TP4 — Poisson inhomogene et pont brownien (corrige)

Source : TP4/Poisson_Brownian_corrige.ipynb — PDF compile (code execute).

Simulating Poisson processes and Brownian bridges

Simulating an inhomogeneous Poisson process by thinning

Inhomogeneous Poisson process

The goal here is to experiment around the thinning of inhomogeneous Poisson processes. An \textit{inhomogeneous Poisson process} (or non-homogeneous Poisson process) is a stochastic process $ \{ N(t) \}_{t \geq 0}$ with a time-varying intensity function $ \lambda(t) $ such that:

• **Incremental Independence:** For any $ 0 \leq t_1 < t_2 < \dots < t_n $, the increments $ N(t_2) - N(t_1), N(t_3) - N(t_2), \dots, N(t_n) - N(t_{n-1}) $ are independent.

• **Inhomogeneous Poisson Increment Property:** The number of events occurring in an interval $ (s, t] $, for $ s < t $, follows a Poisson distribution with mean

$$ \mathbb{E}[N(t) - N(s)] = \int_s^t \lambda(u) \, du $$ That is, $$ N(t) - N(s) \sim \text{Poisson} \left(\int_s^t \lambda(u) \, du \right). $$

• **Initial Condition:** $N(0) = 0 $.

• **Interarrival time** (or event time): The probability density function $f_{T_1}(t_0,t)$ of the first arrival time T_1 after t_0 is: $$f_{T_1}(t)=\lambda(t)e^{-\Lambda(t)}, \text{ with }\Lambda(t)=\int_{t_0}^t \lambda(u) \, du.$$

The thinning method

An inhomogeneous Poisson process of rate $\lambda(t)\leq \lambda_M, \forall t$ can be simulated by considering a bound homogeneous Poisson process of rate λ_M:

• starting at $\tau_{0}=0$

• generating the interarrival time τ of the bound Poisson process of rate λ_M by inversion sampling of the exponential law of parameter λ_M

• resampling this event as a true event with probability $\frac{\lambda(\tau+\tau_0)}{\lambda_M} $

• If $\tau+\tau_0$ is resampled as the interarrival time of the true Poisson process, output $\tau+\tau_0$. If not, update $\tau_0 = \tau_0 + \tau$ and repeat the previous steps.

Refs:

(1) APA Lewis, P. A., & Shedler, G. S. (1979). Simulation of nonhomogeneous Poisson processes by thinning. Naval Research Logistics Quarterly, 26(3), 403-413.

Questions

• Show that the thinning simulation method is correct.

Library importation

In [1] :

```
import numpy as np
import random, math
import matplotlib.pyplot as plt
from matplotlib.collections import PatchCollection
import matplotlib.lines as mlines
from matplotlib.ticker import FixedLocator

plt.rcParams['xtick.major.size'] = 8
plt.rcParams['xtick.minor.size'] = 8
plt.rcParams['ytick.major.size'] = 8
plt.rcParams['ytick.minor.size'] = 8
```

We consider the problem of simulating from the first event time of a non-homogenous Poisson process of rate λ: $$\lambda(t) = \max(\sin(t),0), \forall t\geq 0 $$

Questions

• Find λ_M such that $\lambda_M\geq \lambda(t), \forall t$

In the code below, please,

• Assign a correct value to `self.lambda_M`

• Code the `homogeneousPP_generation` routine which outputs the interarrival time of a Poisson process of homogeneous rate `self.lambda_M` by inversion sampling

• Code the `thinning_generation` routine which outputs the interarrival time of the inhomogeneous Poisson process of rate $\lambda(t)$ (given by the routine `lambda_rate`) by thinning using the PP of rate λ_M

• Code the `theoretical_arrival` routine which outputs the probability density of a given time to be the interarrival time of the target inhomogeneous PP

