.3507]
Sample 5: X_max=0.8903, CI=[0.8925, 1.2874]
```

[133]:
```
# Approximate (1 - alpha) CI for theta using normal approximation of sample mean
xbar = samples.mean(axis=1)
z = sp.t.ppf(alpha/2, df = n - 1)
ci_lower = 2*xbar*(1 + z/np.sqrt(3*n))
ci_upper = 2*xbar*(1 - z/np.sqrt(3*n))

# Empirical coverage: how often the true theta is inside the CI
coverage = np.mean((ci_lower <= theta) & (theta <= ci_upper))
```

```
# Mean length
ci_length = np.mean(ci_upper - ci_lower)

# Show results
print(f"n={n}, R={R}, theta={theta}, alpha={alpha}")
print(f"a={(a):.6f}, b={(b):.6f}")
print(f"Empirical coverage (should be close to {1-alpha:.6f}): {coverage:.6f}")
print(f"Mean length = {ci_length:.3f}")

# Peek at the first few intervals
for i in range(5):
    print(f"Sample {i+1}: X=bar={xbar[i]:.4f}, CI=[{ci_lower[i]:.4f},␣
 ↪{ci_upper[i]:.4f}]")
```

```
n=10, R=100000, theta=1, alpha=0.05
a=0.691503, b=0.997471
Empirical coverage (should be close to 0.950000): 0.945330
Mean length = 0.826
Sample 1: X=bar=0.5505, CI=[0.6463, 1.5558]
Sample 2: X=bar=0.4742, CI=[0.5567, 1.3401]
Sample 3: X=bar=0.5784, CI=[0.6790, 1.6344]
Sample 4: X=bar=0.5437, CI=[0.6383, 1.5364]
Sample 5: X=bar=0.4875, CI=[0.5723, 1.3775]
```

1.4.5 Uke 12

Regresjon, prediksjonsintervall, log-log-lineære modeller, andre nullhypoteser enn $\beta_1 = 0$ Eksempel: Sammenheng mellom hjernestørrelse (i kg) og kroppsvekt (i kg) hos landlevende pattedyr:

[482]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

df = pd.read_feather("df.mammals")
print(df.to_string())
```

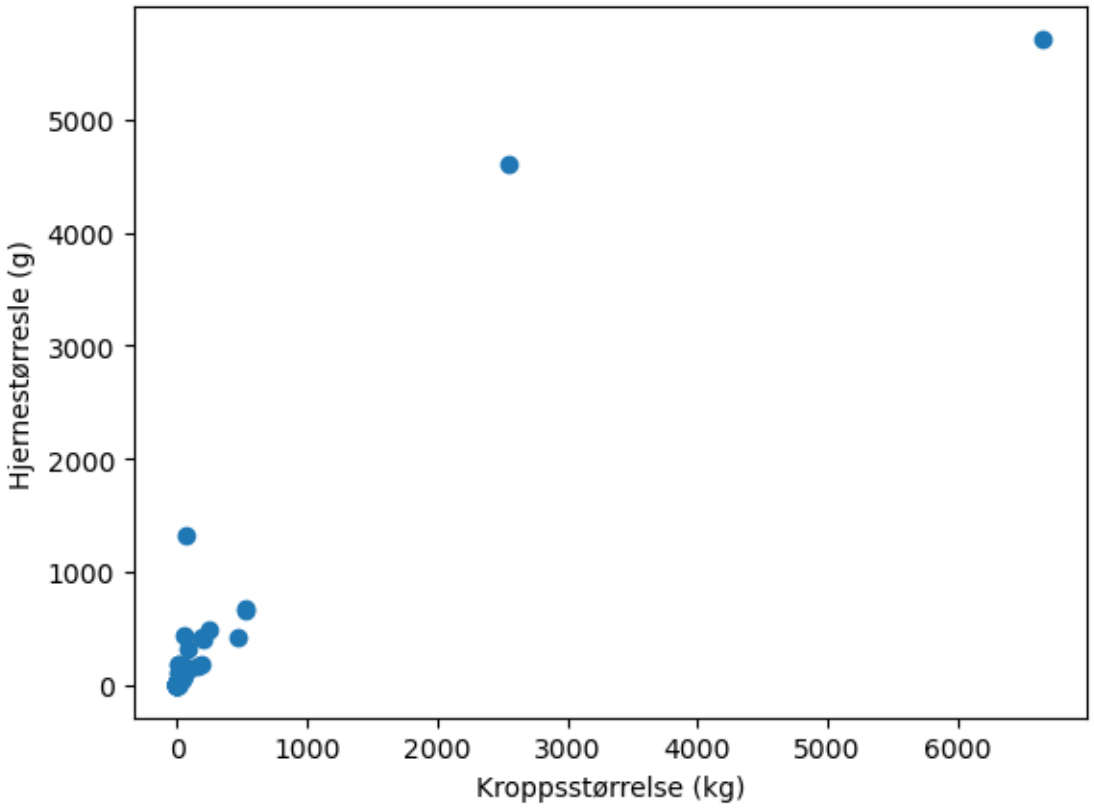
```
                   species      body    brain
0                Arctic fox     3.385    44.50
1                Owl monkey     0.480    15.50
2           Mountain beaver     1.350     8.10
3                       Cow   465.000   423.00
4                Grey wolf    36.330   119.50
5                     Goat    27.660   115.00
6                 Roe deer    14.830    98.20
7                Guinea pig     1.040     5.50
8                    Verbet     4.190    58.00
```

