TP1 — Echantillonnage loi de Poisson (corrige)

Source : TP1/TP1_corrige.ipynb — PDF compile (code execute).

Sampling from a Poisson distribution

In [1] :

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

Complete the following code to sample from a Poisson distribution using the direct inversion method. You should only use uniform random variables on $[0,1]$.

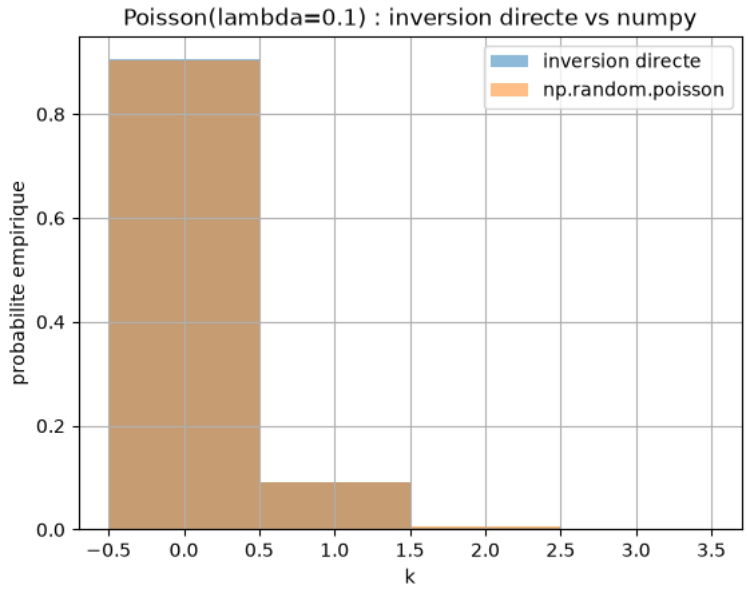
In [2] :

```
def poisson_sampling_direct(llambda, n):
"""Echantillonne n v.a. Poisson(llambda) par inversion directe (Knuth).
N'utilise que des Uniformes[0,1]."""
out = np.empty(n, dtype=int)
L = np.exp(-llambda)
for i in range(n):
k = 0
p = 1.0
while True:
k += 1
p *= np.random.rand()
if p <= L:
break
out[i] = k - 1
return out
```

Test your method comparing the emperical histogram obtained from the method you have implemented and the one obtained using the function np.random.poisson. You will consider $\lambda \in \{0.1,1,10\}$.

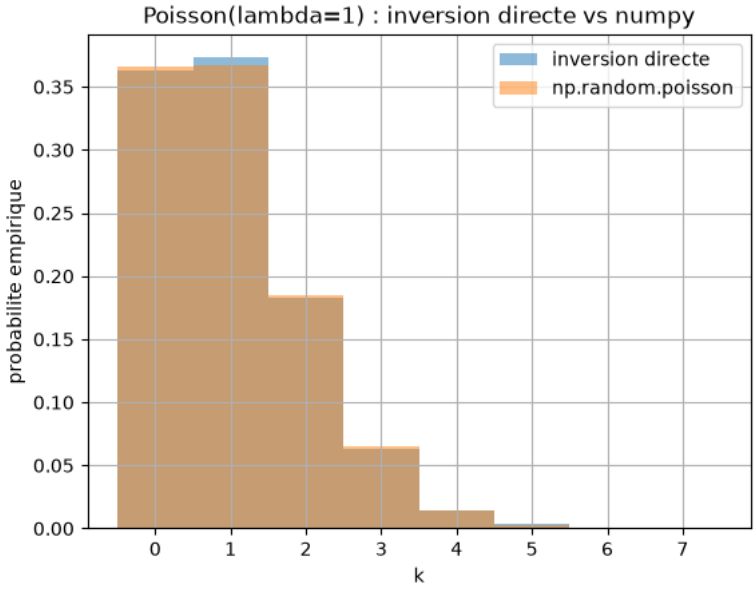
In [3] :

```
import time
for lam in [0.1, 1, 10]:
s = poisson_sampling_direct(lam, 20000)
ref = np.random.poisson(lam, 20000)
kmax = max(int(s.max()), int(ref.max())) + 1
bins = np.arange(-0.5, kmax + 0.5, 1)
plt.figure()
plt.hist(s, bins=bins, density=True, alpha=0.5, label="inversion directe")
plt.hist(ref, bins=bins, density=True, alpha=0.5, label="np.random.poisson")
plt.title(f"Poisson(lambda={lam}) : inversion directe vs numpy")
plt.xlabel("k"); plt.ylabel("probabilite empirique")
plt.legend(); plt.grid(True); plt.show()
print(f"lambda={lam}: moyenne empirique={s.mean():.3f} (theo={lam}), var={s.var():.3f}
(theo={lam})")
```



