TP n° 1

Géostatistique

Soheil Salmani & Komi Agblodoe

17 janvier 2020

1 Processus Gaussiens

Les tâches ont été réalisées avec le langage Python (version 3) et les bibliothèques Numpy et Matplotlib.

1.1 Tâche 1

La tâche consiste à générer un champ aléatoire en considérant divers fonctions de covariance nommés *Delta*, *Sphérique*, *Exponentiel*, *Matérn-3/2*, *Matérn-5/2* et *Gaussien*.

On considère tout d'abord une grille régulière $S = [0, 1]^2$ comportant $n = 32$ points.

En fixant les paramètre $\sigma = 1$ et $a = 0.1$, nous obtenons les figures 1, 2, 3, 4, 5 et 6. On constate un adoucissement de plus en plus prononcé.

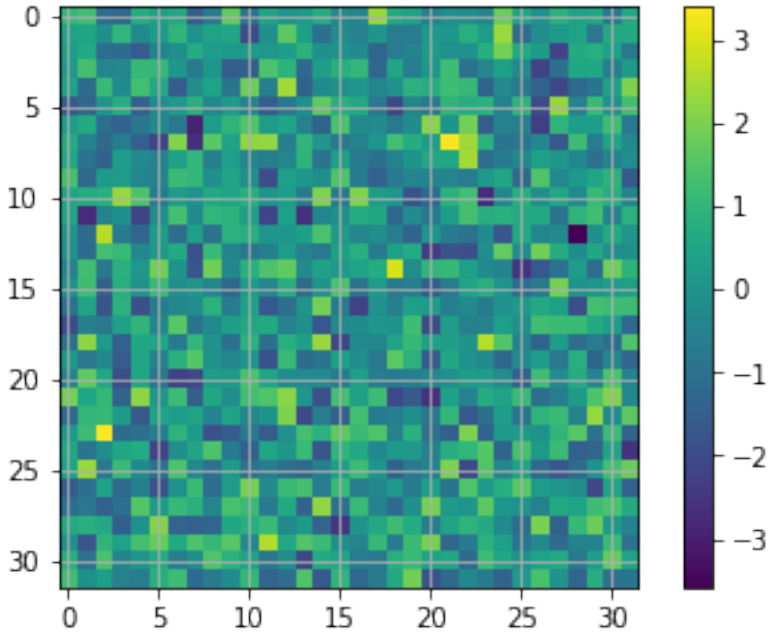

Figure 1 – Simulation ($n = 32, \sigma = 1, a = 0.1$) en utilisant la fonction de covariance delta

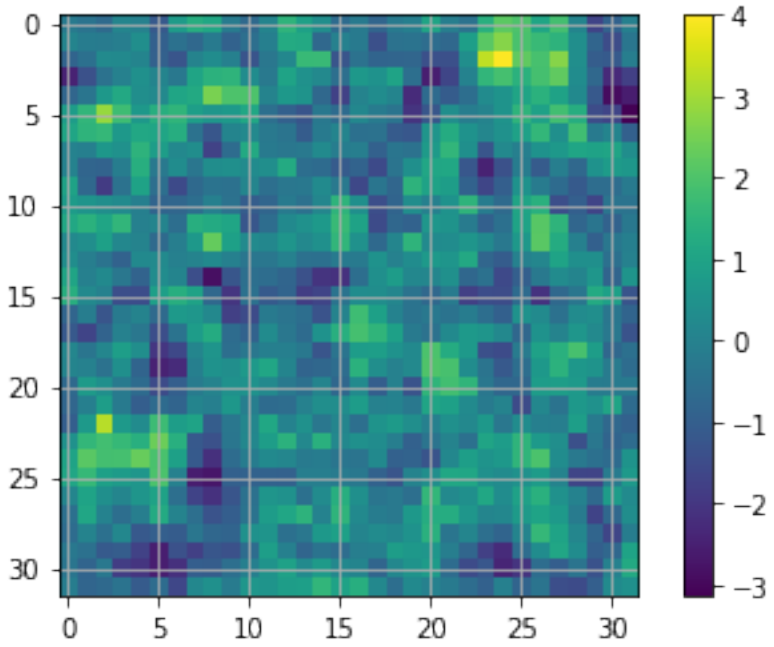

FIGURE 2 – Simulation $(n = 32, \sigma = 1, a = 0.1)$ en utilisant la fonction de covariance sphérique