9	Chinchilla	0.425	6.40
10	Ground squirrel	0.101	4.00
11	Arctic ground squirrel	0.920	5.70
12	African giant pouched rat	1.000	6.60
13	Lesser short-tailed shrew	0.005	0.14
14	Star-nosed mole	0.060	1.00
15	Nine-banded armadillo	3.500	10.80
16	Tree hyrax	2.000	12.30
17	N.A. opossum	1.700	6.30
18	Asian elephant	2547.000	4603.00
19	Big brown bat	0.023	0.30
20	Donkey	187.100	419.00
21	Horse	521.000	655.00
22	European hedgehog	0.785	3.50
23	Patas monkey	10.000	115.00
24	Cat	3.300	25.60
25	Galago	0.200	5.00
26	Genet	1.410	17.50
27	Giraffe	529.000	680.00
28	Gorilla	207.000	406.00
29	Grey seal	85.000	325.00
30	Rock hyrax-a	0.750	12.30
31	Human	62.000	1320.00
32	African elephant	6654.000	5712.00
33	Water opossum	3.500	3.90
34	Rhesus monkey	6.800	179.00
35	Kangaroo	35.000	56.00
36	Yellow-bellied marmot	4.050	17.00
37	Golden hamster	0.120	1.00
38	Mouse	0.023	0.40
39	Little brown bat	0.010	0.25
40	Slow loris	1.400	12.50
41	Okapi	250.000	490.00
42	Rabbit	2.500	12.10
43	Sheep	55.500	175.00
44	Jaguar	100.000	157.00
45	Chimpanzee	52.160	440.00
46	Baboon	10.550	179.50
47	Desert hedgehog	0.550	2.40
48	Giant armadillo	60.000	81.00
49	Rock hyrax-b	3.600	21.00
50	Raccoon	4.288	39.20
51	Rat	0.280	1.90
52	E. American mole	0.075	1.20
53	Mole rat	0.122	3.00
54	Musk shrew	0.048	0.33
55	Pig	192.000	180.00
56	Echidna	3.000	25.00

```
57            Brazilian tapir   160.000   169.00
58                     Tenrec     0.900     2.60
59                  Phalanger     1.620    11.40
60                 Tree shrew     0.104     2.50
61                    Red fox     4.235    50.40
```