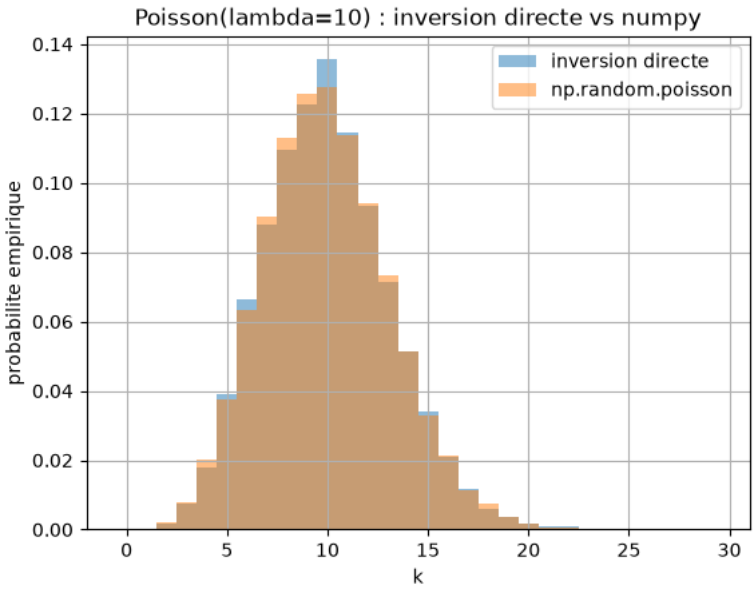

```
<Figure size 640x480 with 1 Axes>
```

```
lambda=0.1: moyenne empirique=0.100 (theo=0.1), var=0.099 (theo=0.1)
```




```
<Figure size 640x480 with 1 Axes>
```

```
lambda=1: moyenne empirique=1.007 (theo=1), var=1.000 (theo=1)
```




```
<Figure size 640x480 with 1 Axes>
```

```
lambda=10: moyenne empirique=9.987 (theo=10), var=9.889 (theo=10)
```

Consider the logistic distribution with parameter $\alpha \in \mathbb{R}$ and $\beta >0$ defined trough its distribution function: $$ F(x) = (1+\exp{-(x-\alpha)/\beta})^{-1} \, , \, x \in\mathbb{R}\, . $$

In the following we aim to sample from a Poisson distribution using the accept-reject method with instrumental distribution the logistic distribution with parameter $\alpha = \lambda$ and $\beta = \sqrt{3\lambda}/\pi$. You should only use uniform random variables on $[0,1]$.

We first implement a method to sample from the logistic distribution with parameter α and β restricted to $[-1/2,\infty)$.

In [4] :

```
def logistic_sampling(alpha, beta, n):
"""Echantillonne la loi logistique(alpha,beta) par inversion.
F(x)=1/(1+exp(-(x-alpha)/beta)), F^{-1}(u)=alpha+beta*log(u/(1-u))."""
U = np.random.rand(n)
# eviter 0 et 1 exactement
U = np.clip(U, 1e-16, 1 - 1e-16)
return alpha + beta * np.log(U / (1.0 - U))

def logistic_sampling_half(alpha, beta, n):
"""Echantillonne la logistique(alpha,beta) conditionnee a X >= -1/2.
Inversion tronquee : X = F^{-1}(F(-1/2)+U*(1-F(-1/2)))."""
F_low = 1.0 / (1.0 + np.exp(-(-0.5 - alpha) / beta))
U = np.random.rand(n)
V = F_low + U * (1.0 - F_low)
V = np.clip(V, 1e-16, 1 - 1e-16)
return alpha + beta * np.log(V / (1.0 - V))
```

We now aim to implement a function which gives the probability $\mathbb{P}(N=k)$ where $N = \lfloor X +1/2 \rfloor$ where X is distributed according to the logistic distribution with parameter α and β restricted to $[-1/2,\infty)$.