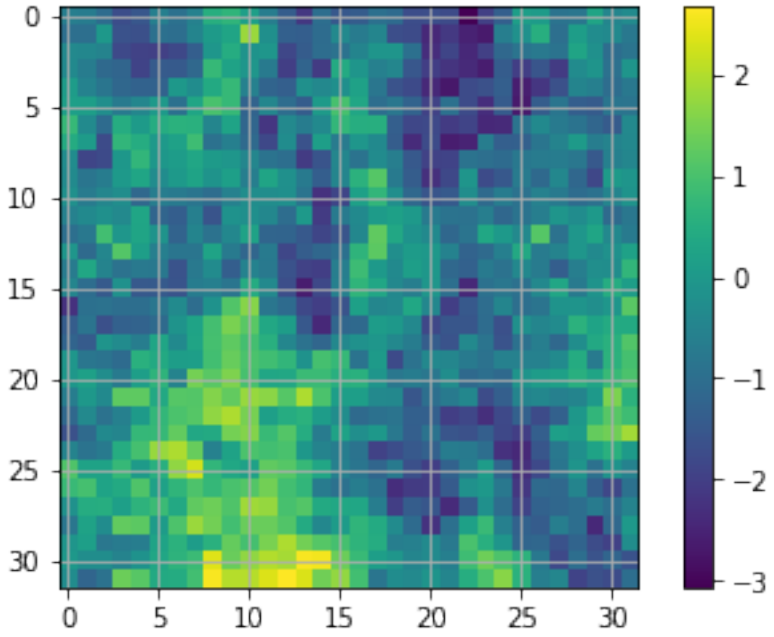

FIGURE 3 – Simulation $(n = 32, \sigma = 1, a = 0.1)$ en utilisant la fonction de covariance exponentiel

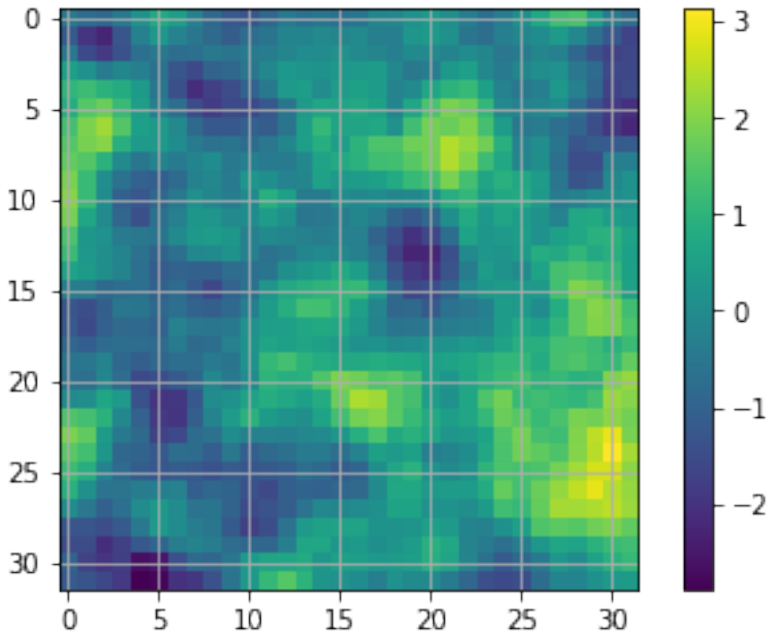

FIGURE 4 – Simulation $(n = 32, \sigma = 1, a = 0.1)$ en utilisant la fonction de covariance Matérn-3/2

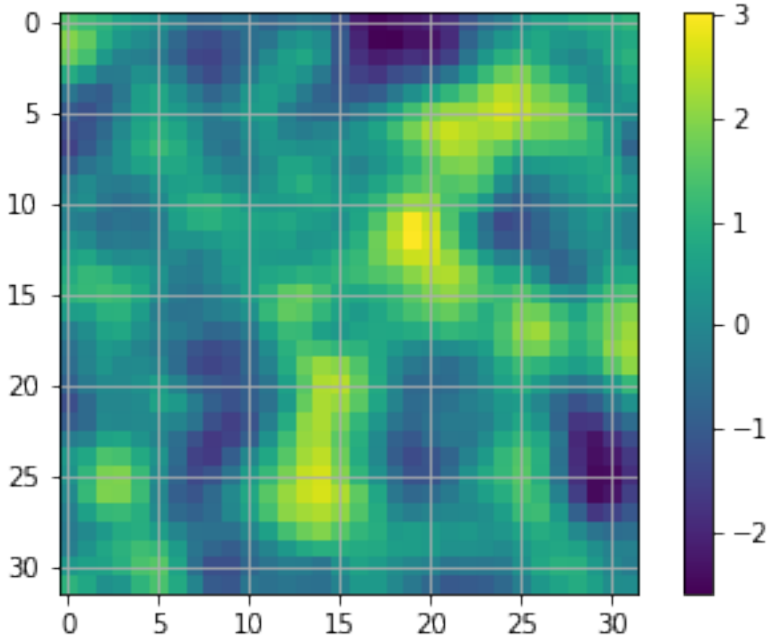

FIGURE 5 – Simulation $(n = 32, \sigma = 1, a = 0.1)$ en utilisant la fonction de covariance Matérn-5/2