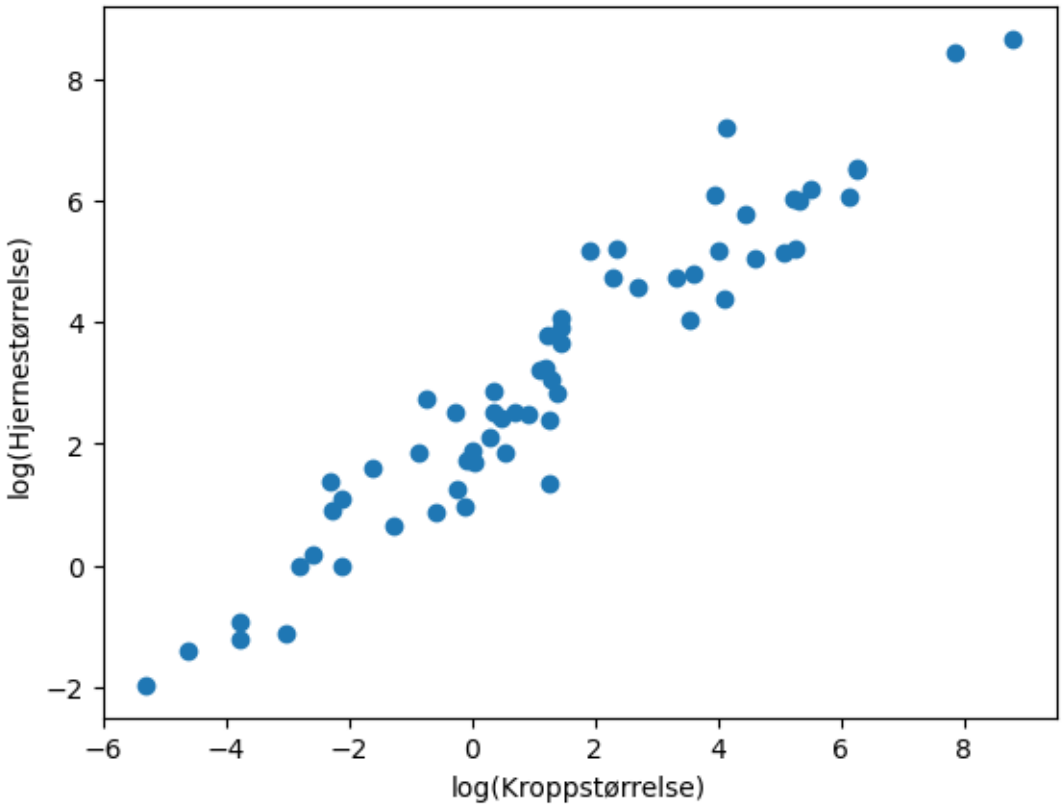
[484]:
```
plt.scatter(df["body"], df["brain"]);
plt.xlabel("Kroppsstørrelse (kg)")
plt.ylabel("Hjernestørresle (g)")
```

[484]: Text(0, 0.5, 'Hjernestørresle (g)')



[490]:
```
df["x"] = np.log(df["body"])
df["y"] = np.log(df["brain"])
plt.scatter(df["x"], df["y"])
plt.xlabel("log(Kroppstørrelse)")
plt.ylabel("log(Hjernestørrelse)")
```

[490]: Text(0, 0.5, 'log(Hjernestørrelse)')



[492]:
```
import statsmodels.api as sm
import statsmodels.formula.api as smf
model = smf.ols("y ~ x", data=df.query('species!="Human"')).fit()
print(model.summary())
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.928
Model:	OLS	Adj. R-squared:	0.927
Method:	Least Squares	F-statistic:	763.0
Date:	Mon, 10 Nov 2025	Prob (F-statistic):	1.91e-35
Time:	11:58:18	Log-Likelihood:	-59.365
No. Observations:	61	AIC:	122.7
Df Residuals:	59	BIC:	127.0
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	2.1150	0.090	23.421	0.000	1.934	2.296
x	0.7423	0.027	27.622	0.000	0.689	0.796

```
==============================================================================
Omnibus:                        0.526   Durbin-Watson:                   2.083
Prob(Omnibus):                  0.769   Jarque-Bera (JB):                0.356
Skew:                           0.186   Prob(JB):                        0.837
Kurtosis:                       2.965   Cond. No.                         3.69
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[512]:
```
# Extract estimated parameters from fitted model object and compute summary␣
 ↪stats
beta0hat = model.params["Intercept"]
beta1hat = model.params["x"]
sigmahat = np.sqrt(model.mse_resid)
xbar = np.average(df["x"])
SS_x = np.sum((df["x"] - xbar)**2)

# Predict log brain size for a mammal with body size of 62 kgs (equal to␣
 ↪average body size for humans)
x0 = np.log(62)
```

