



MESURE DE LA DISTANCE PARCOURUE PAR UN FOOTBALLEUR ET EVALUATION DE L'INCERTITUDE

MOUHAMED NOUROU DINE CISSE

Travaux d'initiative personnelle encadrés
(TIPE) 2023-2024

Plan

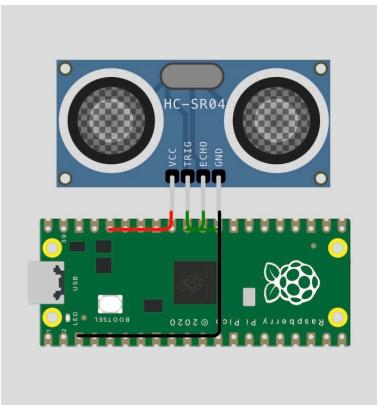
1. Introduction
2. Principe de la triangulation
3. Etude Théorique
4. Etude statistique
5. Conclusion et discussion
6. Annexe

Introduction

La mesure de la distance parcourue par un footballeur permet:

- Le suivi de la performance des joueurs
- L'analyse de la condition physique
- L'optimisation des stratégies d'entraînement et de jeu
- L'aide à la récupération et à la prévention des blessures

Elle se fait soit a l'aide de capteurs soit par stéréovision.



a) Brassière GPS

b) Modèle



Problématique et objectif:

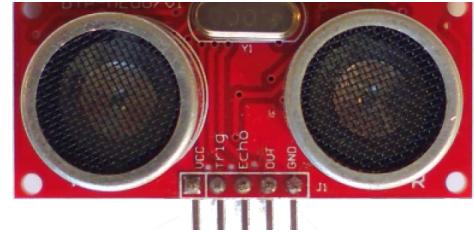
Problématique:

La problématique de recherche concerne la détermination de la distance parcourue par un footballeur et l'évaluation précise de l'incertitude associée.

Objectif:

Mettre en œuvre un algorithme python capable à partir des coordonnées relevées, de mesurer la distance parcourue par le footballeur avec une évaluation précise de l'incertitude associée.

Principe de la triangulation:



Capteurs ultrason(émetteur/ récepteur)

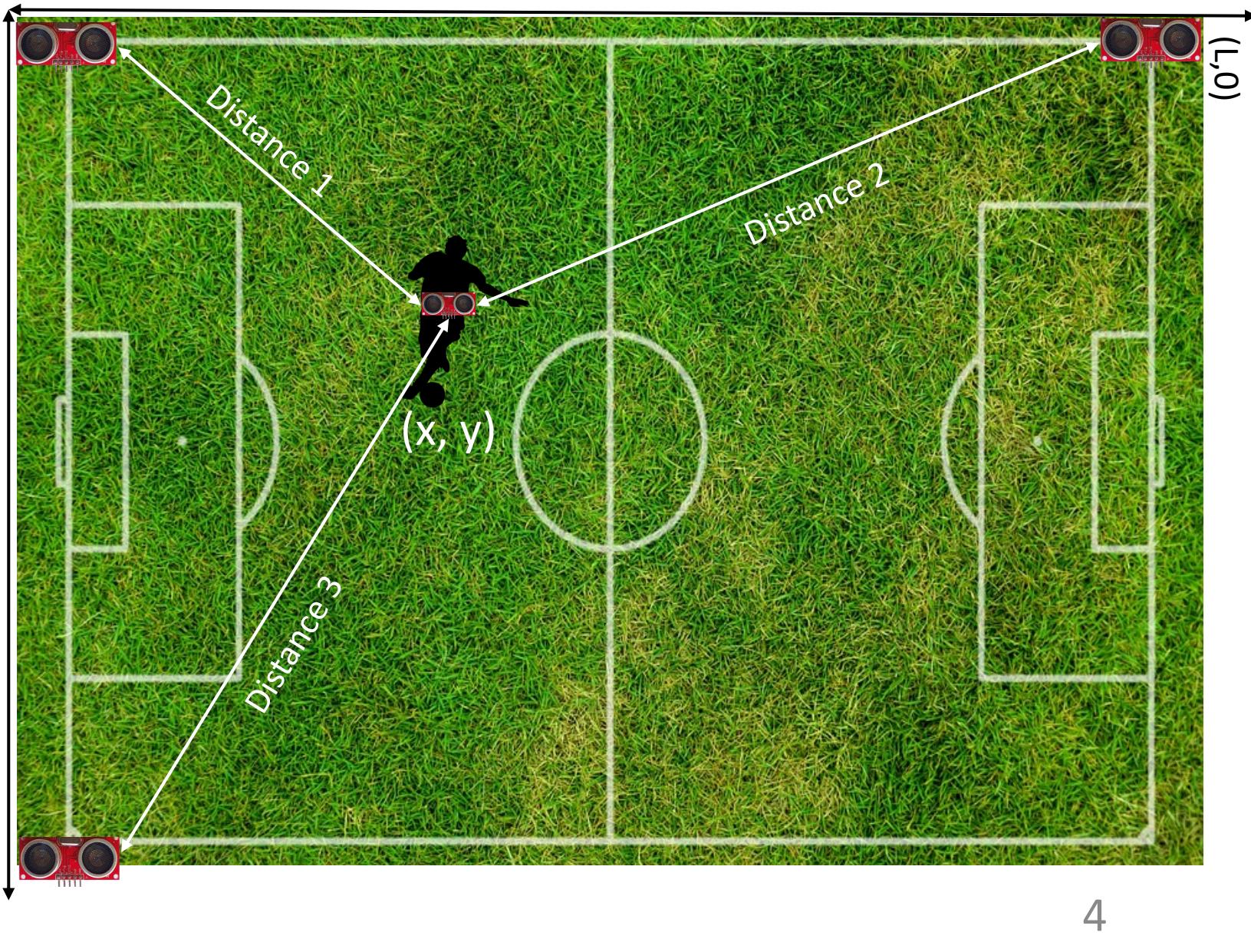
$$x = \frac{(distance\ 1)^2 - (distance\ 2)^2 + L^2}{2L}$$

$$y = \frac{(distance\ 1)^2 - (distance\ 3)^2 + l^2}{2l}$$

(0,0)

(0, l)

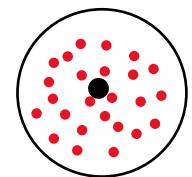
4



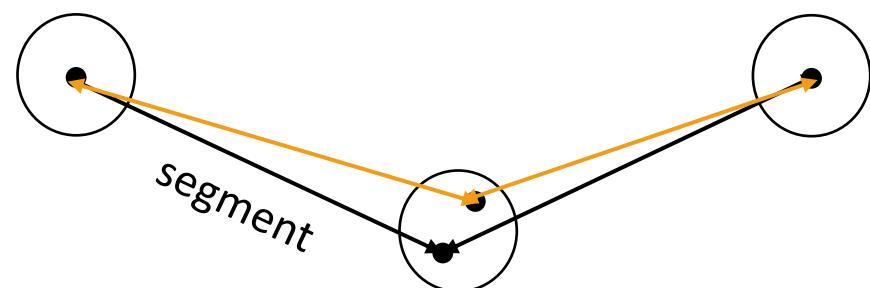
Etude Théorique

σ_x : incertitude des capteurs (abscisse)

σ_y : incertitude des capteurs (ordonnée)

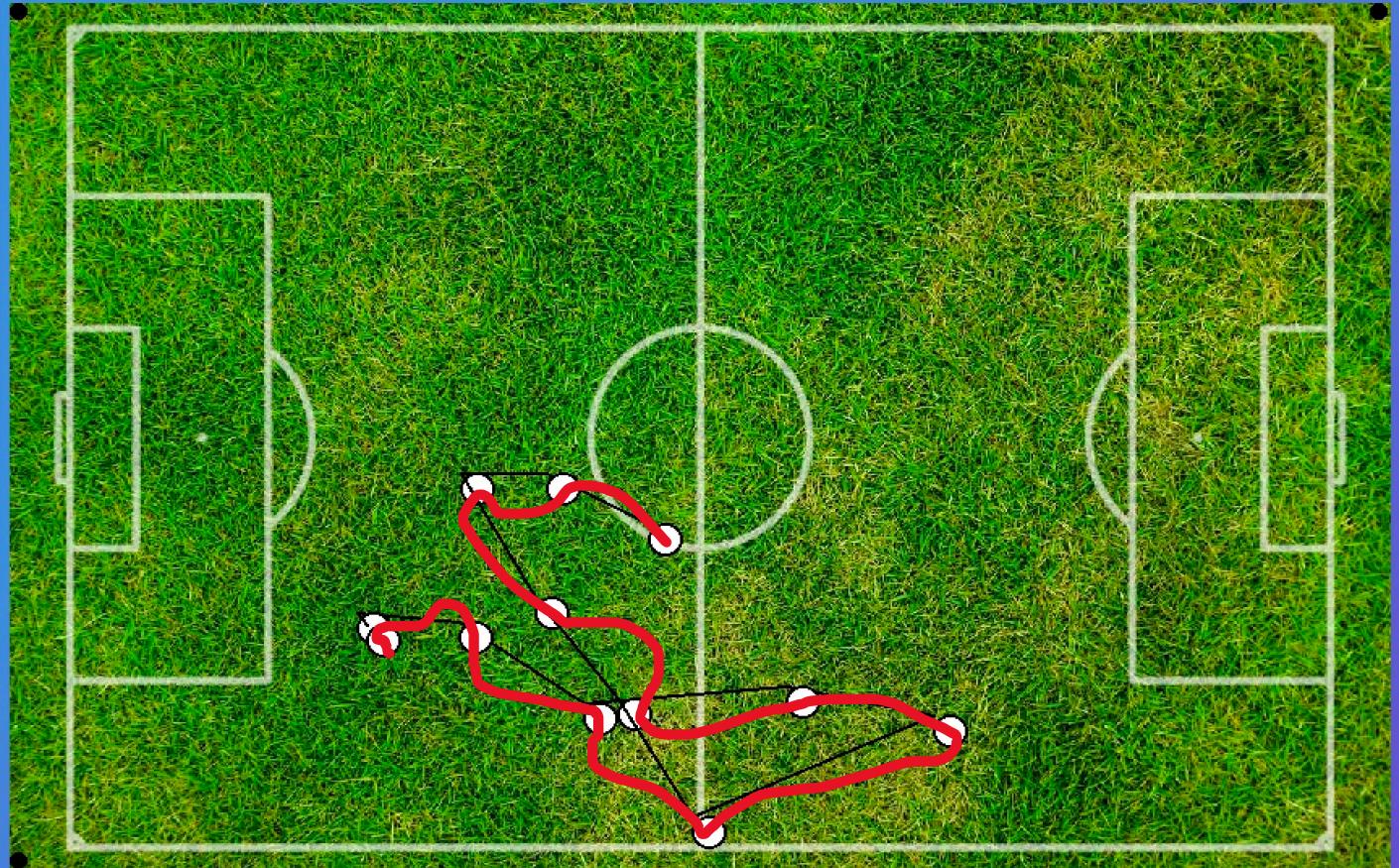


: Cercle d'incertitude
(rayon = 1 m)