On utilise désormais que la fonction de covariance gaussienne. La σ fixée, nous faisons varier a. Nous obtenons les représentations 7, 8 et 9. On remarque que a a une influence sur la distance de corrélation. Deux points éloignés sont d'avantage corrélés si a est élevé.

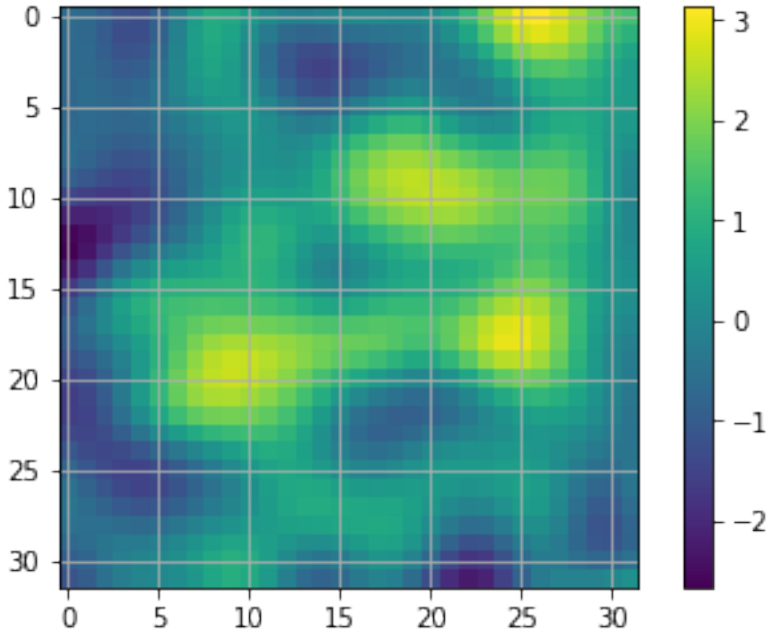

FIGURE 6 – Simulation $(n = 32, \sigma = 1, a = 0.1)$ en utilisant la fonction de covariance gaussien

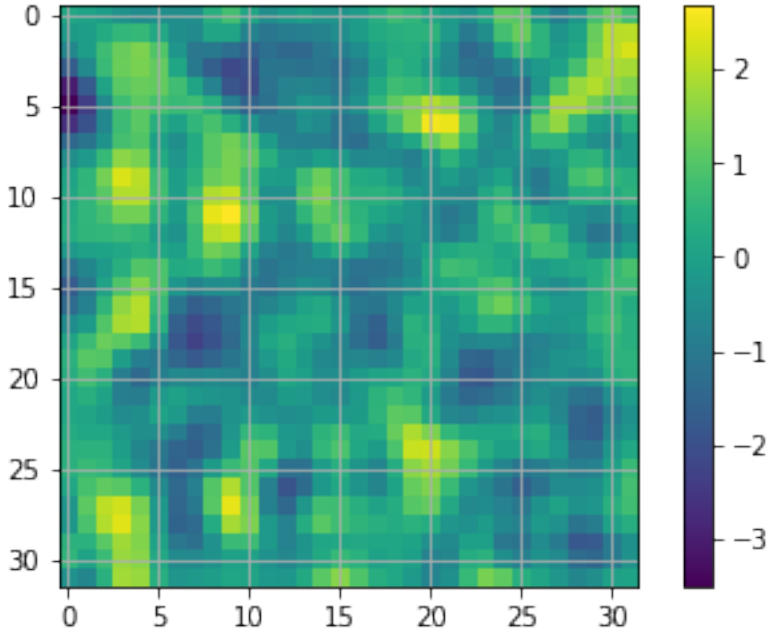

FIGURE 7 – Simulation $(n = 32, \sigma = 1, a = 0.05)$ en utilisant la fonction de covariance gaussienne

