```python
"""
Regroupement de différentes fonctions utiles aux deux programmes
"""
import random as rnd
import matplotlib.pyplot as plt
import numpy as np
import numpy.random as npr

def gen_points(nb_points):    #génère des points dans sur le plan
  points = []
  for i in range(nb_points):
    x = rnd.random() * 2 - 1
    y = rnd.random() * 2 - 1
    points.append((x, y))

  return points

def plot_points(points):    #affiche les points sur le plan
  x_list = []
  y_list = []

  for pt in points:
    x_list.append(pt[0])
    y_list.append(pt[1])
  plt.scatter(x_list, y_list)

def display_path(path, points):   #affiche le chemin sur le plan
    nb_points = len(points)

    x_list = []
    y_list = []

    for i in range(nb_points):
        pt = points[path[i]]
        x_list.append(pt[0])
        y_list.append(pt[1])
    pt = points[path[0]]
    x_list.append(pt[0])
    y_list.append(pt[1])

    plt.plot(x_list, y_list, 'o-')

def dist(pt1, pt2):   #calcule la distance entre deux points
  dx = pt2[0] - pt1[0]
  dy = pt2[1] - pt1[1]

  distance = np.sqrt(dx*dx + dy*dy)

  return distance

def create_distance_matrix(points):   #créé la matrice des distances entre tous les points
    nb_points = len(points)
    matrix = np.ones((nb_points, nb_points))

    for i in range(nb_points):
        for j in range(nb_points):
            matrix[i][j] = dist(points[i], points[j])

    return matrix
```

```python
import numpy as np
import matplotlib.pyplot as plt
from utils import *


''' Génère en utilisant la récursivité une liste de tous les chemins possibles, commençant par 0.
  Pour chaque point, on ajoute tous les chemins commençant par ce point -> appel à get_all_path sur l'ensemble des points privé du point choisi
  fp permet de fixer le premier point à 0
  Cette fonction calcule en même temps la longueur de chaque chemin
  -> renvoie la liste des couples (path_i, length_i), avec i variant de 1 à (n-1)! (car nombre de permutations sur l'ensemble [1, n-1]'''
def get_all_paths(inds, dists, points, fp = False):
  n = len(inds)
  if(n == 0): return [([], 0)]

  paths = []
  for p in [inds[0]] if fp else inds:
    n_point = list(inds)
    del n_point[n_point.index(p)]

    n_paths = get_all_paths(n_point, dists, points)
    for path in n_paths:
      if(path[0] == []): # Cas limite de la récursion
        d = dists[p, 0]
        paths.append([[p], d])
      else:
        d = dists[p, path[0][0]]
        paths.append([[p] + path[0], d + path[1]])
  return paths

''' Enfin, fonction qui sélectionne le chemin le plus court, par un simple appel à min() '''
def get_best_path(inds, dists, points):
        paths = get_all_paths(inds, dists, points, True)
        best_path = min(paths, key = lambda p: p[1]) # p[1] correction à l'élément length de (path, length)

        return best_path

points = [(-0.464, -0.48), (-0.106, -0.155), (-0.484, 0.2225), (0.12, -0.66), (0.308, 0.275), (-0.004, 0.5), (0.51, -0.2525), (-0.4, -0.1825), (0.03, -0.135)]
#points = gen_points(9)

dists = create_distance_matrix(points)

best_path = get_best_path(list(range(9)), dists, points)

display_path(best_path[0], points)
plot_points(points)

plt.title("Meilleur chemin - méthode exhaustive")

plt.show()

```

```python
import matplotlib.pyplot as plt
import numpy as np
import numpy.random as npr
from utils import *

''' Met à jour les phéromones : chemin plus court = plus de phéromones '''
def update_pheromones(path, path_length, pheromone_matrix):

  for i in range(0, len(pheromone_matrix)):
    pt1 = path[i-1]
    pt2 = path[i]

    pheromone_matrix[pt1][pt2] += 1/path_length

''' Pour un point donné, connaissant les points qui n\'ont pas encore été parcourus par la fourmi, calcule la probabilité pour chaque points'''
def gen_probas(last_point, pt_left, pheromone_matrix, distance_matrix, a, b):
  probas = []
  total = 0
  for p in pt_left:
    ro = 1 / distance_matrix[last_point][p]
    delta = pheromone_matrix[last_point][p]
    proba = ro ** a * delta ** b

    probas.append(proba)
    total += proba
  probas = [proba / total for proba in probas] # On divise par la somme des probabilités pour obtenir un SCE (somme des probabilités = 1)
  return probas

''' Point suivant choisi par la fourmi en fonction des probabilités de chaque point'''
def next_point(last_point, points_left, pheromone_matrix, distance_matrix, a, b):
  batch = gen_probas(last_point, points_left, pheromone_matrix, distance_matrix, a, b)
  return npr.choice(points_left, 1, p=batch)[0]

''' Code le comportement de la fourmi : choisit les points un par un au hasard en tenant compte de la distance et des phéromones, en gardant en mémoire les points restants
  De plus, la longueur du chemin est calculée au fur et à mesure de l'avancement de la fourmi, en utilisant des distances précalculées pour accélérer le programme
  a et b sont les paramètres de réglage modifiant l'importance de la distance et des phéromones dans le choix de la fourmi '''
def fourmi(points, pheromone_matrix, distance_matrix, a, b):
  nb_points = len(points)
  path = [0] # le chemin sera construit dans cette liste. exemple de chemin : [0, 6, 4, 2, 3, 5, 1] (remarque : les chemins sont fermés : le dernier point est relié au premier)
  points_left = list(points)
  path_length = 0
  for i in range(nb_points):
    n_p = next_point(path[-1], points_left, pheromone_matrix, distance_matrix, a, b)

    del points_left[points_left.index(n_p)]

    path_length += distance_matrix[path[-1]][n_p]

    path.append(n_p)
  return path, path_length
```

```
51
52  ''' Simule nb_gen générations de fourmis, avec a = 4 et b = 1, et renvoie le meilleur
    chemin.
53    Remplit en plus deux graphes : longueur du chemin par génération et longueur du
    meilleur chemin trouvé jusqu'alors à la n-ième génération'''
54  def generations(points, nb_gen, distance_matrix):
55    nb_points = len(points)
56    pheromone_matrix = np.ones((nb_points, nb_points))
57
58    best_path = []
59    best_path_length = 100000000
60
61    length_graph = []
62    best_length_graph = []
63
64    for i in range(nb_gen):
65
66      path, path_length = fourmi(list(range(1,nb_points)), pheromone_matrix,
    distance_matrix, 4, 1)
67
68      if path_length < best_path_length:
69        best_path_length = path_length
70        best_path        = path
71
72      length_graph.append(path_length)
73      best_length_graph.append(best_path_length)
74
75      update_pheromones(path,      path_length,      pheromone_matrix)
76      update_pheromones(best_path, best_path_length, pheromone_matrix)
77
78    return best_path, pheromone_matrix, distance_matrix, length_graph,
    best_length_graph
79
80  ''' Fonction principale, qui exécute generations() avec des paramètres de test, et
    affiche les graphes utiles '''
81  def main():
82    nb_points = 20
83    nb_gen = 1000
84    points = gen_points(nb_points)
85    distance_matrix = create_distance_matrix(points)
86
87    path, pheromone_matrix, _, length_graph, best_length_graph = generations(points,
    nb_gen, distance_matrix)
88
89    plt.subplot(1, 2, 1)
90
91    display_path(path, points)
92    plot_points(points)
93    plt.title("Meilleur chemin")
94
95    plt.subplot(1, 2, 2)
96
97    plt.plot(list(range(nb_gen)), length_graph, label='Longueur du chemin')
98    plt.plot(list(range(nb_gen)), best_length_graph, label='Meilleur longueur')
99
100   plt.legend()
101   plt.title("Evolution de la longueur du chemin au cours des générations")
102   plt.show()
103
104 main()
```