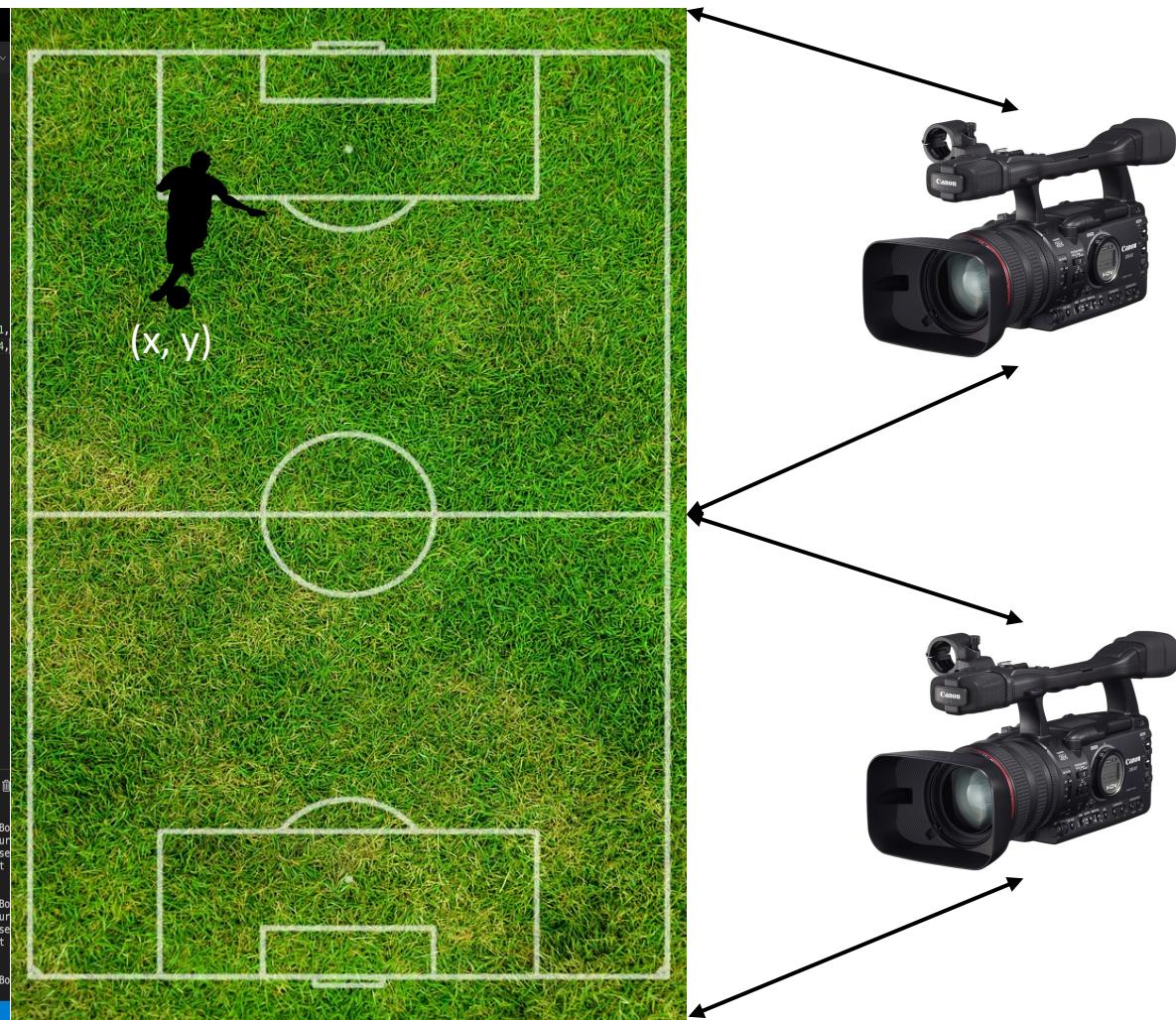
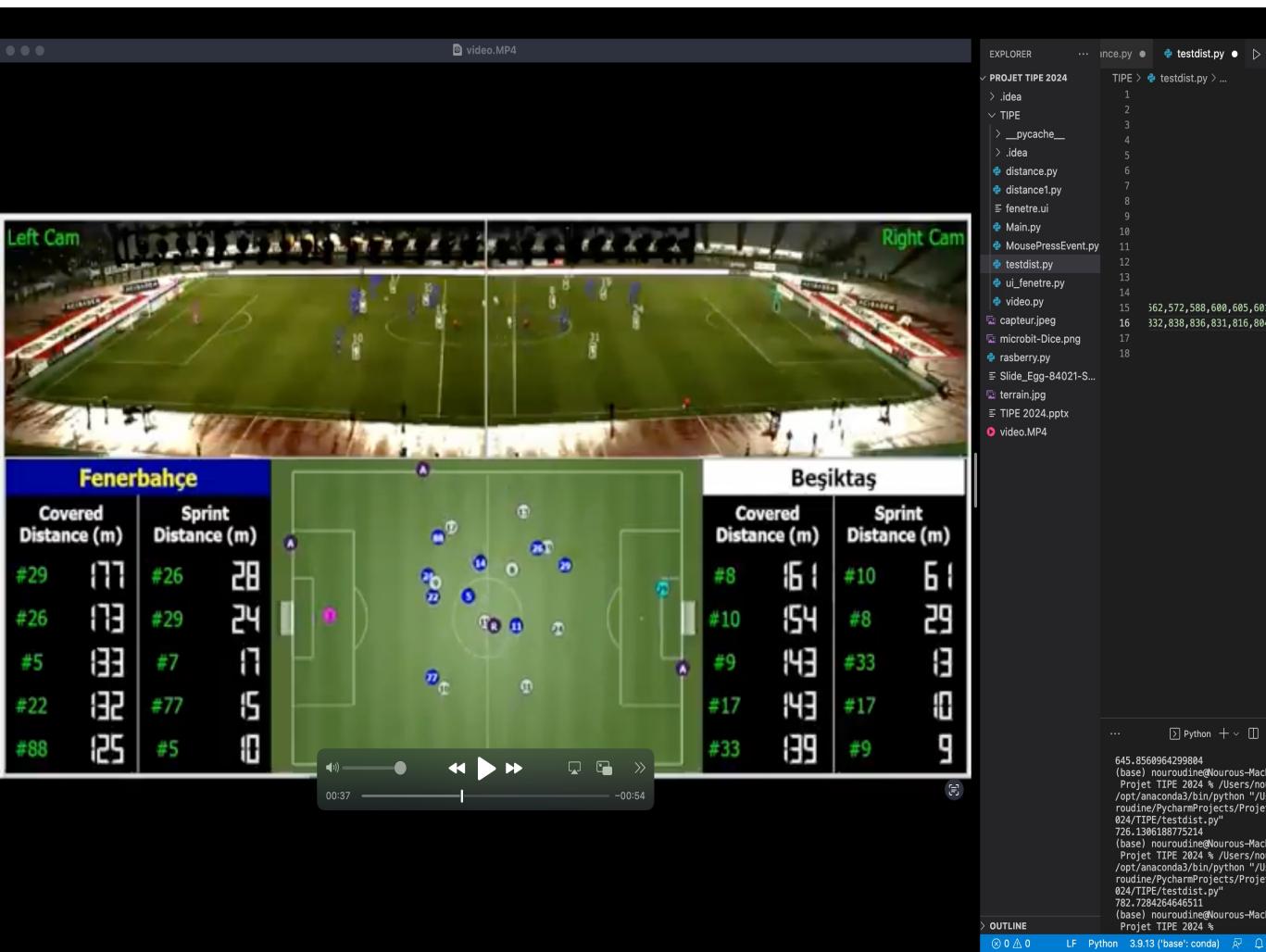


 : Trajectoire réelle

Trajectoire du joueur

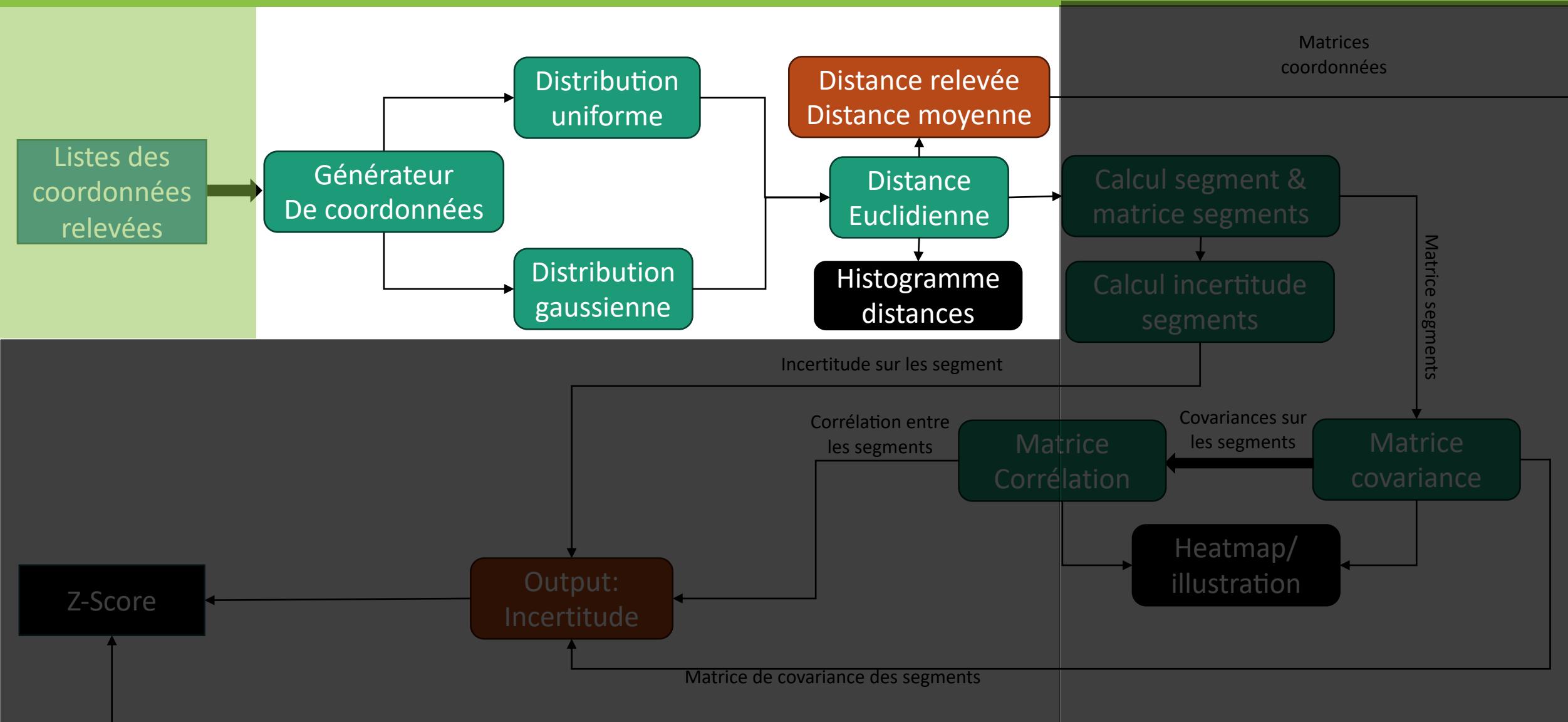


Prise de données réelles (coordonnées):



Calcul de la distance par stéréovision

Etude statistique



Distribution gaussienne :

Générateur
de coordonnées

Exemple:

Liste abscisse = [52, 43.8, 36.8]

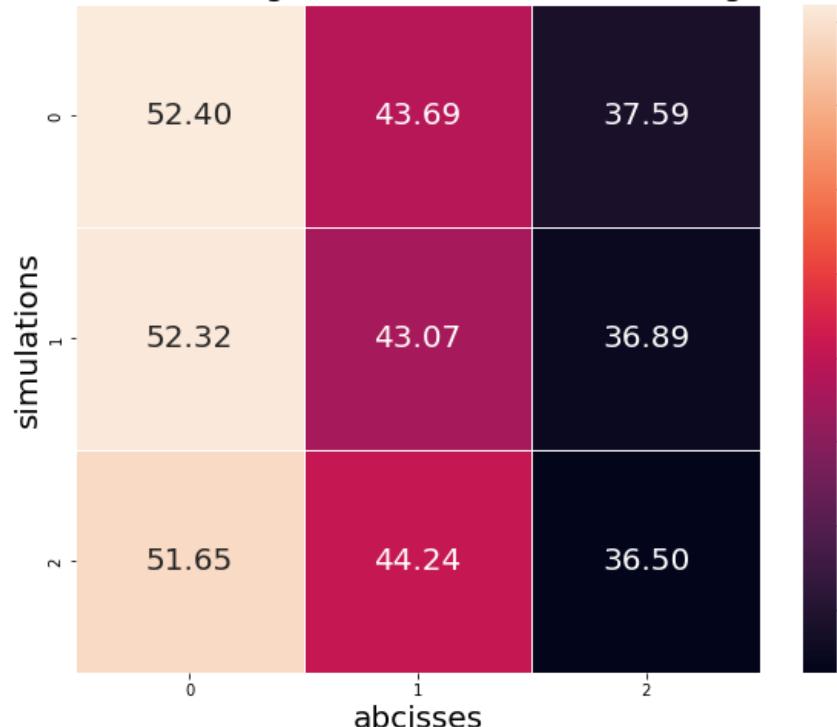
Liste ordonnées = [42.6, 38.4, 38.4]

Formule:

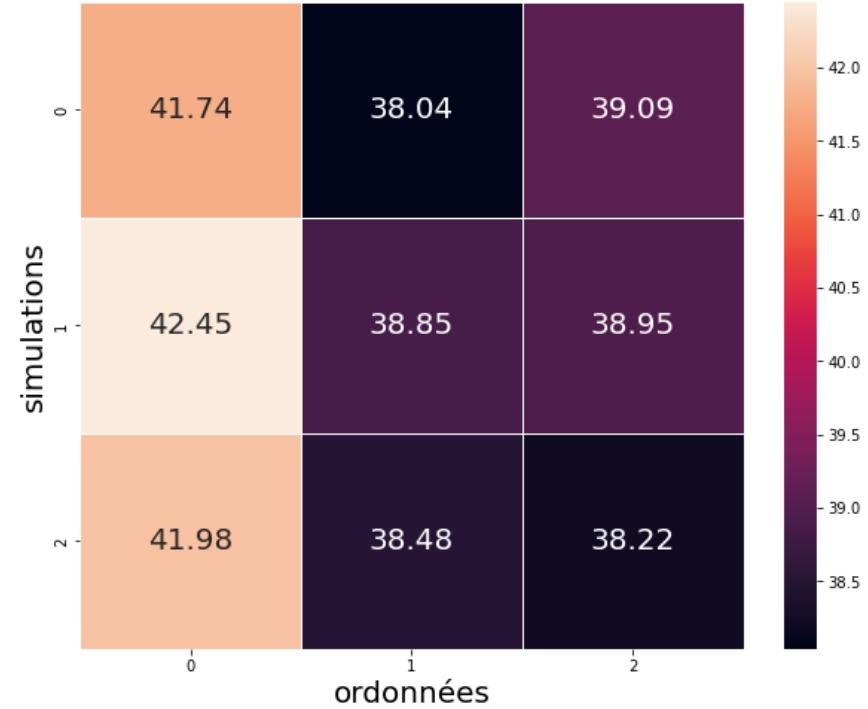
$$x' = x + N(x, \sigma_x)$$

$$y' = y + N(y, \sigma_y)$$

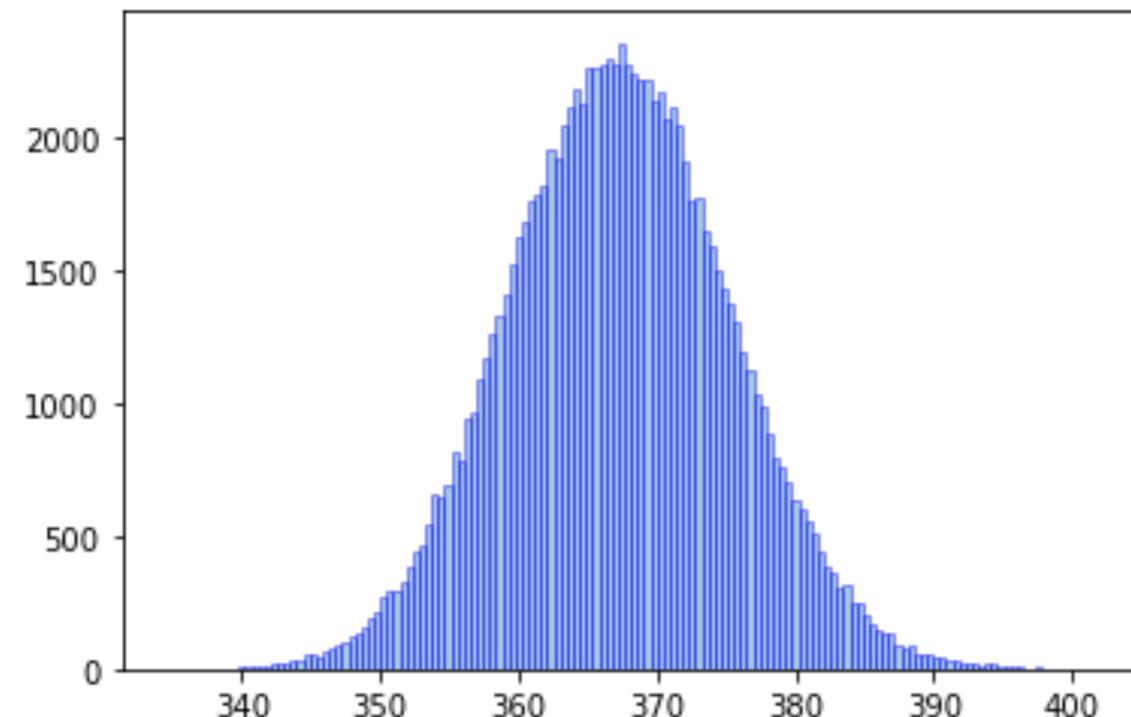
Matrice des abscisses générées avec la distribution gaussienne



Matrice des ordonnées générées avec la distribution gaussienne

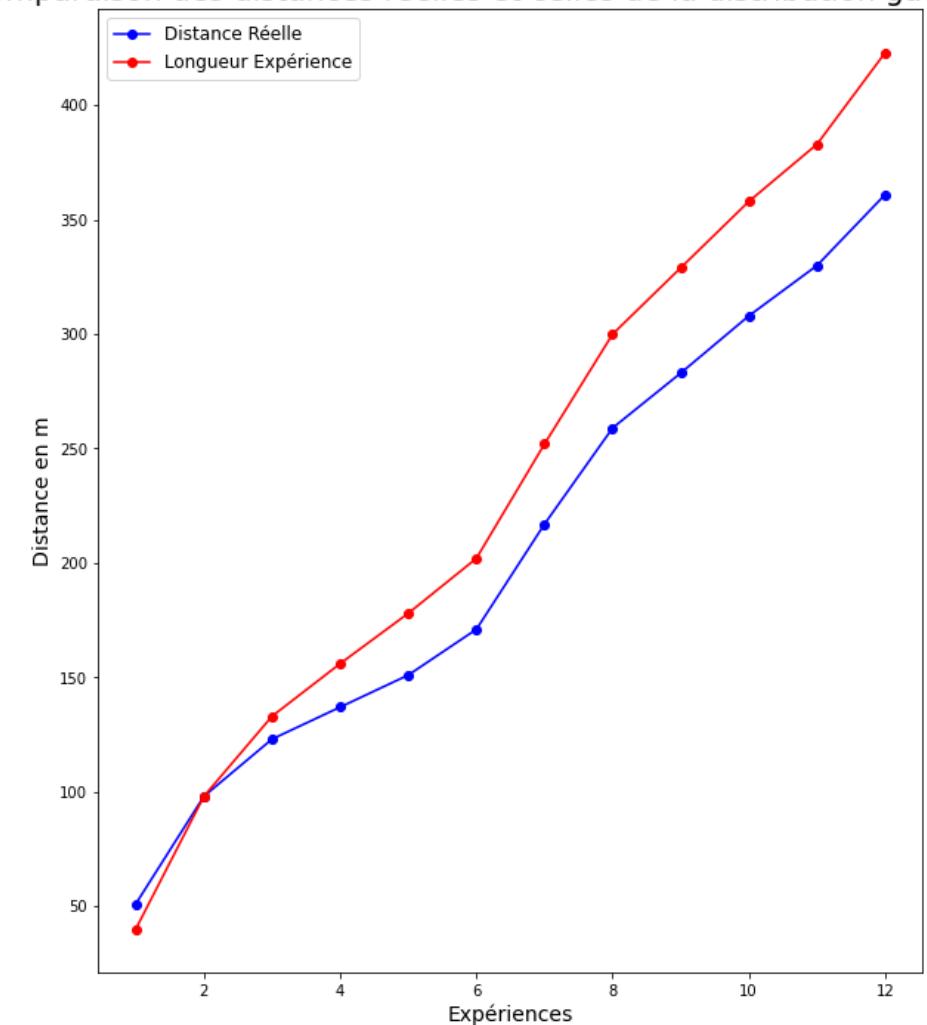


Distribution gaussienne :



Distance réelle = 361 m
Distance relevée = 349 m
Distance moyenne = 367 m

Comparaison des distances réelles et celles de la distribution gaussienne



Distribution uniforme :

Générateur
De coordonnées

Exemple:

Liste abscisse = [52, 43.8, 36.8]

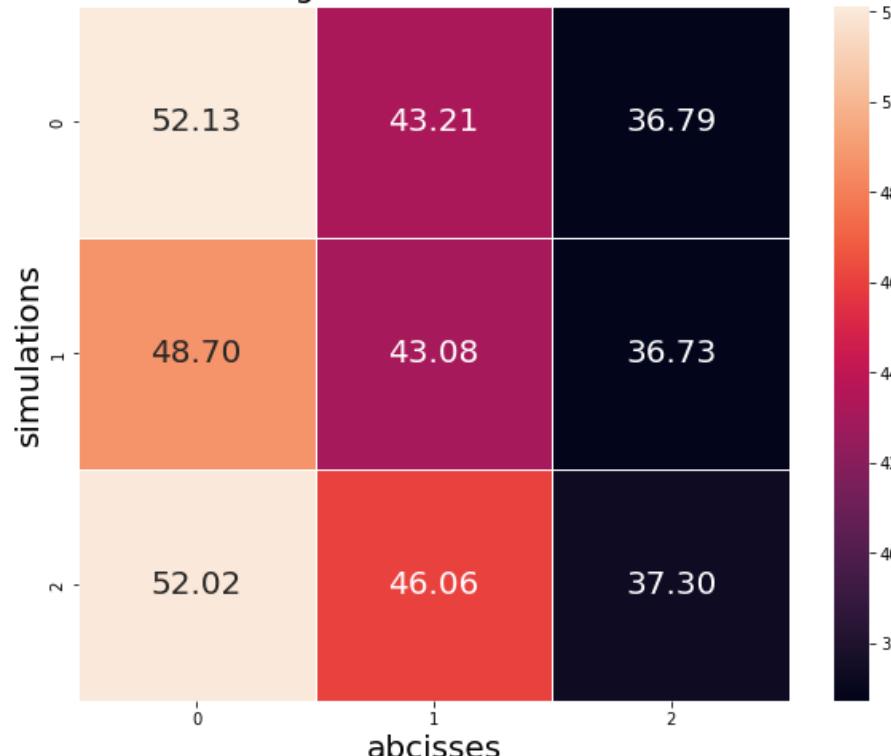
Liste ordonnées = [42.6, 38.4, 38.4]

Formule:

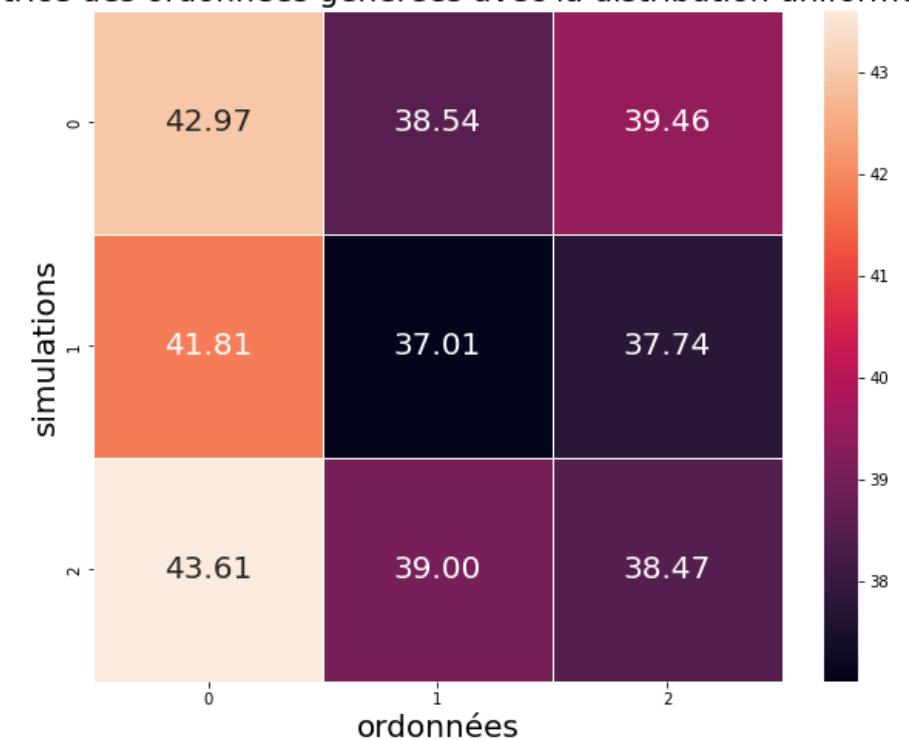
$$x' = x + N(x, \sigma_x)$$

$$y' = y + N(y, \sigma_y)$$

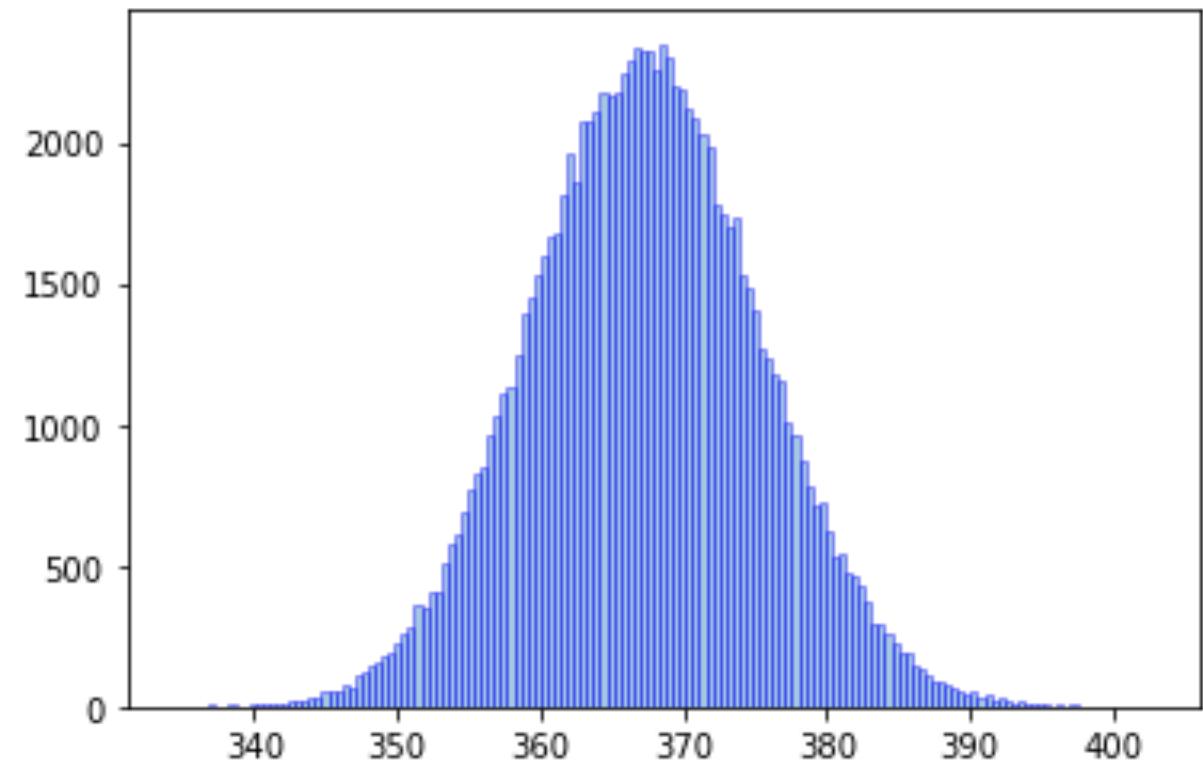
Matrice des abscisses générées avec la distribution uniforme