In [2] :

```
class max_cos_PP:
def __init__(self, lambda_M):
# Q2-Q3 : lambda(t)=max(sin(t),0) <= 1 donc lambda_M = 1 convient (borne minimale).
self.lambda_M = lambda_M

def lambda_rate(self, t):
return max(0, math.sin(t))

def homogeneousPP_generation(self):
# Q4 : inversion de l'exponentielle de parametre lambda_M : -log(U)/lambda_M
return -math.log(random.uniform(0.0, 1.0)) / self.lambda_M

def direct_generation(self):
# inversion directe du PP inhomogene (donne) : utilise Lambda(t)=int_0^t max(sin,0)
ran_tot = -math.log(random.uniform(0.0, 1.0))
tau = 2.0 * math.pi * (ran_tot//2)
ran_tot = ran_tot % 2.0
tau += math.acos(1-ran_tot)
return tau

def thinning_generation(self):
# Q5 : thinning : candidats Exp(lambda_M), acceptes avec proba lambda(t)/lambda_M
t0 = 0.0
while True:
e = self.homogeneousPP_generation()
t_cand = t0 + e
if random.uniform(0.0, 1.0) <= self.lambda_rate(t_cand) / self.lambda_M:
return t_cand
t0 = t_cand

def theoretical_arrival(self, t):
# Q6 : f(t) = lambda(t) exp(-Lambda(t)), Lambda(t)=int_0^t max(sin,0)
import numpy as _np
t = _np.asarray(t, dtype=float)
twopi = 2 * _np.pi
nfull = _np.floor(t / twopi)
r = t - nfull * twopi
Lam = 2.0 * nfull + _np.where(r <= _np.pi, 1 - _np.cos(r), 2.0)
lam = _np.maximum(_np.sin(t), 0.0)
out = lam * _np.exp(-Lam)
return out if out.shape != () else float(out)

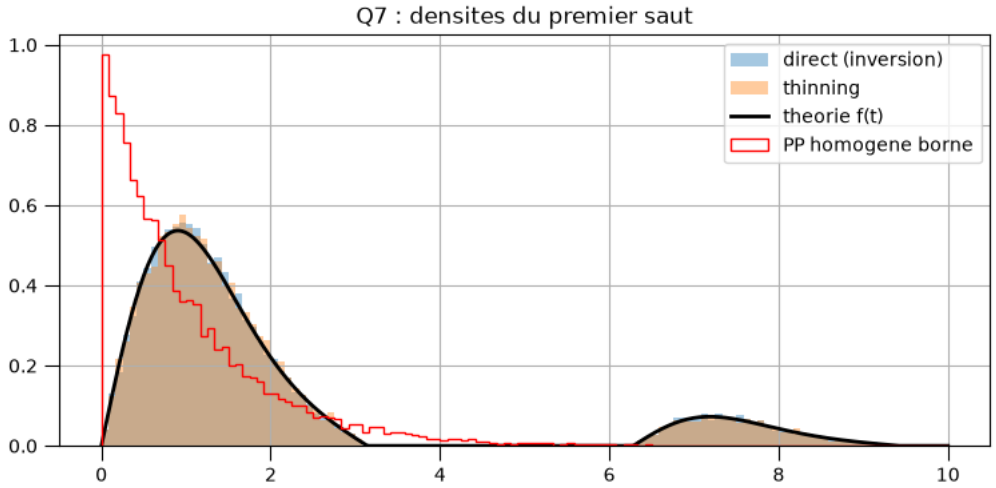
# Q1 (math, reponse) : preuve du thinning — voir corrige : les candidats forment un PP
homogene
# d'intensite lambda_M ; garder chaque point t avec proba lambda(t)/lambda_M
(eclaircissement
# independant) donne un PP d'intensite lambda(t). Formellement,
P(T1>t)=E[prod_{candidats<=t}
# (1-lambda/lambda_M)] = exp(-int_0^t lambda). Donc la loi du premier point accepte est
bien f.
PP = max_cos_PP(lambda_M=1.0)
```

Questions

• Compare the normed histograms of the interarrival times obtained with the `direct_generation`(directly sampling by inversion of the inhomogeneous PP), the `thinning_generation` and the theoretical value. Compare to the normed histogram of the interarrival times of the bound homogeneous PP of rate λ_M.

In [3] :

```
# Q7 : histogrammes normalises vs theorie + PP homogene borne
import numpy as _np
Nsim = 20000
thin = _np.array([PP.thinning_generation() for _ in range(Nsim)])
direct = _np.array([PP.direct_generation() for _ in range(Nsim)])
hom = _np.array([-math.log(random.uniform(0,1))/PP.lambda_M for _ in range(Nsim)])
tg = _np.linspace(0, 10, 400)
theo = PP.theoretical_arrival(tg)
plt.figure(figsize=(9,4))
plt.hist(direct, bins=120, range=(0,10), density=True, alpha=0.4, label="direct
(inversion)")
plt.hist(thin, bins=120, range=(0,10), density=True, alpha=0.4, label="thinning")
plt.plot(tg, theo, "k-", lw=2, label="theorie f(t)")
plt.hist(hom, bins=120, range=(0,10), density=True, histtype="step", color="red",
label="PP homogene borne")
plt.legend(); plt.grid(True); plt.title("Q7 : densites du premier saut"); plt.show()
print(f"moyennes : direct={direct.mean():.3f}, thinning={thin.mean():.3f} (theo
E[T1]~2.05)")
```