Question: compute for any $k\in\mathbb{N}$, $\mathbb{P}(N=k)$.

Make a function which returns $\log \mathbb{P}(N=k)$. You may use the function provided logexp.

In [5] :

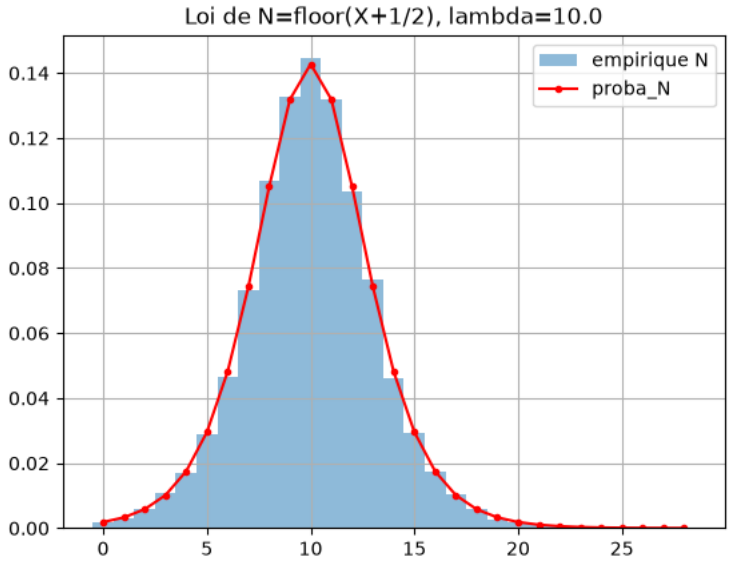
```
def logexp(x):
x1 = (x<= 4)*np.log(1+np.exp(x))
x2 = (x > 4)*(x+np.log(1+np.exp(-x)))
x = x1+x2
return x

# P(N=k) = (F(k+1/2)-F(k-1/2))/(1-F(-1/2)), N=floor(X+1/2), X~logistique tronquee
# Demonstration (reponse math) : pour k>=1, {N=k}={k-1/2<=X<k+1/2} ; pour k=0,
{N=0}={-1/2<=X<1/2}.
# En divisant par P(X>=-1/2)=1-F(-1/2) on obtient la formule ci-dessus.
def proba_N(k, alpha, beta):
"""Retourne log P(N=k). k scalaire ou array."""
k = np.asarray(k, dtype=float)
F_up = 1.0 / (1.0 + np.exp(-(k + 0.5 - alpha) / beta))
F_lo = 1.0 / (1.0 + np.exp(-(k - 0.5 - alpha) / beta))
F_m = 1.0 / (1.0 + np.exp(-(-0.5 - alpha) / beta))
pr = (F_up - F_lo) / (1.0 - F_m)
pr = np.maximum(pr, 1e-300)
return np.log(pr)
```

Verify that your code is correct by comparing the histogram you obtain from proba_N with an empirical histogram obtained from sampling from N

In [6] :

```
# Verification : histogramme empirique de N vs proba_N
lam = 10.0
alpha, beta = lam, np.sqrt(3 * lam) / np.pi
n = 50000
X = logistic_sampling_half(alpha, beta, n)
N = np.floor(X + 0.5).astype(int)
kmax = int(N.max())
ks = np.arange(0, kmax + 1)
logp = proba_N(ks, alpha, beta)
plt.figure()
plt.hist(N, bins=np.arange(-0.5, kmax + 1.5, 1), density=True, alpha=0.5, label="empirique
N")
plt.plot(ks, np.exp(logp), "ro-", ms=3, label="proba_N")
plt.title(f"Loi de N=floor(X+1/2), lambda={lam}")
plt.legend(); plt.grid(True); plt.show()
```




```
<Figure size 640x480 with 1 Axes>
```

We recall that for any λ, it was stated in the course that $$ \sup_n \lambda^n/(\mathbb{P}(N = n) e^{\lambda} n!) < 0.1\times \lambda /(0.767 - 3.36/\lambda), $$ with parameter $\alpha = \lambda$ and $\beta = \sqrt{3\lambda}/\pi$.