Matrice des ordonnées générées avec la distribution uniforme



Distribution uniforme :

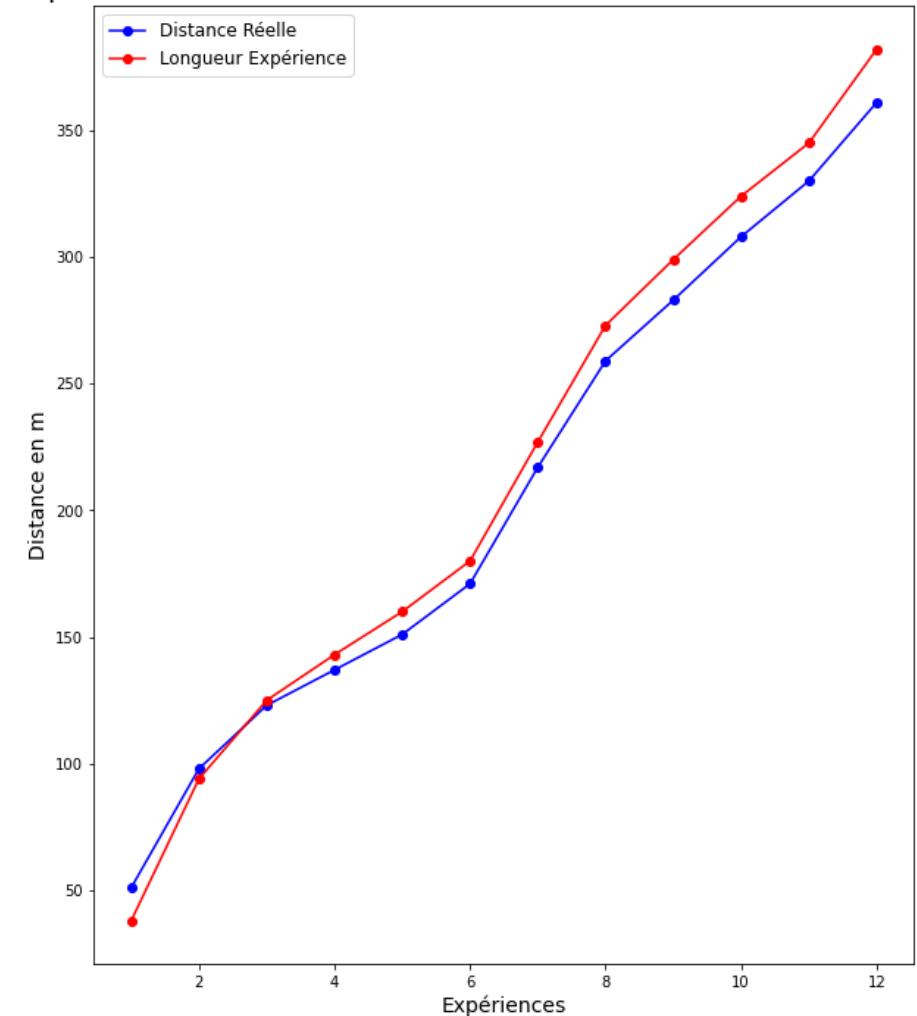


Distance réelle = 361 m

Distance relevée = 349 m

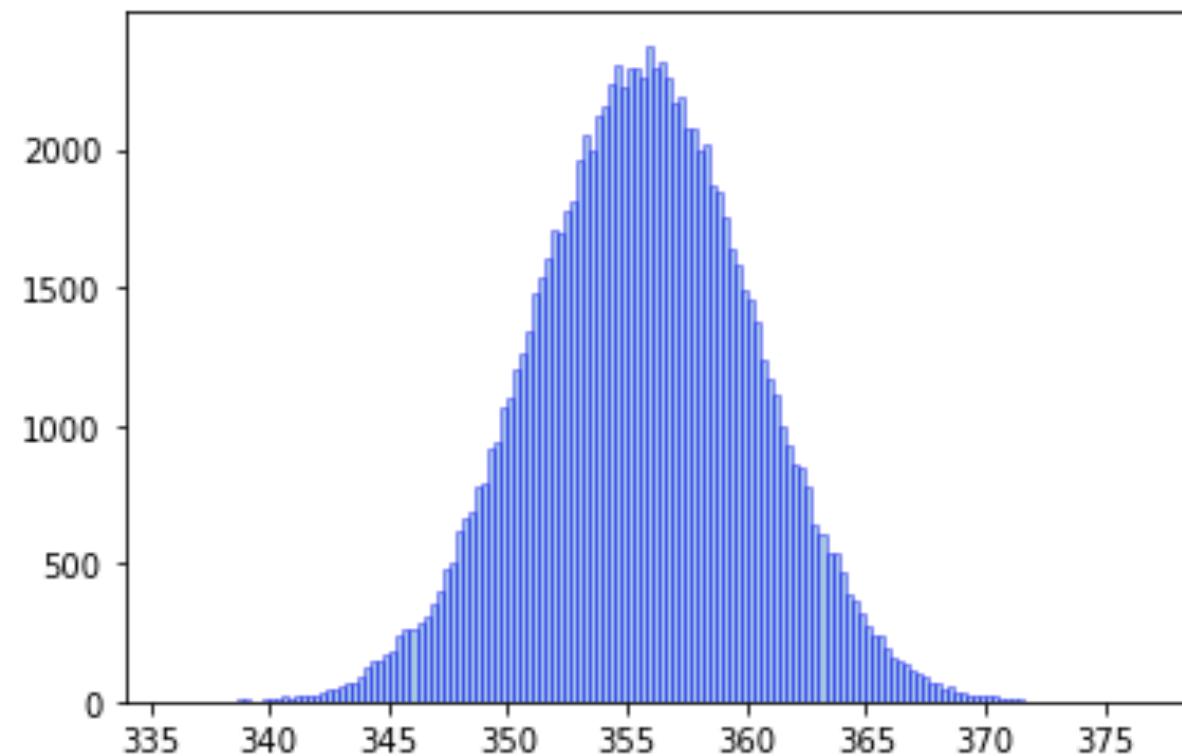
Distance moyenne = 382 m

Comparaison des distances réelles et celles de la distribution uniforme



Distribution uniforme:

Choix du nombre de points à relever par unité de temps:



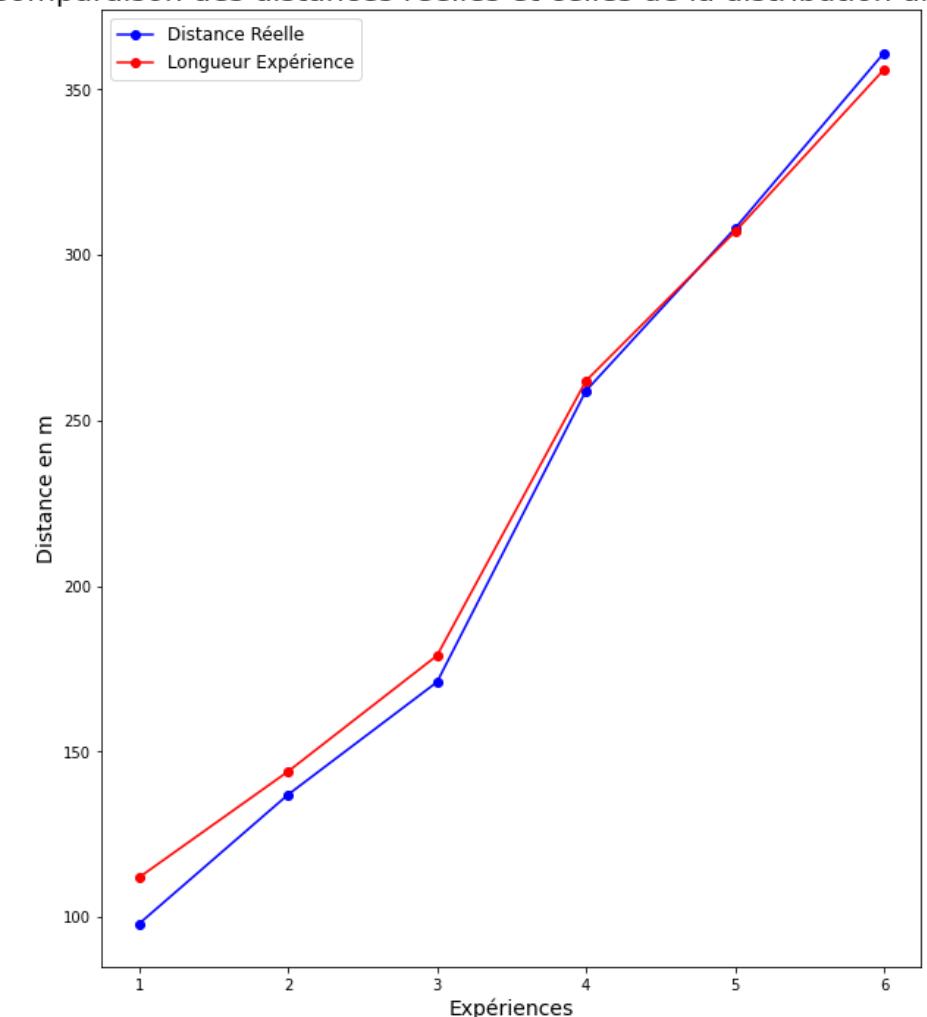
Distance réelle = 361 m

Distance relevée = 349 m

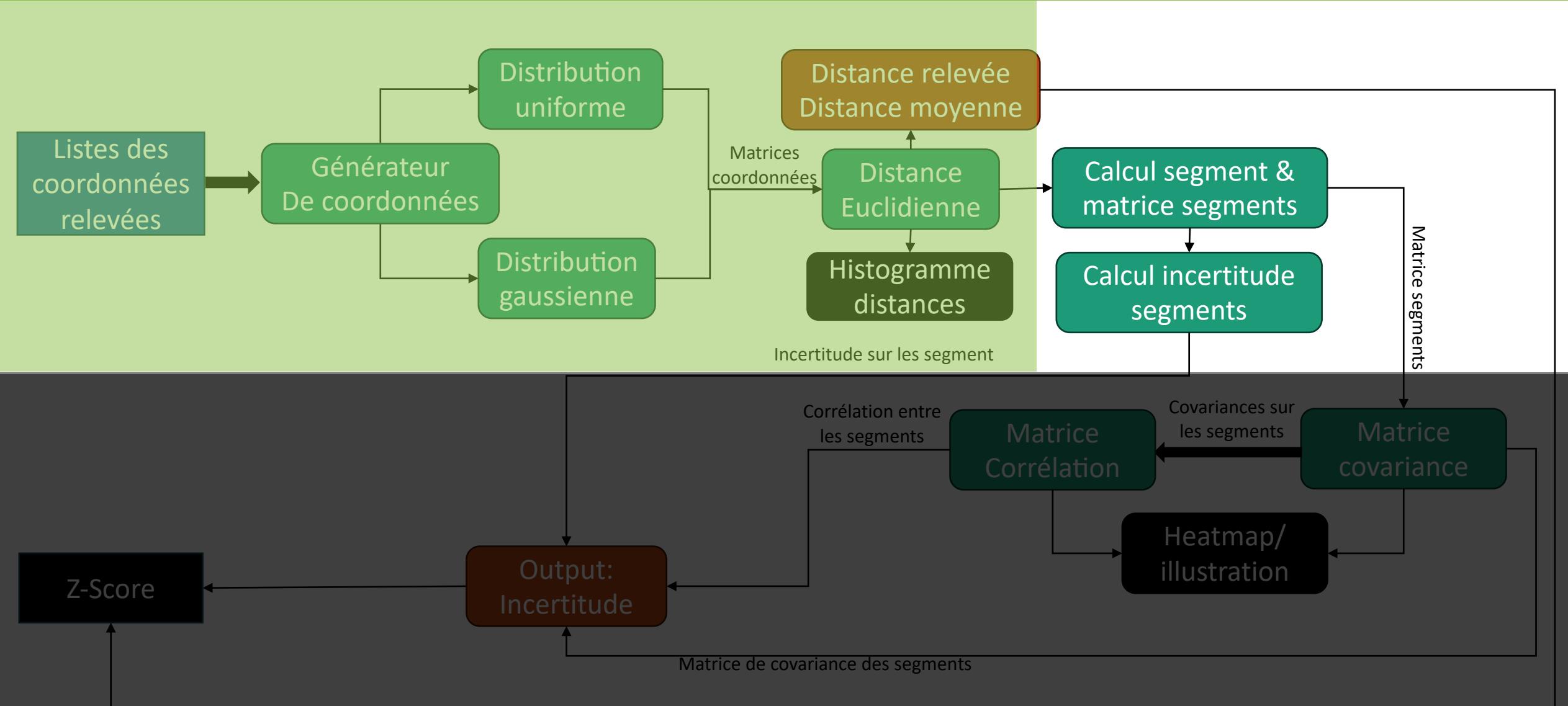
Distance moyenne = 356 m

Avec un point relevé par seconde

Comparaison des distances réelles et celles de la distribution uniforme



Etude statistique



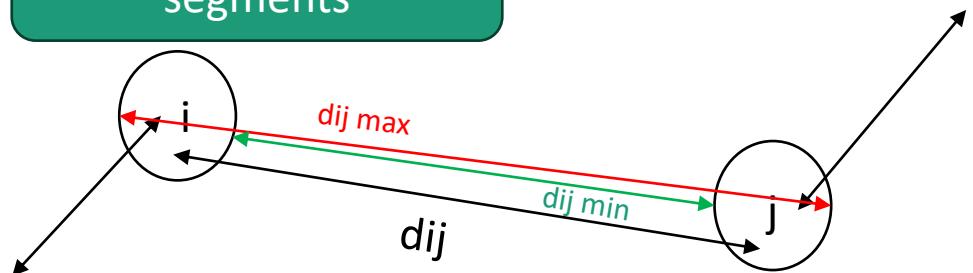
La matrice des segments & incertitude sur les segment:

Calcul segment &
matrice segments

$$d_i = \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}$$

d_{ij} = valeur de d_i à la jème simulation

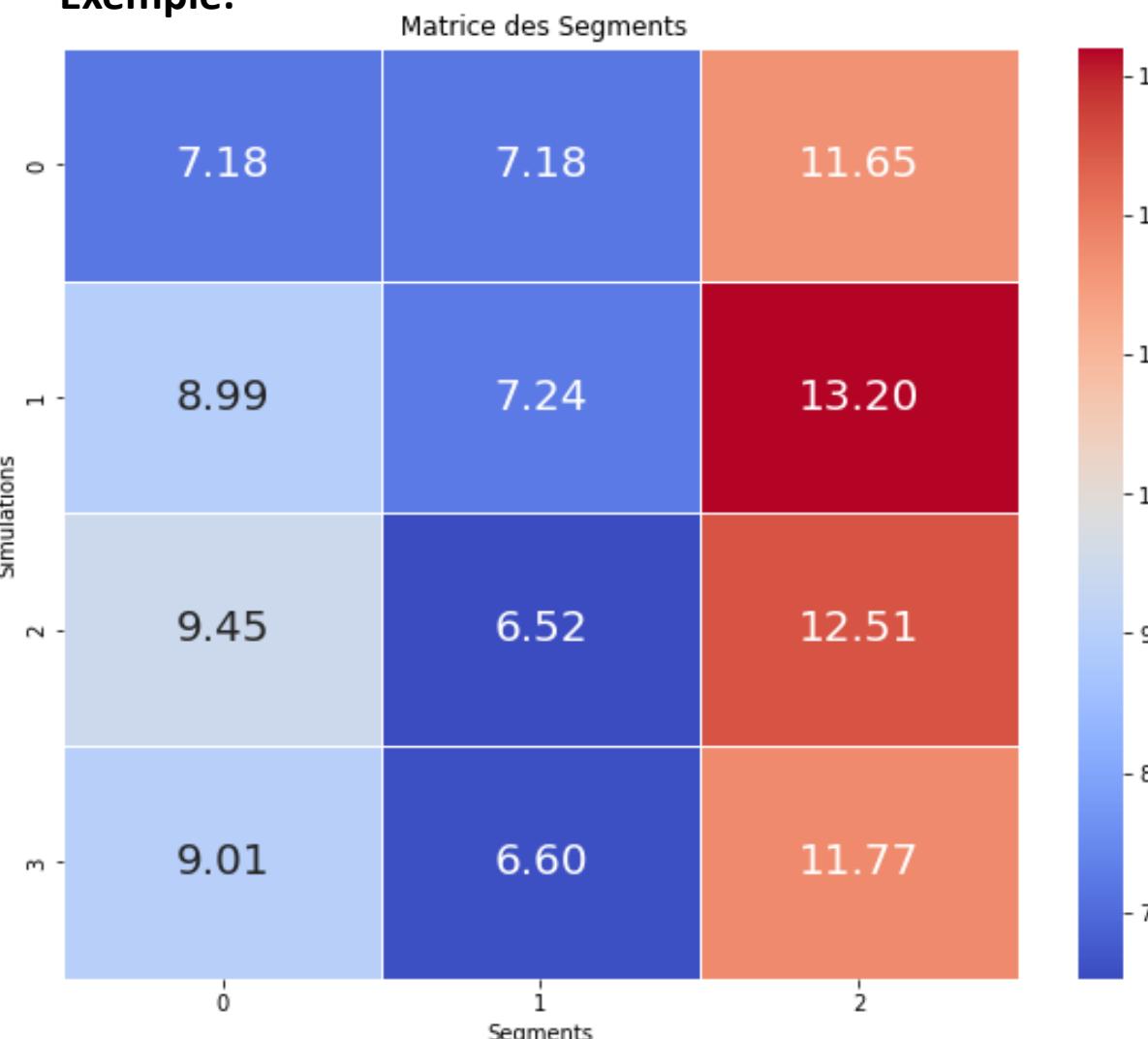
Calcul incertitude
segments



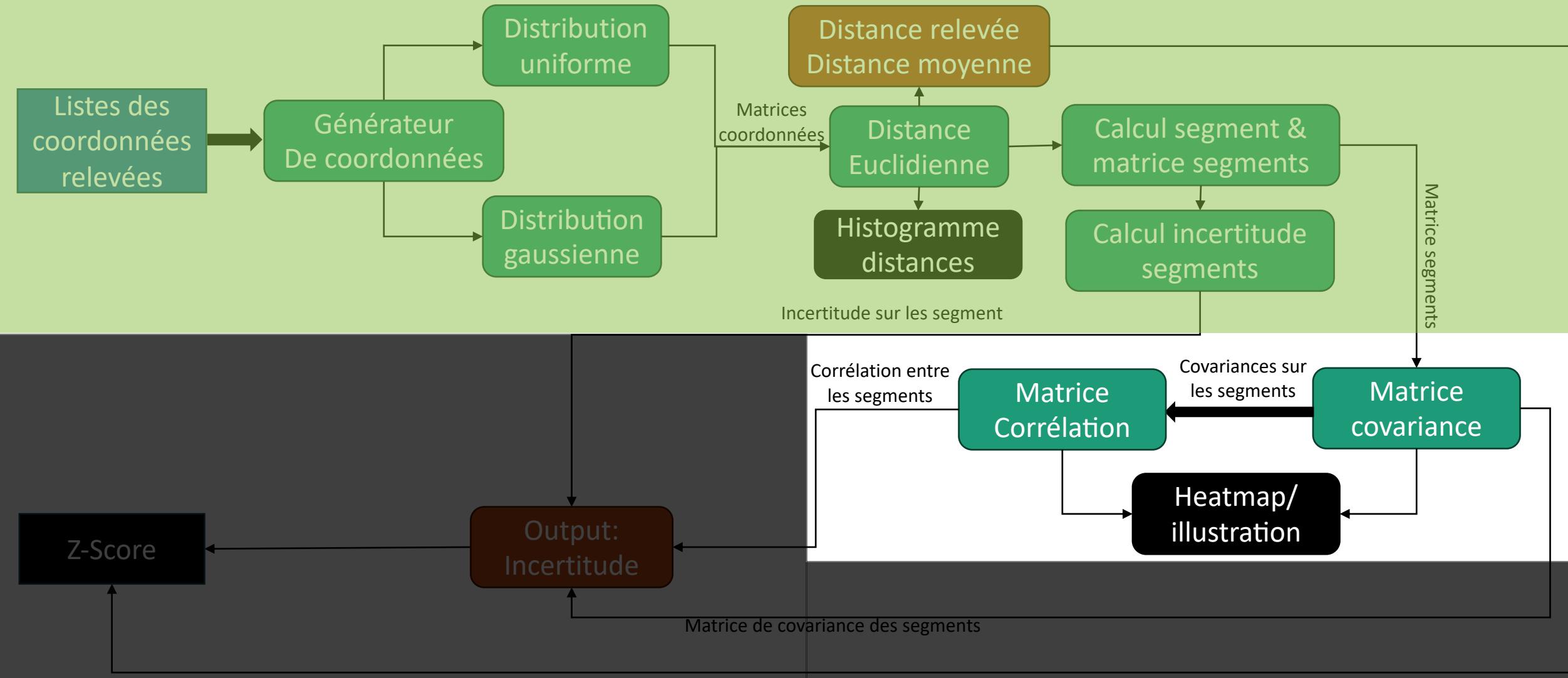
Formule de l'incertitude sur un segment:

$$\Delta dij = \sqrt{\left(\frac{\partial dij}{\partial xi} \sigma_x\right)^2 + \left(\frac{\partial dij}{\partial yi} \sigma_y\right)^2 + \left(\frac{\partial dij}{\partial xj} \sigma_x\right)^2 + \left(\frac{\partial dij}{\partial yj} \sigma_y\right)^2}$$

Exemple:



Etude statistique



Matrice de corrélation

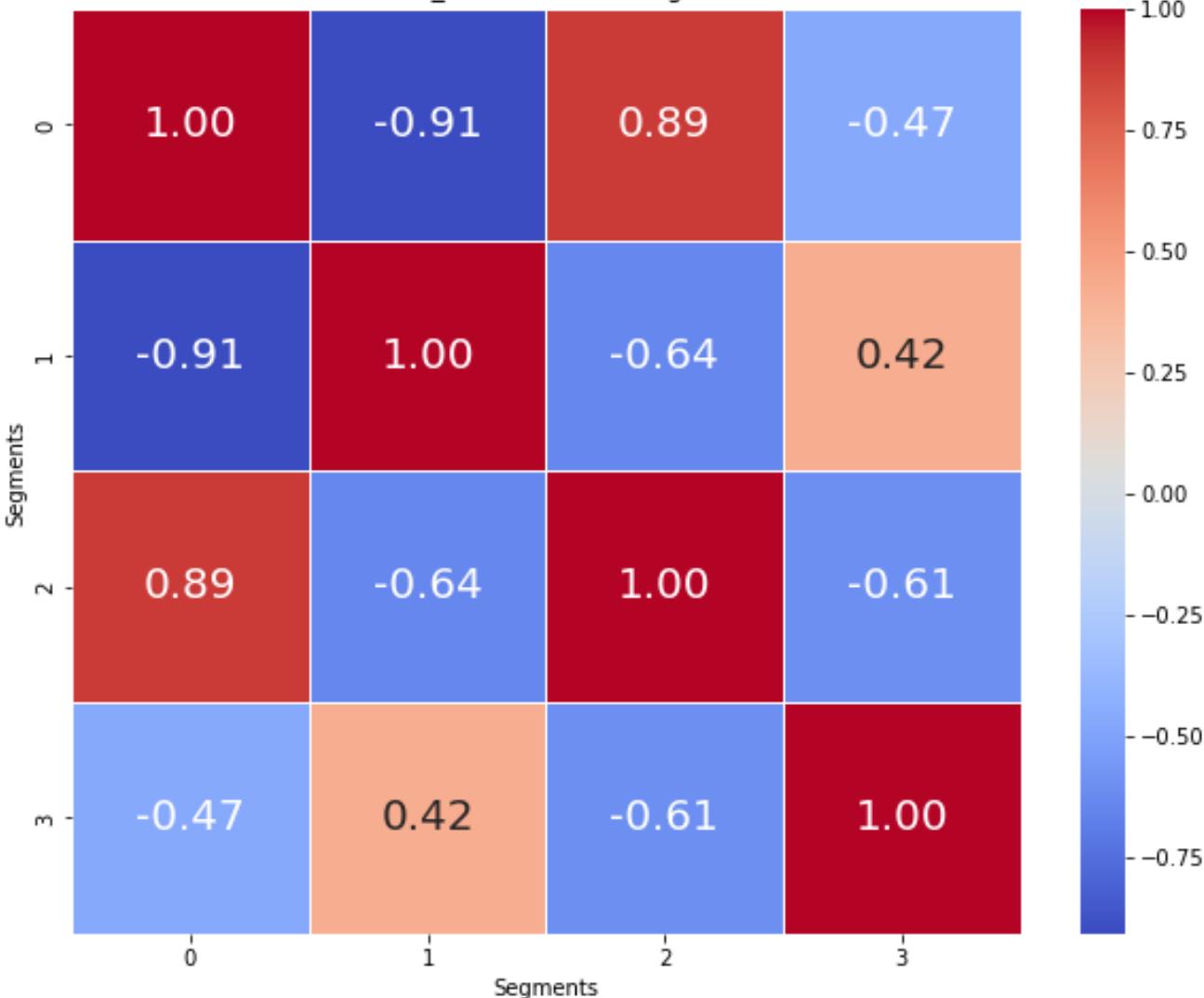
Matrice
Corrélation

Formules:

$$\rho_{ij} = \frac{Cov(di,dj)}{\Delta d_i d_j}$$

$$M_{ij} = (\rho_{ij})$$

Exemple:
Matrice_correlation des Segments



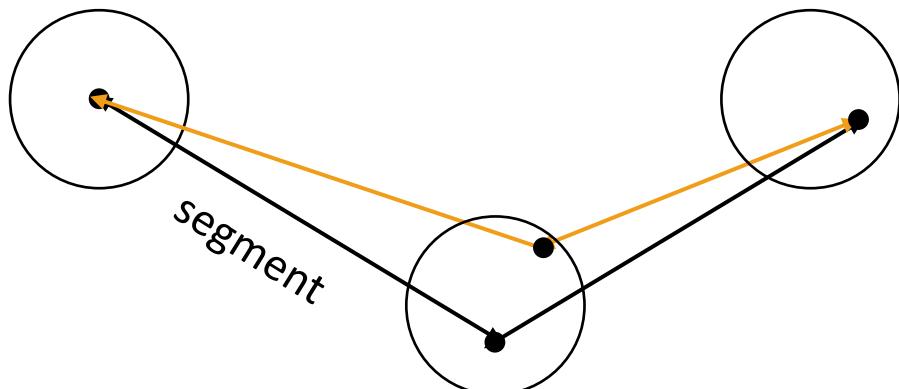
Matrice de covariance

Matrice covariance

Formule:

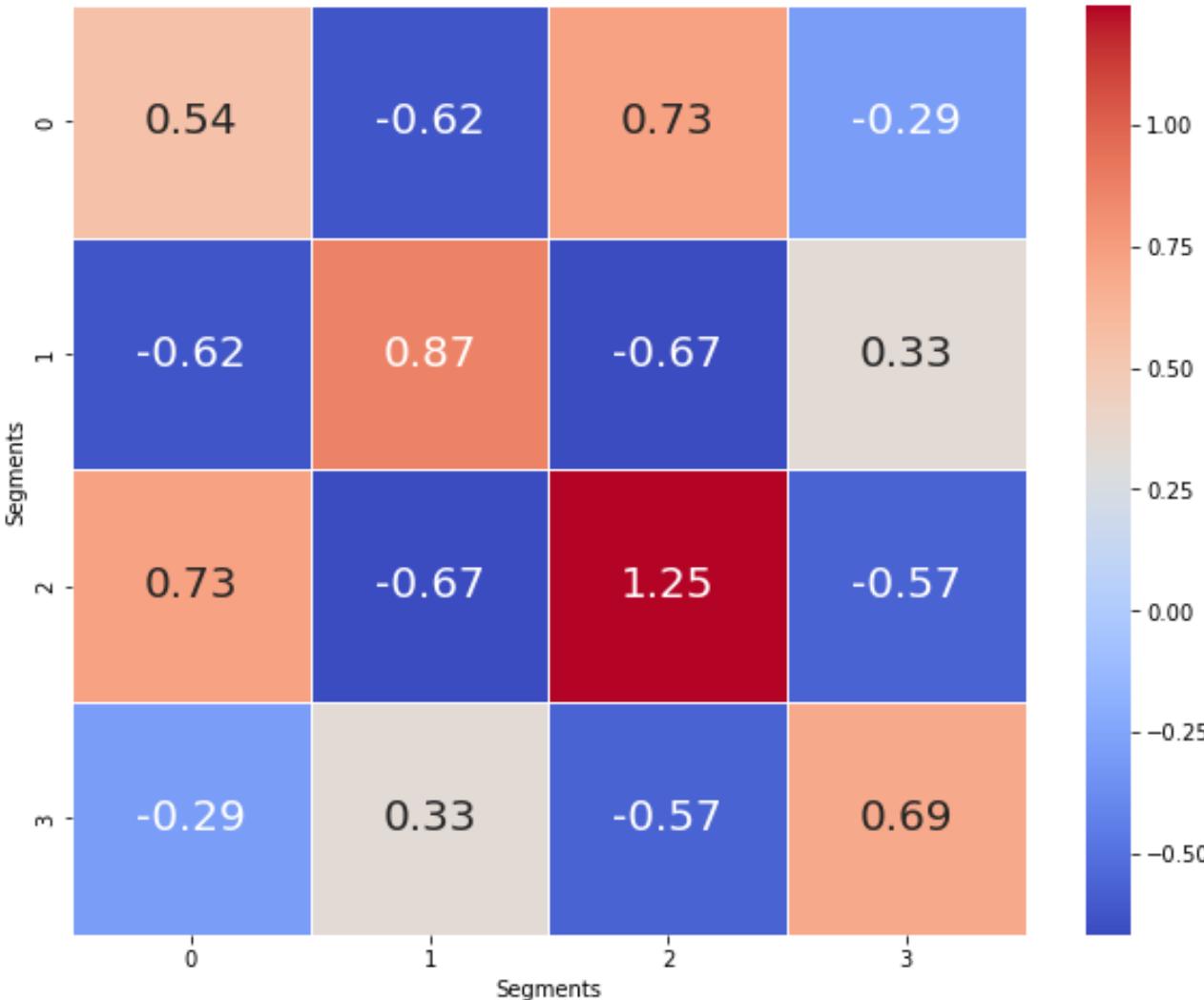
$$Cov(X, Y) = \frac{1}{N-1} \sum_{i=1}^N (X_i - X_{moy})(Y_i - Y_{moy})$$

$$\sum_{ij} = Cov(di, dj)$$

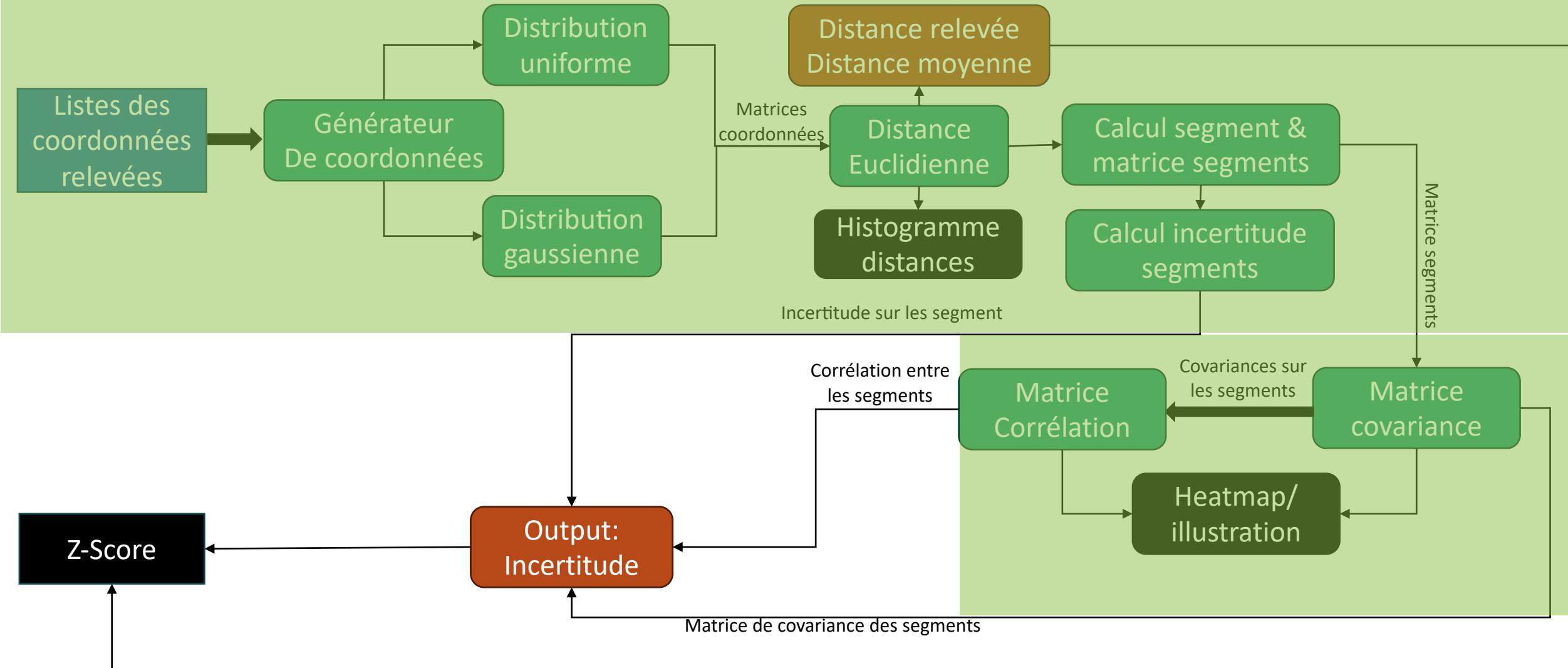


Exemple:

Matrice de Covariance des Segments



Etude statistique



Incertitude sur la distance totale:

Sortie: Incertitude

Distance totale:

Z-Score

$$\sigma_i = Mat_cov_{ii}$$

$$D = \sum_{i=1}^N d_i$$

Incertitude sur la distance totale:

$$\Delta D = \sqrt{\sum_{i=1}^N (\Delta d_i)^2 + 2 \sum_{i=1}^N \sum_{j=i+1}^N \rho_{ij} \sigma_i \sigma_j}$$

Vérification de la justesse des résultats:

$$\text{Z-score} = \frac{x_{reel} - x_{moy}}{\sqrt{\text{incertitude}}}$$

Conclusion:

Avec la distribution uniforme:

Pour 1 minute de jeu avec 60 coordonnées relevées:

Distance réelle = 361 m

Distance relevée = 349 m

Résultats obtenus:

Distance moyenne = 356 m

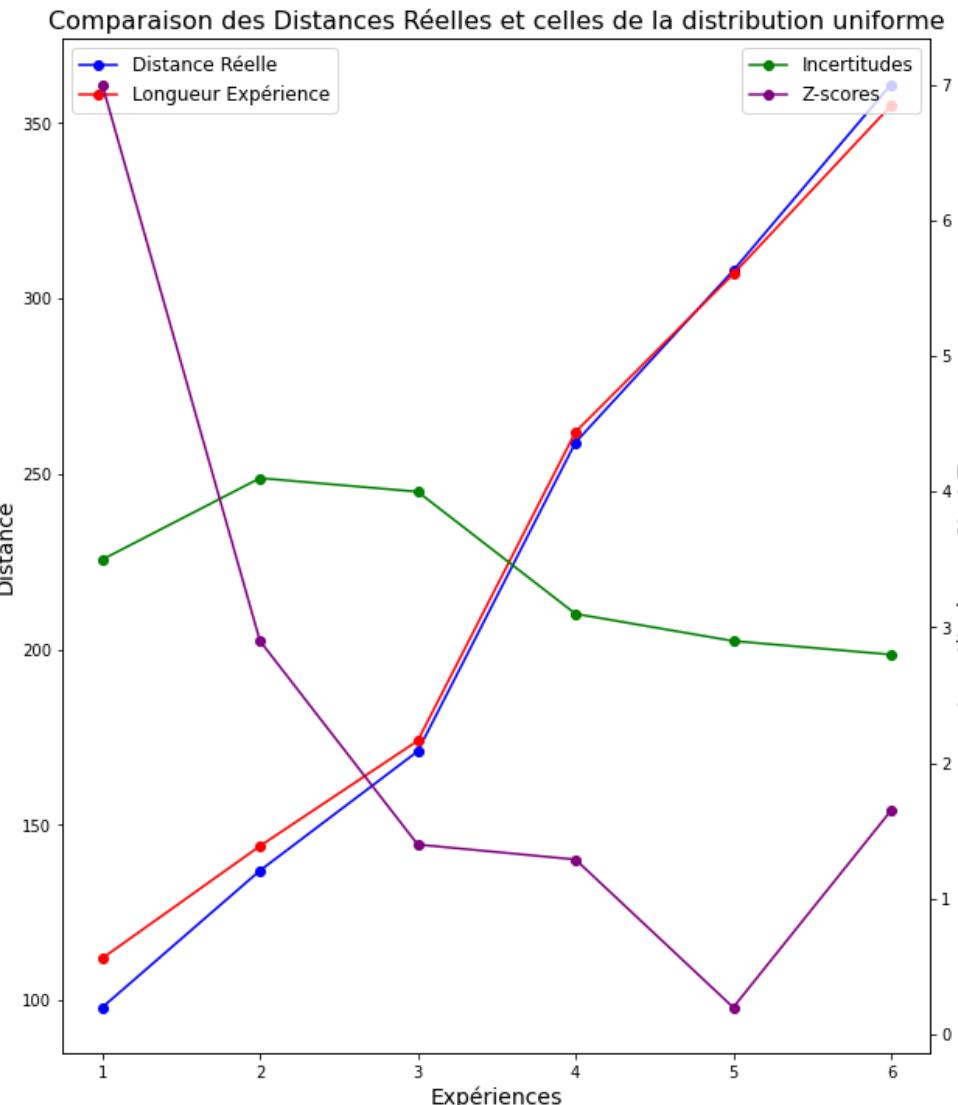
Incertitude = 10 m

Z-score = 1.65

Incertitude de 2.9%

Limites et améliorations

- Prendre en compte d'autres paramètres
- Considérer les interférences que peuvent subir les signaux
- Utilise une méthode plus adapté au déplacement des joueurs
- Contrôler les coordonnées générées
- Confronter la théorie à la réalité



Annexes:

Raspberry Pi Pico distance by coach_adrian

```
1 import machine
2 import utime
3
4 def distance_capteurs(PIN_EMETTEUR, PIN_RECEPTEUR, vitesse_signal):
5     emetteur = machine.Pin(PIN_EMETTEUR, machine.Pin.OUT)
6     recepteur = machine.Pin(PIN_RECEPTEUR, machine.Pin.IN)
7     liste_distance = []
8     try:
9         while True:
10            emetteur.high() # Emettre un signal
11            utime.sleep_ms(1) # Pendant 1 ms
12
13            emetteur.low() # Arreter le signal
14            utime.sleep(1) # Attendez 1 seconde entre les transmissions
15
16            debut = utime.ticks_us()
17            fin = utime.ticks_us()
18
19            while recepteur.value() == 0:
20                debut = utime.ticks_us()
21
22            while recepteur.value() == 1:
23                fin = utime.ticks_us()
24
25            temps_reception = utime.ticks_diff(fin, debut) / 1e6
26            distance_emetteur_recepteur = temps_reception * vitesse_signal # calcule de la distance
27            liste_distance.append(distance_emetteur_recepteur)
28            print('Distance:', distance_emetteur_recepteur)
29
30     except KeyboardInterrupt:
31         pass
32
33 # Exemple d'utilisation :
34 PIN_EMETTEUR = 17
35 PIN_RECEPTEUR = 27
36 vitesse_signal = 343 # Vitesse du son dans l'air 20°C (m/s)
37
38 distance_capteurs(PIN_EMETTEUR, PIN_RECEPTEUR, vitesse_signal)
39
```

Simulation

HC-SR04

Raspberry Pi Pico

Annexes:

The screenshot shows a PyCharm IDE window and a Spyder interface side-by-side.