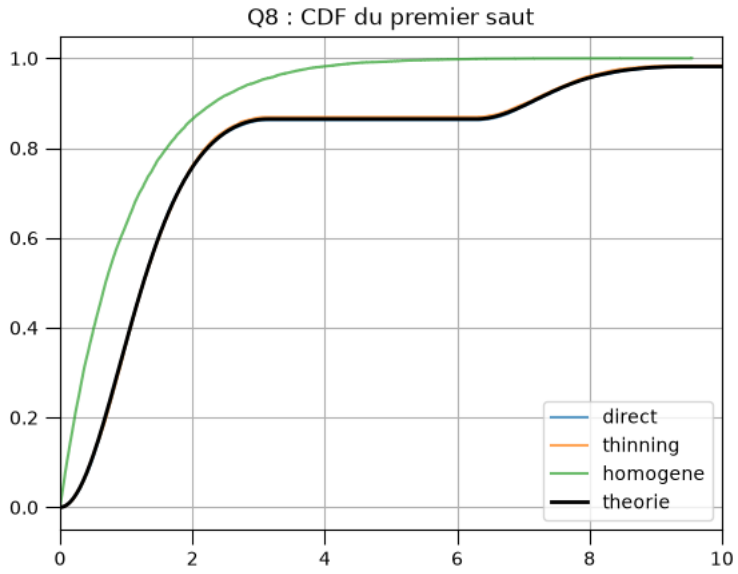

```
<Figure size 900x400 with 1 Axes>
```

```
moyennes : direct=2.200, thinning=2.154 (theo E[T1]~2.05)
```

• Compare the cumulative histograms of the first event times obtained with the `direct_generation`(directly sampling by inversion of the inhomogeneous PP), the `thinning_generation` and the theoretical values. Compare also to the cumulative histogram of the first event times of the bound homogeneous PP of rate λ_M. Confirm why the thinning method is correct.

In [4] :

```
# Q8 : fonctions de repartition (cumul) : thinning correct
xs = _np.sort(_np.linspace(0,10,400))
Ftheo = _np.array([_np.trapezoid(PP.theoretical_arrival(_np.linspace(0,x,300)),
_np.linspace(0,x,300)) for x in xs])
for name, samp in [("direct",direct),("thinning",thin),("homogene",hom)]:
s=_np.sort(samp); F=_np.arange(1,len(s)+1)/len(s)
plt.step(s, F, where="post", alpha=0.7, label=name)
plt.plot(xs, Ftheo, "k-", lw=2, label="theorie")
plt.xlim(0,10); plt.legend(); plt.grid(True); plt.title("Q8 : CDF du premier saut");
plt.show()
print("Les courbes direct et thinning coincident avec la theorie ; l'homogene est
stochastiquement plus petit.")
```




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

```
Les courbes direct et thinning coincident avec la theorie ; l'homogene est stochastiquement plus
petit.
```

• **[BONUS]** How is the number of generated bound events required to resample a true one behaving with λ_M? Please, provide a plot.

• **[BONUS]** What is a good choice for λ_M?

In [5] :

```
        # Q9-Q10 BONUS : efficacite vs lambda_M (nb moyen de candidats par vrai point)
        import time
        for lM in [1.0, 1.5, 2.0, 3.0]:
        pp = max_cos_PP(lambda_M=lM)
        t0=time.time(); K=2000
        s=[pp.thinning_generation() for _ in range(K)]
        # nb candidats ~ lM * E[T1] ; acceptance ~ int lambda / (lM E[T]) ; rendement = 1/lM *
        moyenne de lambda
        print(f"lambda_M={lM}: temps {time.time()-t0:.2f}s pour {K} points,
        moyenne={_np.mean(s):.3f}")
        print("Q10 : le bon choix est la plus petite majoration possible, lambda_M=1 (taux
        d'acceptation maximal).")
        print("Un lambda_M plus grand gaspille des candidats (rejet proportionnel a
        1-lambda/lambda_M).")
```

```
lambda_M=1.0: temps 0.01s pour 2000 points, moyenne=2.098
lambda_M=1.5: temps 0.01s pour 2000 points, moyenne=2.187
lambda_M=2.0: temps 0.02s pour 2000 points, moyenne=2.225
lambda_M=3.0: temps 0.03s pour 2000 points, moyenne=2.198
Q10 : le bon choix est la plus petite majoration possible, lambda_M=1 (taux d'acceptation
maximal).
Un lambda_M plus grand gaspille des candidats (rejet proportionnel a 1-lambda/lambda_M).
```

Questions

In the code below, please,

• Code the `intermediate_position` routine which generates an intermediate position at a given time between two successive positions of the Brownian motions.