Verify numerically that this is true for $\lambda \in \{10,100\}$.

In [7] :

```
from scipy.special import gammaln
def poisson_logpmf(k, lam):
return k * np.log(lam) - lam - gammaln(k + 1)

for lam in [10, 100]:
alpha, beta = lam, np.sqrt(3 * lam) / np.pi
ks = np.arange(0, int(3 * lam) + 50)
log_g = proba_N(ks, alpha, beta)
log_f = poisson_logpmf(ks, lam)
ratio = np.exp(log_f - log_g)
mask = np.exp(log_g) > 1e-15
sup_emp = ratio[mask].max()
borne = 0.1 * lam / (0.767 - 3.36 / lam)
print(f"lambda={lam}: sup empirique={sup_emp:.3f}, borne cours={borne:.3f}, ok={sup_emp <
borne}")
```

```
lambda=10: sup empirique=1.309, borne cours=2.320, ok=True
lambda=100: sup empirique=1.255, borne cours=13.635, ok=True
```

From the previous functions that we have implemented, code a function that sample from the Poisson distribution using the accept-reject method based on the logistic distribution restricted to $[-1/2,\infty]$ with parameter $\alpha = \lambda$ and $\beta = \sqrt{3\lambda}/\pi$.

In [8] :

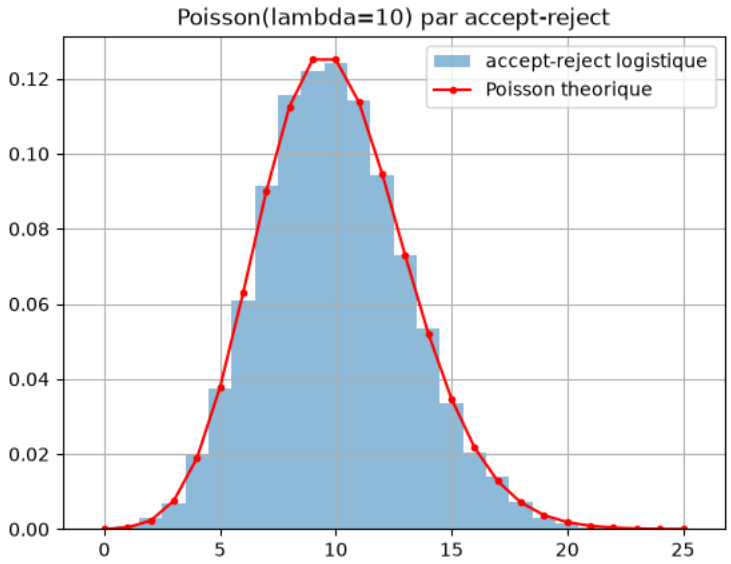
```
from scipy.special import gammaln
def poisson_logpmf(k, lam):
return k * np.log(lam) - lam - gammaln(k + 1)

def poisson_sampling_logistic(llambda, n):
"""Accept-reject Poisson(llambda), proposition N=floor(X+1/2)."""
alpha = llambda
beta = np.sqrt(3 * llambda) / np.pi
ks = np.arange(0, int(3 * llambda) + 100)
log_g = proba_N(ks, alpha, beta)
log_f = poisson_logpmf(ks, llambda)
M = float(np.exp(log_f - log_g)[np.exp(log_g) > 1e-15].max() * 1.01)
out = np.empty(n, dtype=int)
i = 0
while i < n:
batch = max(1000, 2 * (n - i))
Xp = logistic_sampling_half(alpha, beta, batch)
Np = np.floor(Xp + 0.5).astype(int)
Np = Np[Np >= 0]
if len(Np) == 0:
continue
log_g_p = proba_N(Np, alpha, beta)
log_f_p = poisson_logpmf(Np, llambda)
U = np.random.rand(len(Np))
accept = np.log(U) <= (log_f_p - log_g_p - np.log(M))
acc = Np[accept]
m = min(len(acc), n - i)
out[i:i + m] = acc[:m]
i += m
return out
```