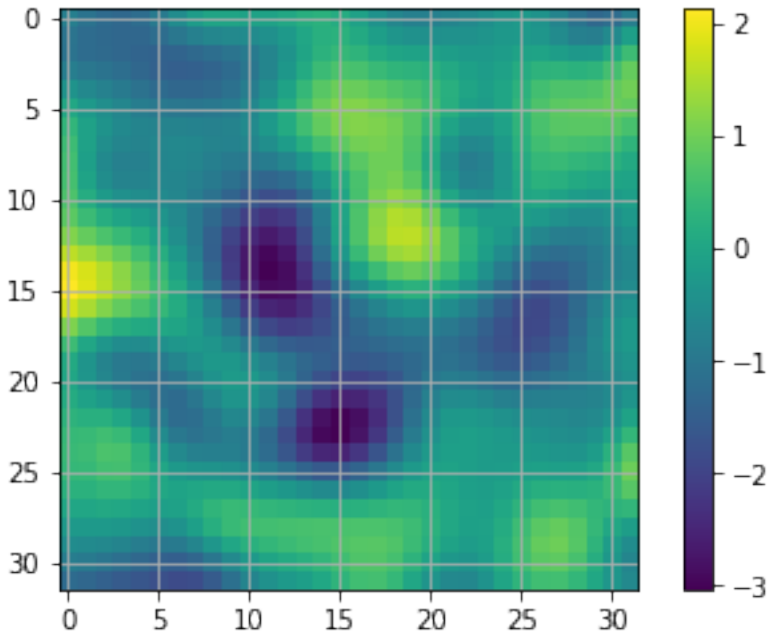

FIGURE 8 – Simulation ($n = 32, \sigma = 1, a = 0.10$) en utilisant la fonction de covariance gaussienne

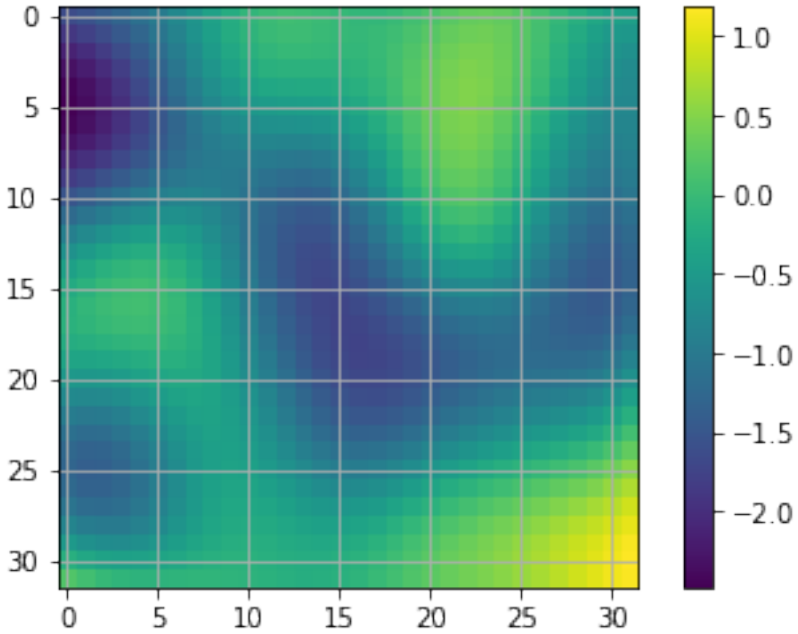

FIGURE 9 – Simulation ($n = 32, \sigma = 1, a = 0.15$) en utilisant la fonction de covariance gaussienne

En fixant a et en augmentant progressivement σ, nous obtenons des valeurs d'avantage contrastées.

On considère maintenant la fonction de covariance Matérn-5/2. On souhaite constater l'influence de n. Les figures 10 et 11 présente deux réalisations, l'une pour $n = 32$ et l'autre pour $n = 64$. La représentation graphique est plus adoucis lorsque n croit.