• Code the `complete_motion` routine which generates the intermediate middle position between all two successive positions of a given Brownian motion sequence

In [6] :

```
        # Q11-Q12 : pont brownien (a completer dans la classe ci-dessous si besoin de test)
        # Rappel : B_s | B_t1,B_t2 ~ N(m, v), m=Bt1+(Bt2-Bt1)(s-t1)/(t2-t1),
        v=(s-t1)(t2-s)/(t2-t1).
```

In [7] :

```
        class brownian_motion:
        def __init__(self,):
        pass

        def position_sequence(self, xinit, t_arr):
        dt_list = t_arr[1:] - t_arr[:-1]
        dB = np.sqrt(dt_list) * np.random.randn(len(t_arr)-1)
        B = np.insert(xinit + np.cumsum(dB),0,xinit)
        return B

        def intermediate_position(self, s, t1, t2, Bt1, Bt2):
        # Q11 : pont brownien
        m = Bt1 + (Bt2 - Bt1) * (s - t1) / (t2 - t1)
        v = (s - t1) * (t2 - s) / (t2 - t1)
        return np.random.normal(m, np.sqrt(max(v, 0.0)))

        def complete_motion(self, t_arr, Bt):
        # Q12 : raffine chaque intervalle par son milieu (pont brownien)
        t_arr = np.asarray(t_arr, dtype=float); Bt = np.asarray(Bt, dtype=float)
        t_new = [t_arr[0]]; B_new = [Bt[0]]
        for k in range(len(t_arr)-1):
        t1, t2 = t_arr[k], t_arr[k+1]
        s = 0.5*(t1+t2)
        b = self.intermediate_position(s, t1, t2, Bt[k], Bt[k+1])
        t_new += [s, t2]; B_new += [b, Bt[k+1]]
        return np.array(t_new), np.array(B_new)
```

Questions

• Generate the positions at time $0,1,2, \dots 10$ (time increment = 1) of a Brownian motion and complete the sequence of the positions of the sequence down to a time increment of 0.03125. Plot an example.

• Check your result by comparing from samples obtained by direct simulation and obtained by the iterated generation:

• the distribution of the position value at $t=5-0.03125$. Add the expected theoretical distribution to the plot.

• the distribution of the maximal absolute position value obtained.

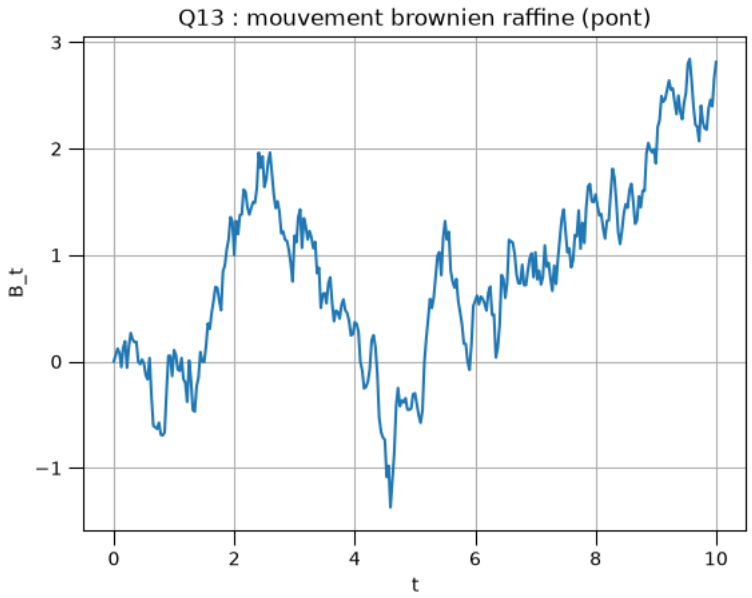
• the distribution of the time t at which the maximal absolute position value is reached.