Verify your method by ploting the empirical histogram you obtained from the method you have implemented and the one of the Poisson distribution

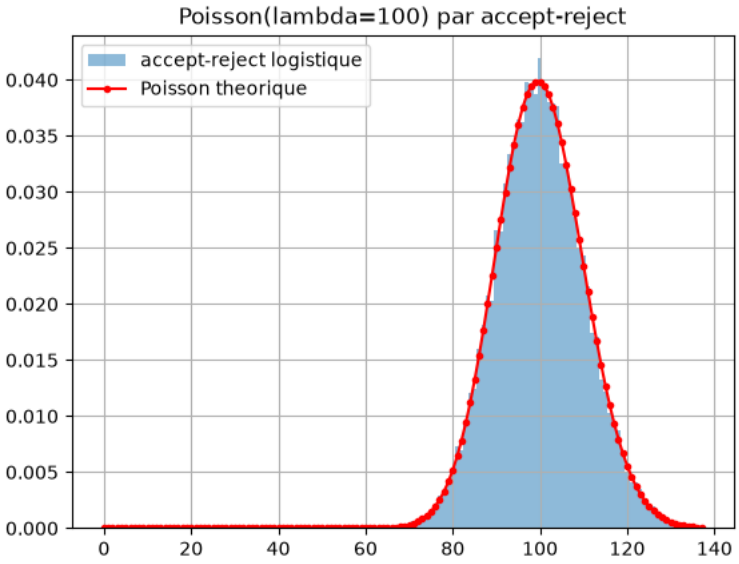
In [9] :

```
        for lam in [10, 100]:
        s = poisson_sampling_logistic(lam, 20000)
        ref_k = np.arange(0, int(s.max()) + 1)
        from scipy.special import gammaln
        theo = np.exp(ref_k * np.log(lam) - lam - gammaln(ref_k + 1))
        plt.figure()
        plt.hist(s, bins=np.arange(-0.5, s.max() + 1.5, 1), density=True, alpha=0.5,
        label="accept-reject logistique")
        plt.plot(ref_k, theo, "ro-", ms=3, label="Poisson theorique")
        plt.title(f"Poisson(lambda={lam}) par accept-reject")
        plt.legend(); plt.grid(True); plt.show()
        print(f"lambda={lam}: moy={s.mean():.2f} (theo={lam}), var={s.var():.2f}")
```




```
<Figure size 640x480 with 1 Axes>
```

```
lambda=10: moy=9.98 (theo=10), var=10.01
```




```
<Figure size 640x480 with 1 Axes>
```

```
lambda=100: moy=99.93 (theo=100), var=99.81
```

Compare the computational times of the two methods that you have implemented for sampling from the Poisson distribution for a number of samples $n=10000$ and $\lambda \in \{10,100,200\}$.

In [10] :

```
        import time
        for lam in [10, 100, 200]:
        n = 10000
        t0 = time.time(); poisson_sampling_direct(lam, n); t1 = time.time()
        t2 = time.time(); poisson_sampling_logistic(lam, n); t3 = time.time()
        print(f"lambda={lam}, n={n}: directe={t1-t0:.3f}s, logistic-AR={t3-t2:.3f}s")
        print("\nConclusion : Knuth en O(lambda) par tirage, impraticable pour grand lambda ; AR
        logistique quasi-independant de lambda.")
```

```
lambda=10, n=10000: directe=0.087s, logistic-AR=0.006s
```

```
lambda=100, n=10000: directe=0.826s, logistic-AR=0.004s
```

```
lambda=200, n=10000: directe=1.621s, logistic-AR=0.005s

Conclusion : Knuth en O(lambda) par tirage, impraticable pour grand lambda ; AR logistique
quasi-independant de lambda.
```