PyCharm Editor: The code in the editor is as follows:

```
1 import RPi.GPIO as GPIO
2 import time
3
4 def distance_capteurs(PIN_EMETTEUR, PIN_RECEPTEUR, vitesse_signal):
5     GPIO.setmode(GPIO.BCM)
6     GPIO.setup(PIN_EMETTEUR, GPIO.OUT)
7     GPIO.setup(PIN_RECEPTEUR, GPIO.IN)
8     liste_distance = []
9     try:
10         while True:
11             GPIO.output(PIN_EMETTEUR, GPIO.HIGH) # Emettre un signal
12             time.sleep(1e-3) # Pendant 1 ms
13
14             GPIO.output(PIN_EMETTEUR, GPIO.LOW) # Arreter le signal
15             time.sleep(1) # Attendez 1 seconde entre les transmissions
16
17             debut = time.time()
18             fin = time.time()
19
20             while GPIO.input(PIN_RECEPTEUR) == GPIO.LOW:
21                 debut = time.time()
22
23             while GPIO.input(PIN_RECEPTEUR) == GPIO.HIGH:
24                 fin = time.time()
25
26             temps_reception = fin - debut
27             distance_emetteur_recepteur = temps_reception * vitesse_signal # calcule de la distance
28             liste_distance.append(distance_emetteur_recepteur)
29     return liste_distance
30
31 except KeyboardInterrupt:
32     GPIO.cleanup()
33
34 # Exemple d'utilisation :
35 PIN_EMETTEUR = 17
36 PIN_RECEPTEUR = 27
37 vitesse_signal = 343 # Vitesse du son dans l'air 200C (m/s)
38
39 distance_capteurs(PIN_EMETTEUR, PIN_RECEPTEUR, vitesse_signal)
40
41 def coordonnees_joueur(PIN_EMETTEUR, PIN_RECEPTEUR_1, PIN_RECEPTEUR_2, PIN_RECEPTEUR_3):
42     liste_distance_1 = distance_capteurs(PIN_EMETTEUR, PIN_RECEPTEUR_1, vitesse_signal)
43     liste_distance_2 = distance_capteurs(PIN_EMETTEUR, PIN_RECEPTEUR_2, vitesse_signal)
44     liste_distance_3 = distance_capteurs(PIN_EMETTEUR, PIN_RECEPTEUR_3, vitesse_signal)
45
46     liste_abcisses = []
47     liste_ordonnees = []
48     n = len(liste_distance_1)
49     L = 105 # longueur du terrain
50     l = 68 # largeur du terrain
51
52     for i in range(n):
```

Spyder Interface: The Spyder interface shows the usage documentation for the `distance_capteurs` function:

Usage
Here you can get help of any object by pressing **Cmd+I** in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Below the documentation, the Spyder console shows the Python environment:

```
Python 3.9.13 (main, Aug 25 2022, 18:29:29)
Type "copyright", "credits" or "license" for more information.
IPython 7.31.1 -- An enhanced Interactive Python.
In [1]:
```

Annexes:

The screenshot shows the PyCharm IDE interface with a Python script named `rasberry.py` open in the editor. The script uses the RPi.GPIO library to measure distances between three ultrasonic sensors connected to pins 17, 27, and 22. It calculates the coordinates of three points on a 105x68 grid based on the measured distances.

```
12 time.sleep(1e-3) # Pendant 1 ms
13
14 GPIO.output(PIN_EMETTEUR, GPIO.LOW) # Arreter le signal
15 time.sleep(1) # Attendez 1 seconde entre les transmissions
16
17 debut = time.time()
18 fin = time.time()
19
20 while GPIO.input(PIN_RECEPTEUR) == GPIO.LOW:
21     debut = time.time()
22
23 while GPIO.input(PIN_RECEPTEUR) == GPIO.HIGH:
24     fin = time.time()
25
26 temps_reception = fin - debut
27 distance_emetteur_recepteur = temps_reception * vitesse_signal # calcule de la distance
28 liste_distance.append(distance_emetteur_recepteur)
29 return liste_distance
30
31 except KeyboardInterrupt:
32     GPIO.cleanup()
33
34 # Exemple d'utilisation :
35 PIN_EMETTEUR = 17
36 PIN_RECEPTEUR = 27
37 vitesse_signal = 343 # Vitesse du son dans l'air 200C (m/s)
38
39 distance_capteurs(PIN_EMETTEUR, PIN_RECEPTEUR, vitesse_signal)
40
41 def coordonnees_joueur(PIN_EMETTEUR, PIN_RECEPTEUR_1, PIN_RECEPTEUR_2, PIN_RECEPTEUR_3):
42     liste_distance_1 = distance_capteurs(PIN_EMETTEUR, PIN_RECEPTEUR_1, vitesse_signal)
43     liste_distance_2 = distance_capteurs(PIN_EMETTEUR, PIN_RECEPTEUR_2, vitesse_signal)
44     liste_distance_3 = distance_capteurs(PIN_EMETTEUR, PIN_RECEPTEUR_3, vitesse_signal)
45
46     liste_abscisses = []
47     liste_ordonnees = []
48     n = len(liste_distance_1)
49     L = 105 # longueur du terrain
50     l = 68 # largeur du terrain
51
52     for i in range(n):
53         x = (liste_distance_1[i]**2 - liste_distance_3[i]**2 + L**2)/2*L
54         y = (liste_distance_1[i]**2 - liste_distance_2[i]**2 + l**2)/2*l
55
56         liste_abscisses.append(x)
57         liste_ordonnees.append(y)
58
59     return liste_abscisses, liste_ordonnees
60
61
62
```

The PyCharm interface includes tabs for `distance.py`, `distance1.py`, `rasberry.py*`, `untitled0.py*`, and `untitled1.py*`. The status bar at the bottom indicates the environment is `conda: base (Python 3.9.13)`.

On the right, the Spyder application is open, showing the `Console` tab with the following output:

```
Python 3.9.13 (main, Aug 25 2022, 18:29:29)
Type "copyright", "credits" or "license" for more information.
IPython 7.31.1 -- An enhanced Interactive Python.
In [1]:
```

The Spyder interface also includes tabs for `Source`, `Console`, and `Object`. A help panel titled `Usage` provides information on getting help for objects.

Annexes:

The screenshot shows a Python IDE interface with a dark theme. The top bar displays the project name "Projet TIPE 2024" and the file "TIPE > illustration.py". The left sidebar includes sections for "Project", "Structure", "Bookmarks", and "Database". The main area contains the source code for "illustration.py".

```
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
import sys
import random as rd
from ui_fenetre import Ui_MainWindow

nombre_points = 12
nombre_simulations = 10

class fenetreLignes(QMainWindow,Ui_MainWindow):
    def __init__(self,parent=None):
        super().__init__(parent)
        self.setupUi(self)
        self.setGeometry(600,100,1900,1000)
        self.canvas=ZoneG(self)
        self.canvas.setGeometry(450,100,853,567)

class ZoneG(QWidget):
    def __init__(self,parent=None): ...

    def paintEvent(self, QPaintEvent):
        p=QPainter(self)
        p.drawPixmap(0,0,self.qp)

    def generateur_coordonnees_modele_1(self, liste_abcisses, liste_ordonnees, nombre_points, nombre_simulations):
        matrice_abcisses = [] # Initialisation de l'ensemble des listes des abcisses
        matrice_ordonnees = [] # Initialisation de l'ensemble des listes des ordonnees

        for _ in range(nombre_simulations):
            matrice_abcisses.append([])
            matrice_ordonnees.append([])

        for i in range(nombre_points): # Iteration pour chaque coordonnee
            x0 = liste_abcisses[i] # Recuperation de la ieme abcisse
            y0 = liste_ordonnees[i] # Recuperation de la ieme ordonnee

            for j in range(nombre_simulations): # On releve n points coordonnes autour de chaque point
                x1 = rd.uniform(x0 - 1, x0 + 1) # Aleatoirement
                y1 = rd.uniform(y0 - 1, y0 + 1)

                matrice_abcisses[j].append(x1) # Ajout à la ieme liste des abcisses
                matrice_ordonnees[j].append(y1) # Ajout à la ieme liste des ordonnees
```

Annexes:

The screenshot shows a Python code editor interface with the following details:

- Project View:** Shows the project structure with files: illustration.py, ui_fenetre.py, and distance.py.
- Code Editor:** The main pane displays the content of `illustration.py`. The code uses the `PyQt5` library to draw points on a window. It defines two functions:
 - `joueur_reel`: A method that iterates through points to draw ellipses and connecting lines.
 - `main`: The entry point of the application, creating a window and running the application loop.
- Toolbars and Status Bar:** Standard Qt-style toolbars are visible at the top. The status bar at the bottom shows the file path (`fenetreLignes > __init__.py`), version control information (Version Control, TODO, Problems, Terminal, Python Packages, Python Console), and system details (Event Log, 14:27, LF, UTF-8, 4 spaces, Python 3.10).

Annexes:

The screenshot shows the PyCharm IDE interface with a Python script named `distance.py` open. The script performs various operations including importing libraries, defining lists of distances, and generating random coordinates for points on a football field. A right-click context menu is visible over the code at line 38, with options like 'Copy', 'Cut', 'Paste', etc.

```
1 import matplotlib.pyplot as plt
2 import seaborn as sns
3 import random as rd
4 import numpy as np
5
6
7 liste_distance_reelle1 = [51, 98, 123, 137, 151, 171, 217, 259, 283, 308, 330, 361]
8 lu = [38, 94, 125, 143, 160, 180, 227, 273, 299, 324, 345, 382]
9 lg = [40, 98, 133, 156, 178, 202, 252, 300, 329, 358, 383, 423]
10 lr = []
11 distance_reelle = 51
12 nombre_points = 5
13 nombre_simulations = 5
14
15 sigma_x = 1
16 sigma_y = 1
17 sigma_rayleigh = np.sqrt((sigma_x**2 + sigma_y**2) / 2)
18
19 liste_abcisses1 = [52, 43.8, 36.8, 42.8, 49.6, 55.6, 75.2, 63.2, 46.8, 36.6, 28.4, 29.2, 29.8, 26.4, 27.4, 31.0, 33.2, 33.8, 33.4, 32.4,
20 liste_ordonnees1 = [42.6, 38.4, 38.4, 48.6, 56.8, 66.4, 58.2, 55.8, 57.2, 50.6, 49.8, 50.8, 53.4, 52.4, 50.4, 52.2, 54.4, 56.4, 56.8, 55.
21
22 liste_abcisses2 = [52.8, 52.0, 49.6, 43.8, 40.4, 38.4, 37.2, 36.6, 38.8, 42.8, 46.6, 49.8, 51.8, 56.0, 61.0, 63.4, 63.8, 63.2, 51.4, 47.0
23 liste_ordonnees2 = [44.4, 42.6, 39.2, 38.2, 34.2, 35.2, 34.4, 38.4, 43.6, 48.8, 52.0, 56.8, 61.8, 66.6, 60.4, 58.2, 56.0, 55.8, 60.4, 57.
24
25 liste_abcisses = liste_abcisses1[:5]
26 liste_ordonnees = liste_ordonnees1[:5]
27
28 def coordonnees(nombre_points):
29     # Taille du terrain de foot
30     longueur = 105
31     largeur = 68
32     dmax = 10
33
34     # Initialiser une liste pour stocker les coordonnées des points
35     liste_abcisses = []
36     liste_ordonnees = []
37     x0, y0 = 50, 25
38     x = rd.uniform(max(0, x0 - dmax), min(longueur, x0 + dmax))
39     y = rd.uniform(max(0, y0 - dmax), min(largeur, y0 + dmax))
40
41     # Placer des points aléatoires sur le terrain
42     for _ in range(nombre_points):
43         x0, y0 = x, y
44         x = rd.uniform(max(0, x0 - dmax), min(longueur, x0 + dmax))
45         y = rd.uniform(max(0, y0 - dmax), min(largeur, y0 + dmax))
46         liste_abcisses.append(x)
47         liste_ordonnees.append(y)
48
49 return liste_abcisses, liste_ordonnees
```

The PyCharm interface includes tabs for `distance.py`, `distance1.py`, `rasberry.py*`, `untitled0.py*`, and `untitled1.py*`. The status bar at the bottom shows conda: base (Python 3.9.13), Completions: conda(base), LSP: Python, Line 37, Col 20, and other system information.

The right side of the interface features a `Console` tab with a help panel titled "Usage". It provides instructions on how to get help for objects using `Cmd+I`. Below the help panel is a "New to Spyder? Read our tutorial" link. The console window displays the Python version and license information.

Annexes:

The screenshot shows the PyCharm IDE interface with the following details:

- File Bar:** /Users/nouroudine/PycharmProjects/Projet TIPE 2024/TIPE
- Editor Tab:** distance.py
- Code Content:** Python script for generating random coordinates and calculating Euclidean distance.
- Right Panel:**
 - Usage:** Help for any object by pressing Cmd+I.
 - Help:** Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Help.
 - New to Spyder?** Read our tutorial.
 - Console:** Shows Python version and license information.
 - Variable Explorer:** Shows variables in the current scope.
 - File Analysis:** Shows code analysis results.
- Status Bar:** conda: base (Python 3.9.13) Completions: conda(base) LSP: Python Line 37, Col 20 UTF-8 CRLF RW Mem 91%

```
50 # Décommenter cette ligne pour générer des coordonnées aléatoires
51 liste_abcisses, liste_ordonnees = coordonnees(nombre_points)
52
53 # Affichage des coordonnées
54 plt.plot(liste_abcisses, liste_ordonnees, 'bo')
55 plt.plot(liste_abcisses, liste_ordonnees, 'r-')
56 plt.xlim(0, 110)
57 plt.ylim(0, 70)
58 plt.show()
59
60 def generateur_coordeunes_modele_1(liste_abcisses, liste_ordonnees, nombre_points, nombre_simulations):
61     matrice_abcisses = [] # Initialisation de l'ensemble des listes des abcisses
62     matrice_ordonnees = [] # Initialisation de l'ensemble des listes des ordonées
63
64     for _ in range(nombre_simulations):
65         matrice_abcisses.append([])
66         matrice_ordonnees.append([])
67
68     for i in range(nombre_points): # Iteration pour chaque coordonnée
69         x0 = liste_abcisses[i] # Recuperation de la ième abcisse
70         y0 = liste_ordonnees[i] # Recuperation de la ième ordonnée
71
72         for j in range(nombre_simulations): # On releve n points coordonnes autour de chaque point
73             x1 = rd.uniform(x0 - sigma_x, x0 + sigma_x) # Aleatoirement
74             y1 = rd.uniform(y0 - sigma_y, y0 + sigma_y)
75
76             matrice_abcisses[j].append(x1) # Ajout à la jème liste des abcisses
77             matrice_ordonnees[j].append(y1) # Ajout à la jème liste des ordonées
78
79     return matrice_abcisses, matrice_ordonnees
80
81 matrice_abcisses, matrice_ordonnees = generateur_coordeunes_modele_1(liste_abcisses, liste_ordonnees, nombre_points, nombre_simulations)
82
83 plt.figure(figsize=(10, 8))
84 sns.heatmap(matrice_abcisses, annot=True, fmt='%.2f', annot_kws={"size": 20}, linewidths=0.5)
85 plt.title('Matrice des abcisses générées avec la distribution uniforme')
86 plt.xlabel('abcisses')
87 plt.ylabel('simulations')
88 plt.show()
89
90 plt.figure(figsize=(10, 8))
91 sns.heatmap(matrice_ordonnees, annot=True, fmt='%.2f', annot_kws={"size": 20}, linewidths=0.5)
92 plt.title('Matrice des ordonnées générées avec la distribution uniforme')
93 plt.xlabel('ordonnées')
94 plt.ylabel('simulations')
95 plt.show()
96
97 def distance_euclidienne(liste_abcisses, liste_ordonnees):
```

Annexes:

The screenshot shows the PyCharm IDE interface with the following details:

- File Bar:** /Users/nouroudine/PycharmProjects/Projet TIPE 2024/TIPE
- Editor Tab:** distance.py
- Code Content:** Python script for calculating Euclidean distances and generating histograms.
- Right Panel:**
 - Usage:** Help for objects via Cmd+I.
 - Help:** Help can be shown automatically after writing a left parenthesis next to an object.
 - New to Spyder?** Read our tutorial.
- Console Tab:** Cons... (Python 3.9.13, IPython 7.31.1)
- Bottom Status Bar:** conda: base (Python 3.9.13), Completions: conda(base), LSP: Python, Line 37, Col 20, UTF-8, CRLF, RW, Mem 91%

```
97 def distance_euclidienne(liste_abcisses, liste_ordonnees):
98     n = len(liste_abcisses)
99     distance = 0
100    for i in range(n-1):
101        x1 = liste_abcisses[i]
102        x2 = liste_abcisses[i+1]
103        y1 = liste_ordonnees[i]
104        y2 = liste_ordonnees[i+1]
105
106        distance += np.sqrt((x1 - x2)**2 + (y1 - y2)**2)
107    return distance
108
109 # Simulation de la distance euclidienne pour chaque simulation
110 liste_distance = []
111 for i in range(nombre_simulations):
112     liste_distance.append(distance_euclidienne(matrice_abcisses[i], matrice_ordonnees[i]))
113
114 # Affichage de l'histogramme des distances
115 plt.hist(liste_distance, bins='auto', alpha=0.4, edgecolor='blue')
116 plt.show()
117
118 distance_moyenne = np.mean(liste_distance)
119 distance_relevee = distance_euclidienne(liste_abcisses, liste_ordonnees)
120
121 print("distance reelle: ", distance_reelle)
122 print("Distance_relevee : ", distance_relevee)
123 print('Distance moyenne = : ', distance_moyenne)
124
125 def calcul_segments(liste_abcisses, liste_ordonnees):
126     liste_segments = []
127     n = len(liste_abcisses)
128
129     for i in range(n-1):
130         liste_segments.append(distance_euclidienne(liste_abcisses[i:i+2], liste_ordonnees[i:i+2]))
131
132     return liste_segments
133
134 liste_segments = calcul_segments(liste_abcisses, liste_ordonnees)
135
136 def matrice_segment(matrice_abcisses, matrice_ordonnees):
137     n = len(matrice_abcisses)
138     matrice_segments = []
139     for i in range(n):
140         segments = calcul_segments(matrice_abcisses[i], matrice_ordonnees[i])
141         matrice_segments.append(segments)
142
143     return matrice_segments
144
```

Annexes:

The screenshot shows a PyCharm IDE interface with the following details:

- File Bar:** /Users/nouroudine/PycharmProjects/Projet TIPE 2024/TIPE
- Editor:** distance.py (active tab)
- Code Content:** Python script for generating a heatmap and performing statistical calculations on segments.

```
144 matrice_segments = matrice_segment(matrice_abcisses, matrice_ordonnees)
145
146 plt.figure(figsize=(10, 8))
147 sns.heatmap(matrice_segments, annot=True, cmap='coolwarm', fmt='.2f', annot_kws={"size": 20}, linewidths=0.5)
148 plt.title(' Matrice des Segments ')
149 plt.xlabel('Segments')
150 plt.ylabel('Simulations')
151 plt.show()
152
153
154 #print("Matrice des segments:", matrice_segment(matrice_abcisses, matrice_ordonnees))
155
156 def incertitude_segment(sigma_x, sigma_y, liste_abcisses, liste_ordonnees, liste_segments):
157     n = len(liste_segments)
158     liste_incertitudes = []
159
160     for i in range(n):
161         x1 = liste_abcisses[i]
162         x2 = liste_abcisses[i+1]
163         y1 = liste_ordonnees[i]
164         y2 = liste_ordonnees[i+1]
165         d = liste_segments[i]
166
167         incertitude_segment = np.sqrt(2) * np.sqrt(((x1 - x2) / d * sigma_x)**2 + ((y1 - y2) / d * sigma_y)**2)
168
169         liste_incertitudes.append(incertitude_segment)
170
171     return liste_incertitudes
172 incertitudes_segments = incertitude_segment(sigma_x, sigma_y, liste_abcisses, liste_ordonnees, liste_segments)
173
174
175 #print("Incertitudes des segments:", incertitude_segment(sigma_x, sigma_y, liste_abcisses, liste_ordonnees, liste_segments))
176
177 def moyenne_colonnes(matrice):
178     n = len(matrice)
179     m = len(matrice[0])
180     moyennes_colonne = []
181
182     for j in range(m):
183         colonne = [matrice[i][j] for i in range(n)]
184         moyenne_colonne = np.mean(colonne)
185         moyennes_colonne.append(moyenne_colonne)
186
187     return moyennes_colonne
188
189 def matrice_covariance(liste_segments, matrice_segments):
190     n = len(matrice_segments[0])
191     mat_cov = np.zeros((n, n))
192
193
194
195
196
197
198
199
200
201
202
```

- Right Panel:**
 - Usage:** Help for any object by pressing Cmd+I.
 - Help:** Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Help.
 - New to Spyder?** Read our tutorial.
- Console:** Shows the Python environment and an In [1]: prompt.

Annexes:

The screenshot shows the PyCharm IDE interface with the following details:

- File Bar:** /Users/nouroudine/PycharmProjects/Projet TIPE 2024/TIPE
- Editor:** distance.py (active tab)
- Code:** Python script containing functions for calculating column means, covariance matrix, and correlation coefficient.
- Toolbars:** Standard PyCharm toolbars at the top.
- Right Panel:**
 - Usage:** Help for objects.
 - Console:** Shows Python version and license information.
 - Variable Explorer:** Available tabs: H..., Variable Exp..., Pl..., Fi..., Code Ana...
 - Output:** In [1]:
- Bottom Status Bar:** conda: base (Python 3.9.13) Completions: conda(base) LSP: Python Line 37, Col 20 UTF-8 CRLF RW Mem 91%

```
176
177 def moyenne_colonnes(matrice):
178     n = len(matrice)
179     m = len(matrice[0])
180     moyennes_colonne = []
181
182     for j in range(m):
183         colonne = [matrice[i][j] for i in range(n)]
184         moyenne_colonne = np.mean(colonne)
185         moyennes_colonne.append(moyenne_colonne)
186
187     return moyennes_colonne
188
189 def matrice_covariance(liste_segments, matrice_segments):
190     n = len(matrice_segments[0])
191     mat_cov = np.zeros((n, n))
192     seg_moy = moyenne_colonnes(matrice_segments)
193
194     for i in range(n):
195         for j in range(n):
196             for k in range(n):
197                 mat_cov[i][j] += (matrice_segments[k][i] - seg_moy[i]) * (matrice_segments[k][j] - seg_moy[j]) / (n - 1)
198
199     return mat_cov
200
201 matrice_covariance = matrice_covariance(liste_segments, matrice_segments)
202
203
204 # Visualiser la matrice de covariance
205 plt.figure(figsize=(10, 8))
206 sns.heatmap(matrice_covariance, annot=True, cmap='coolwarm', fmt='.2f', annot_kws={"size": 20}, linewidths=0.5)
207 plt.title('Matrice de Covariance des Segments')
208 plt.xlabel('Segments')
209 plt.ylabel('Segments')
210 plt.show()
211
212 #print(matrice_covariance(liste_segments, matrice_segments))
213
214 def coefficient_correlation(matrice_covariance):
215     n = len(matrice_covariance)
216     mat_crol = np.zeros((n, n))
217
218     for i in range(n):
219         for j in range(n):
220             mat_crol[i][j] = matrice_covariance[i][j]/np.sqrt(matrice_covariance[i][i]*matrice_covariance[j][j])
221
222     return mat_crol
223
```

Annexes:

The screenshot shows the PyCharm IDE interface with the following details:

- File Bar:** /Users/nouroudine/PycharmProjects/Projet TIPE 2024/TIPE
- Editor:** distance.py (active tab)
- Code Content:** Python script for calculating correlation and uncertainty. The code includes:
 - Defining a function `coefficient_correlation` to calculate the correlation matrix.
 - Calculating the correlation matrix from a covariance matrix.
 - Visualizing the correlation matrix using a heatmap.
 - Defining a function `calcul_incertitude` to calculate the total uncertainty based on segment uncertainties and covariances.
 - Printing the calculated uncertainty.
 - Defining a function `z_score` to calculate the z-score.
 - Printing the calculated z-score.
- Right Panel:**
 - Usage:** Help for objects can be accessed via Cmd+I.
 - Console:** Shows the Python environment and the output of the script execution.