Questions\n13-14 : raffinement brownien et validation\n

In [8] :

```
        # Q13 : positions en 0..10 (pas 1) raffinees jusqu'au pas 0.03125 (=1/32, 5 iterations)
        bm = brownian_motion()
        t = np.arange(0, 11, 1.0)
        B = bm.position_sequence(0.0, t)
        for _ in range(5):
        t, B = bm.complete_motion(t, B)
        print(f"pas final={t[1]-t[0]:.5f}, n={len(t)}")
        plt.plot(t, B); plt.grid(True); plt.title("Q13 : mouvement brownien raffine (pont)");
        plt.xlabel("t"); plt.ylabel("B_t"); plt.show()
```

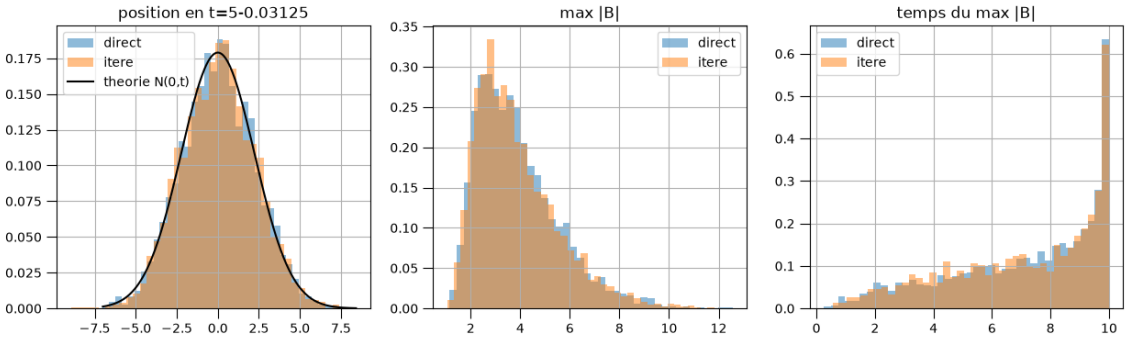
```
pas final=0.03125, n=321
```




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

In [9] :

```
# Q14 : validation direct vs itere
M = 3000
t0 = np.arange(0, 11, 1.0)
# direct fin : simulation exacte au pas 1/32
tf = np.arange(0, 10+0.03125/2, 0.03125)
Bd = np.array([brownian_motion().position_sequence(0.0, tf) for _ in range(M)])
# itere : grossier puis 5 raffinements
Bi = []
for _ in range(M):
tc = np.arange(0, 11, 1.0); Bc = brownian_motion().position_sequence(0.0, tc)
for _ in range(5):
tc, Bc = brownian_motion().complete_motion(tc, Bc)
Bi.append(Bc)
Bi = np.array(Bi)
idx = int(round((5-0.03125)/0.03125)) # position t=5-0.03125
fig, ax = plt.subplots(1,3, figsize=(15,4))
ax[0].hist(Bd[:,idx], bins=40, density=True, alpha=0.5, label="direct")
ax[0].hist(Bi[:,idx], bins=40, density=True, alpha=0.5, label="itere")
xg = np.linspace(Bd[:,idx].min(), Bd[:,idx].max(), 200)
ax[0].plot(xg, 1/np.sqrt(2*np.pi*(5-0.03125))*np.exp(-xg**2/(2*(5-0.03125))), "k-",
label="theorie N(0,t)")
ax[0].legend(); ax[0].set_title("position en t=5-0.03125"); ax[0].grid(True)
ax[1].hist(np.max(np.abs(Bd),axis=1), bins=40, density=True, alpha=0.5, label="direct")
ax[1].hist(np.max(np.abs(Bi),axis=1), bins=40, density=True, alpha=0.5, label="itere")
ax[1].legend(); ax[1].set_title("max |B|"); ax[1].grid(True)
ax[2].hist(np.argmax(np.abs(Bd),axis=1)*0.03125, bins=40, density=True, alpha=0.5,
label="direct")
ax[2].hist(np.argmax(np.abs(Bi),axis=1)*0.03125, bins=40, density=True, alpha=0.5,
label="itere")
ax[2].legend(); ax[2].set_title("temps du max |B|"); ax[2].grid(True)
plt.show()
print("Les 3 distributions coincident : le raffinement par ponts browniens est exact.")
```




```
<Figure size 1500x400 with 3 Axes>
```

```
Les 3 distributions coincident : le raffinement par ponts browniens est exact.
```