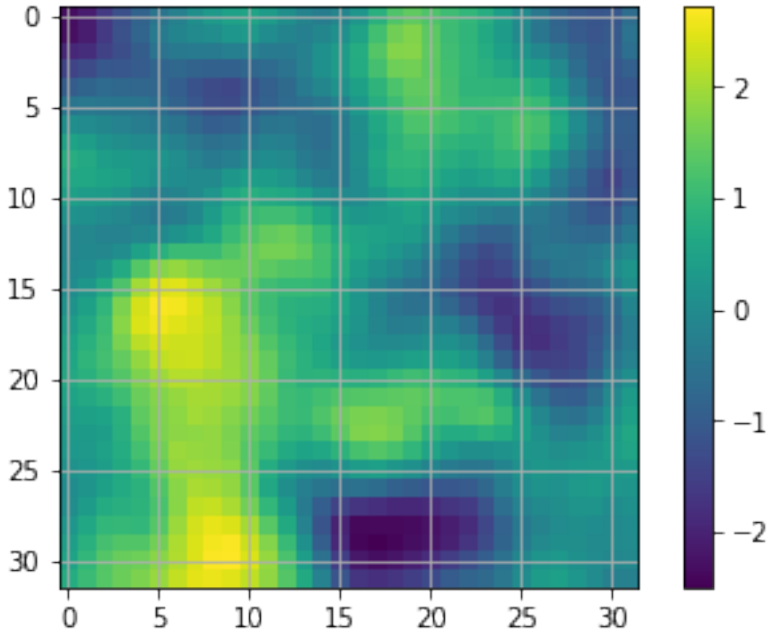

FIGURE 10 – Simulation $(n = 32, \sigma = 1, a = 0.15)$ en utilisant la fonction de covariance gaussienne

[H]

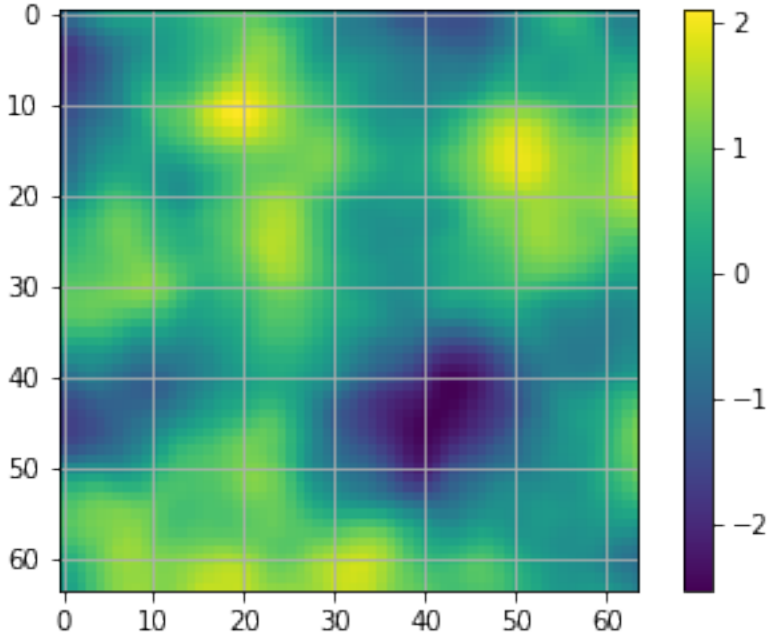

FIGURE 11 – Simulation $(n = 64, \sigma = 1, a = 0.15)$ en utilisant la fonction de covariance gaussienne

1.2 Tâche 2

Pour obtenir une réalisation dans laquelle une direction est suggérée, on peut redéfinir la fonction *norme* en ajoutant des coefficients pour chaque coordonnée. En prenant un coefficient élevé pour chaque coordonnée du paramètre h, nous pouvons obtenir des réalisations comme présentées dans les figures, 12 et 13.

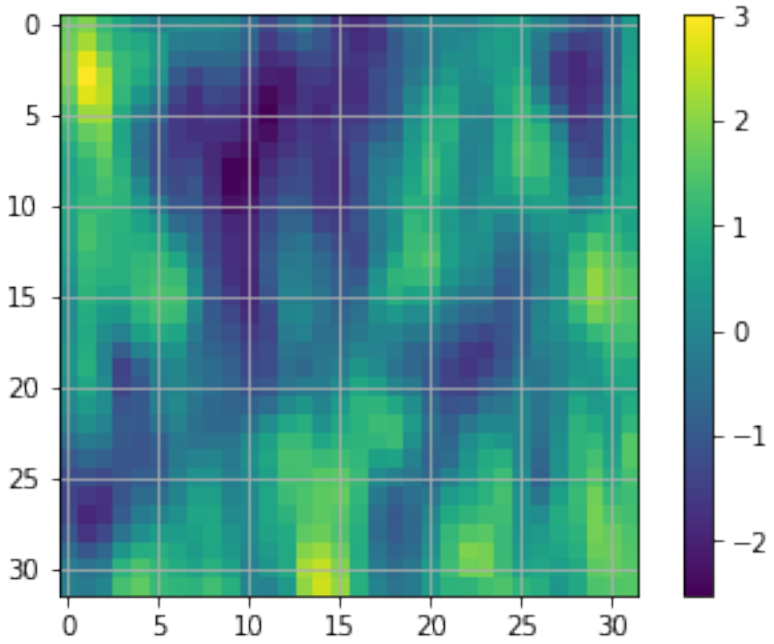

FIGURE 12 – Réalisation avec Matérn-5/2 dans laquelle une direction (verticale) est suggérée

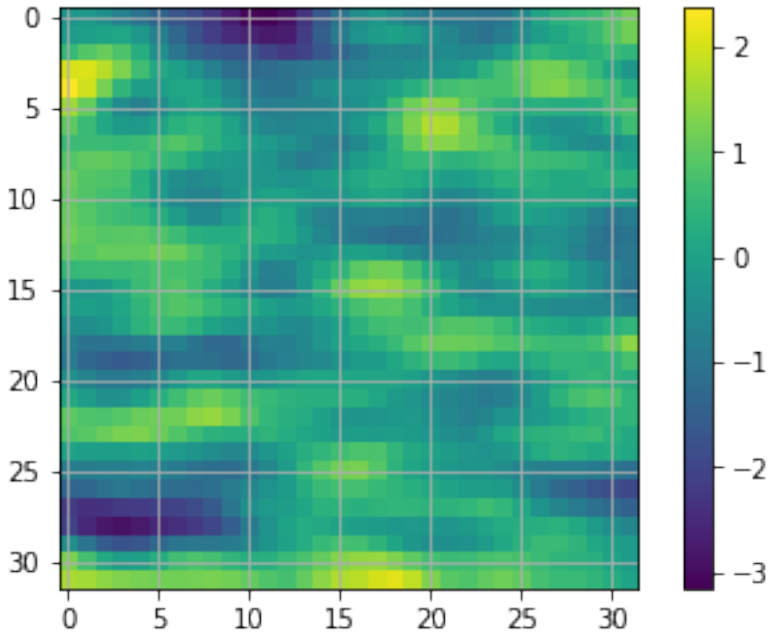

FIGURE 13 – Réalisation avec Matérn-5/2 dans laquelle une direction (horizontale) est suggérée

2 Processus ponctuels de Poisson

Les tâches ont été réalisées avec le langage R en utilisant les bibliothèques tidyverse, MASS et plotly.

2.1 Tâche 1

Nous simulons en R des processus comme décrit dans l'énoncé. Les résultats sont présentés des les figures 14 et 15.

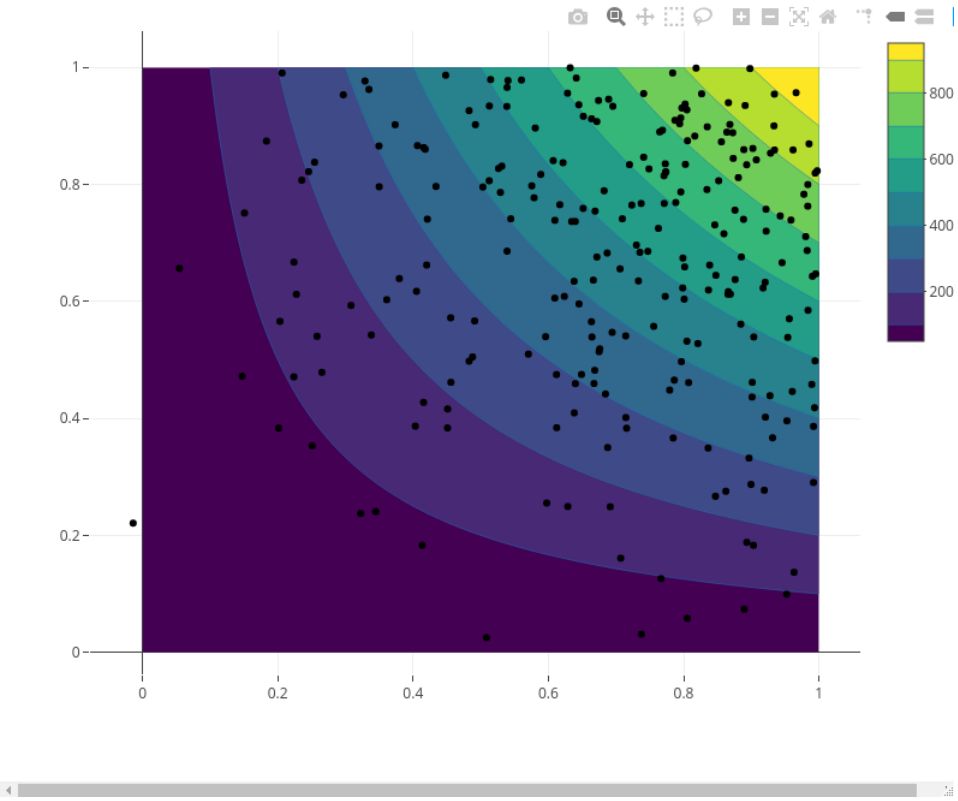

FIGURE 14 – Réalisation d'un processus ponctuel de Poisson (1)

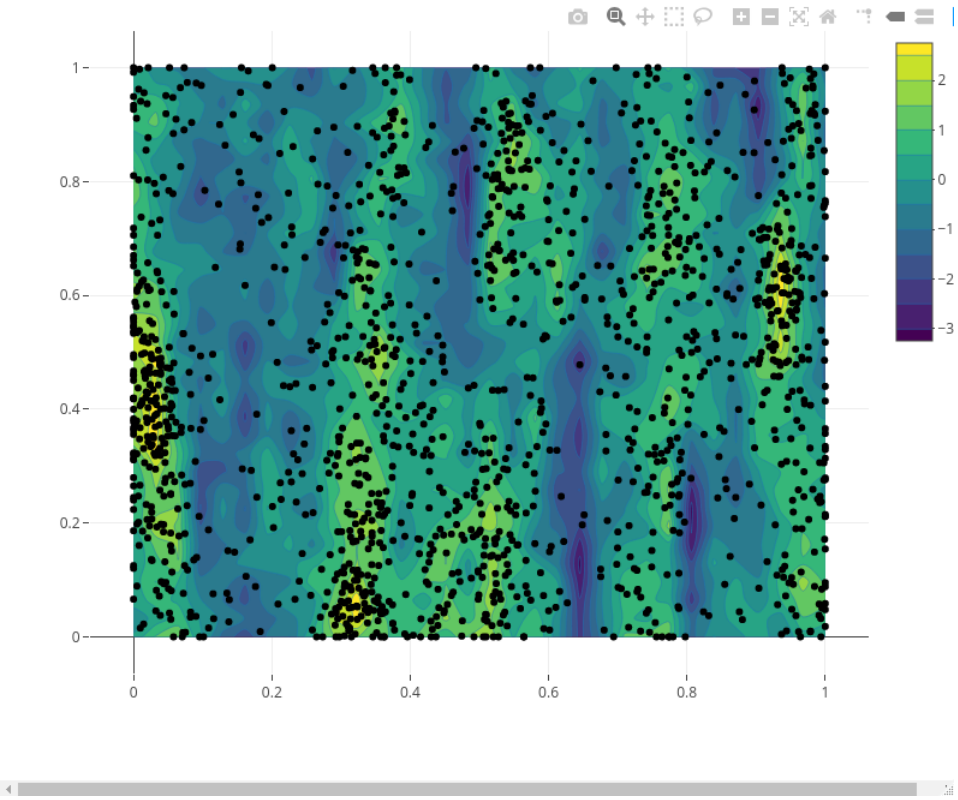

FIGURE 15 – Réalisation d'un processus ponctuel de Poisson (2)

A Code source

A.1 Tâches de la partie 1

```
# Liste des packages
import numpy as np
from numpy.linalg import norm
from numpy import arccos, sqrt, exp, pi
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import matplotlib.cbook as cbook
from matplotlib.path import Path
from matplotlib.patches import PathPatch

# Paramètres
sigma = 1
a = 0.1
n = 32
mu = np.zeros(n**2)

# Fonctions de covariances
c1 = lambda h: sigma**2 if not h.any() else 0
def c2(h):
    if norm(h) <= a:
        return (2*sigma**2/pi)*(arccos(norm(h)/a)-(norm(h)/a)*sqrt(1-(norm(h)**2)/a**2))
    else:
        return 0
c3 = lambda h: sigma**2*exp(-(norm(h))/a)
c4 = lambda h: sigma**2*(1+sqrt(3)*norm(h)/a)*exp(-sqrt(3)*norm(h)/a)
c5 = lambda h: sigma**2*(1+sqrt(5)*norm(h)/a+(5*norm(h)**2)/(3*a**2))*exp(-sqrt(5)*norm(h)/a)
c6 = lambda h: sigma**2*exp(-norm(h)**2/(2*a**2))

# Construit l'ensemble S
A = []
for i in np.linspace(0, 1, n):
    for j in np.linspace(0, 1, n):
        A.append((j, i))
S_1 = [a[0] for a in A]
S_2 = [a[1] for a in A]
S = np.array([S_1, S_2])

# Construit les matrices de covariance
cov_c1 = np.zeros((n**2, n**2))
for i in range(n**2):
    for j in range(n**2):
        s_i = S[:,i]
        s_j = S[:,j]
        cov_c1[i,j] = c1(s_i - s_j)

cov_c2 = np.zeros((n**2, n**2))
for i in range(n**2):
    for j in range(n**2):
        s_i = S[:,i]
        s_j = S[:,j]
        cov_c2[i,j] = c2(s_i - s_j)

cov_c3 = np.zeros((n**2, n**2))
for i in range(n**2):
    for j in range(n**2):
        s_i = S[:,i]
        s_j = S[:,j]
        cov_c3[i,j] = c3(s_i - s_j)

cov_c4 = np.zeros((n**2, n**2))
for i in range(n**2):
    for j in range(n**2):
        s_i = S[:,i]
        s_j = S[:,j]
        cov_c4[i,j] = c4(s_i - s_j)
```

```
cov_c5 = np.zeros((n**2, n**2))
for i in range(n**2):
    for j in range(n**2):
        s_i = S[:,i]
        s_j = S[:,j]
        cov_c5[i,j] = c5(s_i - s_j)

cov_c6 = np.zeros((n**2, n**2))
for i in range(n**2):
    for j in range(n**2):
        s_i = S[:,i]
        s_j = S[:,j]
        cov_c6[i,j] = c6(s_i - s_j)

# Affiche une réalisation avec chaque fonction de covariance
res = np.random.multivariate_normal(mu, cov_c1)
res = res.reshape(n, n)
plt.imshow(res)
plt.grid(True)
plt.colorbar()
plt.show()

res = np.random.multivariate_normal(mu, cov_c2)
res = res.reshape(n, n)
plt.imshow(res)
plt.grid(True)
plt.colorbar()
plt.show()

res = np.random.multivariate_normal(mu, cov_c3)
res = res.reshape(n, n)
plt.imshow(res)
plt.grid(True)
plt.colorbar()
plt.show()

res = np.random.multivariate_normal(mu, cov_c4)
res = res.reshape(n, n)
plt.imshow(res)
plt.grid(True)
plt.colorbar()
plt.show()

res = np.random.multivariate_normal(mu, cov_c5)
res = res.reshape(n, n)
plt.imshow(res)
plt.grid(True)
plt.colorbar()
plt.show()

res = np.random.multivariate_normal(mu, cov_c6)
res = res.reshape(n, n)
plt.imshow(res)
plt.grid(True)
plt.colorbar()
plt.show()
```

A.2 Tâches de la partie 2

```
rho <- function(x, y) 10^3 * x * y

lambda <- integrate(
  function(y) {
    sapply(y, function(y) {
      integrate(function(x) rho(x, y), 0, 1)$value
    })
  }, 0, 1
)$value
```

```
nb_points <- rpois(1, lambda)

probas <- rho(S[, 1], S[, 2]) / sum(rho(S[, 1], S[, 2]))
pts <- sample(1:n^2, nb_points, replace = TRUE, prob = probas)
coord <- S[pts, ] - matrix(abs(rnorm(2 * nb_points, 0, 0.02)), ncol = 2)

p <- plot_ly(
  type = "contour",
  x = x,
  y = y,
  z = matrix(rho(S[, 1], S[, 2]), nrow = n),
  autocontour = TRUE,
  width = 800, height = 600
) %>%
  add_trace(x = coord[, 1], y = coord[, 2], type = "scatter", mode = "markers", color = I("black"))
p

lambda <- sum(exp(gaussian_vec))

nb_points <- rpois(1, lambda)

probas <- exp(gaussian_vec) / sum(exp(gaussian_vec))
pts <- sample(1:n^2, nb_points, replace = TRUE, prob = probas)
coord <- S[pts, ] + matrix(rnorm(2 * nb_points, 0, 0.02), ncol = 2)
coord[coord < 0] <- 0
coord[coord > 1] <- 1

p <- plot_ly(
  type = "contour",
  x = x,
  y = y,
  z = matrix(gaussian_vec, nrow = n, byrow = TRUE),
  autocontour = TRUE,
  width = 800, height = 600
) %>%
  add_trace(x = coord[, 1], y = coord[, 2], type = "scatter", mode = "markers", color = I("black"))
```