[512]:
```
62.00000000000001
```

[518]:
```
# Predicted log brain size for mammal with body size x0
y0hat = beta0hat + beta1hat*x0
y0hat
np.exp(y0hat)
```

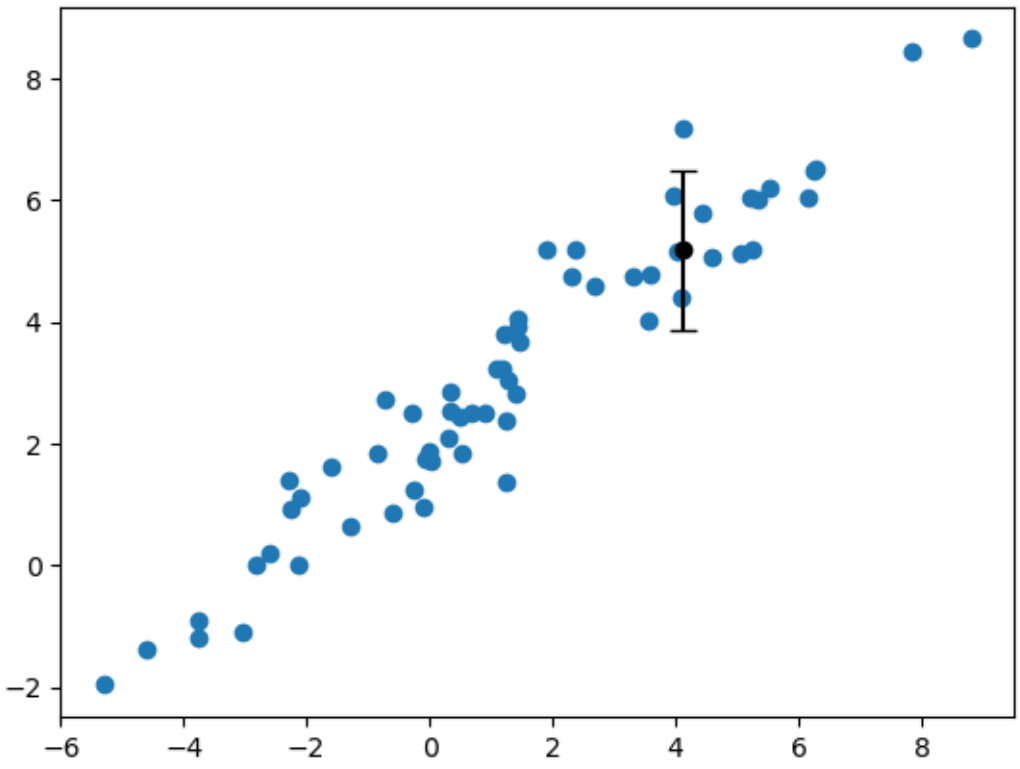
[518]:
```
177.41290743749977
```

[506]:
```
# Compute prediction interval for log grain size
z = stats.t.ppf(0.975, df=model.df_resid)
n = 61
delta = z*sigmahat*np.sqrt(1 + 1/n + (x0 - xbar)**2/SS_x)
ci = np.array([y0hat - delta, y0hat + delta])
ci
```

[506]:
```
array([3.85655275, 6.5004069 ])
```

[498]:
```
plt.scatter(df["x"], df["y"])
plt.errorbar(
    x0, y0hat,
    yerr = [[delta], [delta]],
    fmt="o", color="black", ecolor="black", elinewidth=1.5, capsize=5,
    label="95% PI at x0"
)
```

[498]: <ErrorbarContainer object of 3 artists>



[520]:
```
# Backtransform the prediction interval from log scale. How does this compare
↪to average brain size of humans of 1320grams?
ciexp = np.exp(ci)
ciexp
```

[520]: array([47.30200822, 665.4123347])

[ ]: