```python
# code concaténé.py

## txt_to_db_GTFS_IDF.py

import numpy as np
import codecs
import sqlite3 as sql
import matplotlib.pyplot as pl
import json
import math as m

stops=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\IDFM_gtfs\\stops.txt', 'r', encoding='utf-8')
trips=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\IDFM_gtfs\\trips.txt', 'r', encoding='utf-8')
stop_times=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\IDFM_gtfs\\stop_times.txt', 'r', encoding='utf-8')
routes=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\IDFM_gtfs\\routes.txt', 'r', encoding='utf-8')
transfers=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\IDFM_gtfs\\transfers.txt', 'r', encoding='utf-8')
agency=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\IDFM_gtfs\\agency.txt', 'r', encoding='utf-8')
calendar=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\IDFM_gtfs\\calendar.txt', 'r', encoding='utf-8')
calendar_dates=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\IDFM_gtfs\\calendar_dates.txt', 'r', encoding='utf-8')

conn=sql.connect(r"F:\informatique\TIPE\database\produit exploitable\GTFS.db")
c = conn.cursor()
erreur=[]


## Remplire de données exploitables la DB
def modele(nom_table_string, fonction, fichier, delete):
    num_lig=-1
    print(nom_table_string)
    if delete:
        c.execute('DELETE FROM {}'.format(nom_table_string))
        conn.commit()
    for line in fichier:
        if num_lig!=-1:  #la première ligne
            try:
                fonction(num_lig, line)
            except Exception:
                erreur.append([line,nom_table_string])
        num_lig+=1
        if (num_lig%10000)==0:
            print(num_lig)
    conn.commit()
    print(len(erreur), 'erreurs \n')

#il y a parfois des virgules dans les "stop_name"
def traitement_stops(num_lig,line):
    data=line.split('"')
    stop_name=traitement_accents_tirets(data[1])
    data2=[]
    fin_de_ligne=''
    for i in range(2, len(data)):
        fin_de_ligne+=data[i]
    for subline in (data[0], fin_de_ligne):
        data2.extend(subline.split(','))
    c.execute('insert into stops(id, stop_id, stop_name, stop_lat, stop_lon) values (?,?,?,?,?)', (num_lig, data2[0], stop_name, data2[4], data2[5]))

```

```
0055|
0056| def traitement_stop_times(num_lig,line):
0057|     data=line.split(',')
0058|     c.execute('insert into stop_times(trip_id, departure_time, stop_id, stop_sequence) values (?,?,?,?)', (data[0], data[1], data[3], int(data[4])))
0059|
0060| def traitement_trips(num_lig,line):
0061|     data=line.split(',')
0062|     c.execute('insert into trips(route_id, trip_id, trip_short_name, service_id) values (?,?,?,?)', (data[0], data[2], traitement_guillemets(data[3]), data[1]))
0063|
0064| def traitement_routes(num_lig,line):
0065|     data=line.split(',')
0066|     [name1, name2]=data[2:4]
0067|     c.execute('insert into routes(route_id, agency_id, route_short_name, route_long_name, route_type) values (?,?,?,?,?)', (data[0], data[1],
traitement_guillemets(name1), traitement_guillemets(name2), int(data[5])))
0068|
0069| def traitement_transfers(num_lig,line):
0070|     data=line.split(',')
0071|     c.execute('insert into transfers(from_stop_id, to_stop_id, transfer_time) values (?,?,?)', (data[0], data[1], data[3]))
0072|
0073| def traitement_agency(num_lig,line):
0074|     data=line.split(',')
0075|     c.execute('insert into agency (agency_id, agency_name) values (?,?)', (data[0], traitement_guillemets(data[1])))
0076|
0077| def traitement_calendar(num_lig,line):
0078|     data=line.split(',')
0079|     c.execute('insert into calendar (service_id, start_date, stop_date) values (?,?,?)', (data[0], data[8], data[9]))
0080|
0081| def traitement_calendar_dates(num_lig,line):
0082|     data=line.split(',')
0083|     c.execute('insert into calendar_dates(service_id, date, exception_type) values (?,?,?)', (data[0], data[1], data[2]))
0084|
0085| def traitement_guillemets(string):
0086|     n=len(string)
0087|     return string[1:n-1]
0088|
0089|
0090| #pour les frequentations il faut rejoindre les gares à leur id par leur nom
0091| def traitement_accents_tirets(string):
0092|     n=len(string)
0093|     suppression=0  #j'enlève les tirets et les espaces associés
0094|     string=string.upper() #majuscules
0095|     i=0
0096|     while i < n-suppression:
0097|         lettre=string[i]
0098|         if lettre=='-':
0099|             if string[i+1]==' ' and string[i-1]==' ':
0100|                 string=string[:i-1]+' '+string[i+2:]
0101|                 suppression+=2
0102|             else:
0103|                 string=string[:i]+' '+string[i+1:]
0104|         if lettre in 'ÀÄÂ':
0105|             string=string[:i]+'A'+string[i+1:]
0106|         elif lettre in 'ÉÈÊË':
0107|             string=string[:i]+'E'+string[i+1:]
0108|         elif lettre=='Ç':
0109|             string=string[:i]+'C'+string[i+1:]
```

```
        i+=1
    return string

## frequentation
# elles sont loin d'être exactes mais permettent des estimations
'''les fréquentations utilisées dans TIPE_traffic sont obtenues à partir du nombre de gares proches (importance donc du pôle de population) et d'un facteur multiplicatif
le facteur est calculé pour obtenir des fréquentations cohérentes avec celles du RER B qui sont obtenues ici'''

voyageurs_ratp=open('F:\\informatique\\TIPE\database\\format brut\\trafic-annuel-entrant-par-station-RATP.json','r').read()

# on est obligé de considérer que le nom de chaque gare d'idf est unique
def remplir_frequentation():
    c.execute('delete from flux_emis')
    conn.commit()
    print('delete done')
    liste=selection_reseau_RER_metro()
    ajout_voyageurs(liste)

def ajout_voyageurs(liste):
    data=json.loads(voyageurs_ratp)
    print(len(data), 'gares enregistrées')
    compt=0
    for line in data:
            nom, freq, route_type=ratp_voyageurs(line)
            reponse=comparaison(liste, nom)
            if reponse!=None:
                id_groupe,route_id=reponse
                c.execute('insert into flux_emis (id_groupe, route_id, frequentation_m, frequentation_c) values (?,?,?,?)',(id_groupe, route_id, 0, freq))
                compt+=1
            if compt%200==0:
                print(compt)
    print('done\n', compt, ' freq_c attribuées\n')
    conn.commit()

def ratp_voyageurs(line):
    nom=traitement_accents_tirets(line['fields']['station'])
    freq=line['fields']['trafic']//365
    res=line['fields']['reseau']
    if res=='Métro':
        route_type=1
    elif res=='RER':
        route_type=2
    else:
        route_type=None
    return nom, freq, route_type

#on ne sélectionne que les stations de RER/métro où ne passe qu'une seule ligne --> pas d'ambiguité sur l'attribution du flux émis
def selection_reseau_RER_metro():
    c.execute('''
    select id_groupe, stop_name, route_id, route_type
    from (
        select id_groupe, stop_name, route_id, route_type, count(route_id) as c
        from(
            select DISTINCT stops_groupe.id_groupe, stops_groupe.stop_name, graphe.route_id, graphe.route_type
                from graphe
```

```
                join stops_groupe
                    on stops_groupe.id_groupe=graphe.from_id_groupe
                where (route_type=1 or route_type=2)
                order by stop_name
            )
        group by id_groupe
        )
        where c=1''')
    liste=c.fetchall()
    print('liste réseau RER_metro récupérée')
    return liste

#le coût est un peu grand mais le nombre de gares réduit aux RER donc ok
def comparaison(liste, nom):
    for id_groupe, stop_name, route_id, route_type in liste:
        if nom in stop_name or stop_name in nom:
            liste.remove((id_groupe, stop_name, route_id, route_type))
            return id_groupe, route_id


##mobilités
code_postaux=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\laposte_hexasmal.csv','r')
mobilites=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\flux_mobilite_domicile_lieu_travail.csv','r')

#limites de l'idf (grossières):
latmin, latmax, lonmin, lonmax=47.9, 49.5, 1.1, 3.6
def traitement_postaux(num_lig, line):
    data=line.split(';')
    code_insee, name, code_postal = data[:3]
    coords=data[5]
    lat, lon = coords.split(',')
    lat, lon = float(lat), float(lon)
    if lat>latmin and lon>lonmin and lon<lonmax and lat<latmax and int(code_insee)!=89126:
        c.execute('insert into mobilites_pro(cle, code_commune, code_postal, name, lat, lon) values(?,?,?,?,?,?)', (num_lig, code_insee, code_postal, name, lat,
lon))

def elimination_doublons():
    c.execute('''
    delete from mobilites_pro
    where cle not in (
    select  cle
        from mobilites_pro
        group by code_commune)''')
    conn.commit()
    print('sans doublons\n')

def traitement_mobilites(num_lig, line):
    if num_lig>4:
        data=line.split(',')
        code_insee=data[0]
        flux_emis_jour=data[3]
        c.execute('update mobilites_pro set flux_emis_jour={f} where code_commune={c}'.format(f=flux_emis_jour, c=code_insee))



##fonctions à appeler
```

```
0220| def ecriture():
0221|     modele("stops", traitement_stops, stops, True)
0222|     modele("routes", traitement_routes, routes, True)
0223|     modele("trips", traitement_trips, trips, True)
0224|     modele("stop_times", traitement_stop_times, stop_times, True)
0225|     modele("transfers", traitement_transfers, transfers, True)
0226|     modele("agency", traitement_agency, agency, True)
0227|     modele("calendar", traitement_calendar, calendar, True)
0228|     modele("calendar_dates", traitement_calendar_dates, calendar_dates, True)
0229|     remplir_frequentation()
0230|     modele('mobilites_pro', traitement_postaux, code_postaux, True)
0231|     elimination_doublons()
0232|     modele('mobilites_pro', traitement_mobilites, mobilites, False)
0233|     conn.close()
0234|
0235|
0236|
0237|
0238|
0239|
0240|
0241|
0242|
0243|
0244|
0245|
0246| ##algorithmes_de_minimisation.py
0247| import numpy as np
0248|
0249|
0250|
0251| ##méthode du nombre d'or
0252| phi=(np.sqrt(5)+1)/2
0253| def minimiser(xmin, xmax, seuil_x, f):
0254|     print(xmin, xmax)
0255|     x2=xmin+(xmax-xmin)/(phi+1)
0256|     if abs(x2-xmin)<seuil_x or abs(x2-xmax)<seuil_x:
0257|         return x2
0258|     else:
0259|         y2=f(x2)
0260|         x3 = x2+(xmax-x2)/(phi+1)
0261|         y3=f(x3)
0262|         if y3>y2: #on réduit l'intervalle
0263|             return  minimiser(xmin, x2, seuil_x, f)
0264|         else:
0265|             return  minimiser(x3, xmax, seuil_x, f)
0266|
0267| ##algorithme de gradient descent
0268| '''variables contient les N paramètres de f
0269| infinitésimaux contient des variations de chacun des paramètres, suffisantes pour approximer la dérivée
0270| le test de convergence se fait sur la norme du gradient
0271| '''
0272|
0273|
0274| # approximation de la dérivée partielle par rapport à la variable d'index num_deriv à partir de la tangente
0275| def derive_partielle(f, variables, infinitesimaux, num_deriv):
```

```
0276|     X1=variables.copy(); X2=variables.copy()
0277|     X1[num_deriv]-=infinitesimaux[num_deriv]; X2[num_deriv]+=infinitesimaux[num_deriv]
0278|     f1=f(X1); f2=f(X2)
0279|     return (f2-f1)/(2*infinitesimaux[num_deriv])
0280|
0281| def gradient(f, variables, infinitesimaux, N):
0282|     grad=[]
0283|     for num_deriv in range(N):
0284|         grad.append(derive_partielle(f, variables, infinitesimaux, num_deriv))
0285|     return grad
0286|
0287| def norme(vect):
0288|     somme=0
0289|     for direc in vect:
0290|         somme+=direc**2
0291|     return np.sqrt(somme)
0292|
0293| def diff_vect(U, V):
0294|     W=[]
0295|     for i in range(len(U)):
0296|         W.append(U[i]-V[i])
0297|     return W
0298|
0299| def prod_vect(alpha, U):
0300|     W=[]
0301|     for u in U:
0302|         W.append(alpha*u)
0303|     return(W)
0304|
0305| #dans le changement du pas, si les variables changent peu sans que pour autant le grad soit petit, c'est que l'une des variable est limitée par les bornes du
modèle
0306| def minimum(f, variables0, bornes, infinitesimaux, pas_cv_grad, pas, dessin):
0307|     variables=variables0.copy()
0308|     N=len(variables)
0309|     iteration=1; grad=gradient(f, variables, infinitesimaux, N)
0310|     if dessin:
0311|         X, Y= [], []
0312|     while iteration<15 and norme(grad)>pas_cv_grad:
0313|         if dessin:
0314|             X.append(variables[0])
0315|             Y.append(variables[1])
0316|         print('itération ', iteration, variables, grad, pas, '\n')
0317|         avancement=prod_vect(pas, grad)
0318|         variables2=diff_vect(variables, avancement)
0319|         for num_dir in range(N):
0320|             borne_dir=bornes[num_dir]
0321|             variation_var=0
0322|             if variables2[num_dir]<borne_dir[1] and variables2[num_dir]>borne_dir[0]: #on ne change la valeur du paramètre que s'il reste dans les limites du
modèle
0323|                 variation_var+=abs(variables[num_dir]-variables2[num_dir])
0324|                 variables[num_dir]=variables2[num_dir]
0325|         #amélioration du taux s'apprentissage:
0326|         # if variation_var<pas:
0327|         #     pas=pas/10
0328|         iteration+=1
0329|         grad=gradient(f, variables, infinitesimaux, N)
```

```
0330|     if dessin:
0331|         return X, Y
0332|     else:
0333|         return variables
0334|
0335| ##exemple
0336| from mpl_toolkits.mplot3d import Axes3D
0337| import matplotlib.pyplot as pl
0338|
0339| def fonction_test(variables):
0340|     x, y = variables
0341|     return 10*(np.sin(x)**2 + 0.8*np.sin(y)**2)**2
0342|
0343| def fction_non_vect(x, y):
0344|     return 10*(np.sin(x)**2 + 0.8*np.sin(y)**2)**2
0345|
0346| def grid():
0347|     g=np.vectorize(fction_non_vect)
0348|     X=np.arange(-1.5, 0.9, 0.1)
0349|     Y=np.arange(-1.5, 0.7, 0.1)
0350|     X, Y = np.meshgrid(X, Y)
0351|     Z = g(X, Y)
0352|     return X, Y, Z
0353|
0354| def plot_3D():
0355|     X, Y, Z = grid()
0356|     fig=pl.figure()
0357|     ax=Axes3D(fig)
0358|     ax.plot_surface(X, Y, Z)
0359|     pas = 0.01
0360|     X, Y = minimum(fonction_test, [-1, -1.5], [[-10, 10], [-10, 10]], [0.0001, 0.0001], 0.001, pas, True)
0361|     n = len(X)
0362|     decal = -1
0363|     for k in range(n):
0364|         x, y = X[k], Y[k]
0365|         z = fction_non_vect(x, y)
0366|         ax.quiver(x, y, z+decal, 0, 0, -decal, color = 'black', linewidth=1, arrow_length_ratio=0)
0367|         ax.scatter(xs=x, ys=y, zs=z+decal, color='red')
0368|     pl.show()
0369|
0370| def plot_contour():
0371|     X, Y, Z = grid()
0372|     pl.grid()
0373|     pl.contour(X, Y, Z, range(30), colors =['blue']*30, alpha=0.7)
0374|     pas = 0.01
0375|     X, Y = minimum(fonction_test, [-1, -1.5], [[-10, 10], [-10, 10]], [0.0001, 0.0001], 0.001, pas, True)
0376|     n = len(X)
0377|     for k in range(n):
0378|         x, y = X[k], Y[k]
0379|         pl.scatter(x, y, color='red', s=20)
0380|         if k<n-1:
0381|             pl.annotate('', xy=(x, y), xytext=(X[k+1], Y[k+1]),  arrowprops=dict(arrowstyle="->", lw=3, mutation_scale=1))
0382|     pl.savefig('contour 2D gradient', dpi=300,  bbox_inches='tight')
0383|     pl.show()
0384|
0385| # minimum(fonction_test, [1, 1], [[-2, 2], [-2, 2]], [0.01, 0.01], 0.001, 0.3, True)
```

```
0386|
0387|
0388| ##monte Carlo
0389| #n est le nombre de valeur prise par variable d'entrée de f
0390| # si f est une fonction de 3 variables, alors il y aura n^^3 valeurs prises
0391| #on représente chaque point testé par une liste (écriture en base n)
0392| def monteCarlo(f, bornes, n):
0393|     N = len(bornes)
0394|     mini = np.inf; meilleur_var='init'
0395|     pas = [(bornes[k][1] - bornes[k][0])/n for k in range(N)]
0396|     pos = [0]*N #point de départ
0397|     for k in range(n**N):
0398|         if k%200==0:
0399|             print(pos)
0400|         var = [bornes[j][0]+pos[j]*pas[j] for j in range(N)]
0401|         resultat = f(var)
0402|         if resultat<mini:
0403|             print('minimum de ', resultat, ' obtenu en ', var, 'correspondant à', pos)
0404|             mini = resultat
0405|             meilleur_var = var
0406|         dizaine = True
0407|         j = 0
0408|         while dizaine and j<N:
0409|             if pos[j]==n-1:
0410|                 pos[j] = 0
0411|                 j+=1
0412|             else:
0413|                 pos[j]+=1
0414|                 dizaine = False
0415|     return mini, meilleur_varimport sqlite3 as sql
0416|
0417|
0418|
0419|
0420|
0421|
0422|
0423|
0424|
0425|
0426|
0427|
0428| ##Donnees_trafic.py
0429| import sqlite3 as sql
0430| import numpy as np
0431| import matplotlib.pyplot as pl
0432| from traitement_BDD import creation_liste_dest, correction_directs_ligne, facteur_population
0433|
0434|
0435| #le route_id existe-il
0436| def test_existence(c, route_id):
0437|     c.execute('select count(*) from routes where route_id="{}"'.format(route_id))
0438|     if c.fetchone()[0]==1:
0439|         return True
0440|     else:
0441|         return False
```

```
0442|
0443|
0444| ##données de database - creation de points, distances, gares,
0445| def creer_points_id(c, c2, route_id):
0446|     gares=[]
0447|     c.execute('''
0448|     select distinct from_id_groupe
0449|     from graphe
0450|     join stops_groupe
0451|         on stops_groupe.id_groupe=graphe.from_id_groupe
0452|     where route_id="{}"
0453|     order by from_id_groupe'''.format(route_id))
0454|     points=[]; liste_id=[]; gares=[]
0455|     compt=0
0456|     for from_id_groupe in c:
0457|         id_g=from_id_groupe[0]
0458|         liste_id.append(id_g)
0459|         c2.execute('select x,y from stops_groupe where id_groupe={}'.format(id_g))
0460|         x, y = c2.fetchone()
0461|         points.append([x,y])
0462|         gares.append([[], []]) #gare=[quais, frequentations]
0463|         compt+=1
0464|         print(id_g)
0465|     return liste_id, points, gares
0466|
0467| def creer_distances_graphe_gares(liste_id, c, route_id, gares):
0468|     c.execute('select from_id_groupe, to_id_groupe, distance from graphe where route_id="{}"'.format(route_id))
0469|     liste=c.fetchall()
0470|     n=len(liste)
0471|     print(n, 'liaisons')
0472|     N=len(liste_id)
0473|     print(N, 'gares')
0474|     distances=np.zeros((N,N))
0475|     compt=0
0476|     for from_id_groupe, to_id_groupe, distance in liste:
0477|         [numgare1, numgare2] = correspondance_DB_traffic([from_id_groupe, to_id_groupe], liste_id)
0478|         distances[numgare1, numgare2] = distance
0479|         nb_voies=1;
0480|         gares[numgare1][0].append([numgare2, 0, nb_voies])
0481|         compt+=1
0482|         if compt%10==0:
0483|             print(compt)
0484|     gares=correction_nb_voies(gares)
0485|     return distances, gares
0486|
0487| # rajoute des gares fictives qui permette de faire sortir du modèle les trains en fin de service
0488| def voies_garages(gares, dest, liste_id, c):
0489|     compt=0
0490|     for destination in dest:
0491|         num_terminus = destination[-2] #destination[-1]=-1 fin de service
0492|         if gares[num_terminus][0][-1]!=[-1, 0, 1]:
0493|             gares[num_terminus][0].append([-1, 0, 1]) #voie de garage
0494|     return gares
0495|
0496| # 2 voies par direction dans les gares ayant plus de deux directions (fourches)
0497| def correction_nb_voies(gares):
```

```
    for numgare in range(len(gares)):
        nb_dir=len(gares[numgare])
        if nb_dir>2: #plus de 2 dir
            for numdir in range(nb_dir):
                gares[numgare][numdir][2]=2
    return gares




## calcul de la distribution des flux à l'échelle d'une ligne - transit
'''flux_emis=flux_recu=frequentation_m*facteur_population
ceci est logique en considérant qu'il s'agit de flux domicile-travail qui sont donc symétrique
il s'agit de déterminer alpha, beta et K pour que la répartition soit correcte
En particulier il faut que la somme des flux_ij sur i soit Ei et la somme des flux_ij sur j soit Aj

'''

alpha=-0.0001
beta=-0.0003 #résultats meilleurs que la distance moyenne

def calcul_de_beta():
    conn=sql.connect(r"F:/informatique/TIPE/database/produit exploitable/GTFS.db")
    c=conn.cursor()
    c.execute('select avg(distance) from graphe')
    dist_moy=c.fetchone()[0]
    conn.close()
    return 1/dist_moy
#renvoie 0.003


def init_transit(route_id):
    conn=sql.connect(r"F:/informatique/TIPE/database/produit exploitable/GTFS.db")
    c=conn.cursor()
    c2=conn.cursor()
    liste_id, points, gares = creer_points_id(c, c2, route_id)
    return liste_id, points, c, c2, conn

def maj_transits(route_id='810:B'):
    liste_id, points, c, c2, conn = init_transit(route_id)
    distribution_transits(liste_id, points, route_id, c, c2, conn)
    conn.close


#le graphe ne renseigne que les gares directement reliées, il faut prendre en compte toutes les stations de la ligne
#la distance utilisé pour calculer l'impédance est à vol d'oiseau car il est inutile et compliqué de calculer en cumulé selon les stations reliées
def distribution_transits(liste_id, points, route_id, c, c2, conn):
    print('calcul transit... \n')
    c.execute('delete from flux_transit')
    n=len(liste_id)
    for i in range(n):
        id_grp1 = liste_id[i]
        xi, yi = points[i]
        c2.execute('select frequentation_m from flux_emis where id_groupe="{idg}" and route_id="{idr}"'.format(idg=id_grp1, idr=route_id))
        Vi=c2.fetchone()[0]
        Ei=Vi*facteur_population #flux émis par la gare i
```

```
0554|         flux_emis_j=0
0555|         for j in range(n):
0556|             if i!=j:
0557|                 id_grp2 = liste_id[j]
0558|                 xj, yj = points[j]
0559|                 dist = np.sqrt((yj-yi)**2+(xj-xi)**2)
0560|                 c2.execute('select frequentation_m from flux_emis where id_groupe="{idg}" and route_id="{idr}"'.format(idg=id_grp2, idr=route_id))
0561|                 Vj=c2.fetchone()[0]
0562|                 Aj=facteur_population*Vj
0563|                 impedance=dist**alpha*np.exp(beta*dist)
0564|                 flux_ij=Ei*Aj*impedance
0565|                 flux_emis_j+=flux_ij
0566|                 c.execute('insert into flux_transit(from_id_groupe, to_id_groupe, deplacements, route_id) values(?,?,?,?)', (id_grp1, id_grp2, int(flux_ij),
route_id))
0567|         K=Ei/flux_emis_j
0568|         c2.execute('update flux_transit set deplacements=deplacements*{K} where from_id_groupe={idg1}'.format(K=K, idg1=id_grp1))
0569|     conn.commit()
0570|     print('done \n')
0571|
0572| def comparaison_params():
0573|     global alpha, beta
0574|     for alpha in [-0.5, -0.7, -1]:
0575|         for beta in [-0.001, -0.02, -0.01]:
0576|             maj_transits(False)
0577|
0578|
0579|
0580| ##représentation des transits
0581| def repr_transit(maj, route_id='810:B', id_groupe=8824):
0582|     liste_id, points, c, c2, conn = init_transit(route_id)
0583|     if maj:
0584|         distribution_transits(liste_id, points, route_id, c, c2, conn)
0585|     representation_transits(points, liste_id, c, c2, id_groupe, route_id)
0586|     conn.close
0587|
0588| def representation_transits(points, liste_id, c, c2, id_groupe, route_id):
0589|     c2.execute('select stop_name from stops_groupe where id_groupe={}'.format(id_groupe))
0590|     nom_gare = c2.fetchone()[0]
0591|     fig_flux, [ax_pie, ax_graph] = pl.subplots(1, 2, figsize=[23, 16])
0592|     c.execute('select sum(deplacements) from flux_transit where from_id_groupe="{idg}" and route_id="{idr}"'.format(idg=id_groupe, idr=route_id))
0593|     total=c.fetchone()[0]  #on peut aussi récupérer frequentation_m dans flux_emisen multipliant par facteur_population
0594|     taux = total/50
0595|     c.execute('''
0596|     select deplacements, stop_name
0597|     from flux_transit
0598|     join stops_groupe
0599|         on stops_groupe.id_groupe=flux_transit.to_id_groupe
0600|     where flux_transit.from_id_groupe={}
0601|     '''.format(id_groupe))
0602|     labels=[]; sizes=[]
0603|     for dep, stop_name in c:
0604|         if dep>taux:
0605|             labels.append(stop_name)
0606|             sizes.append(dep)
0607|             print(dep)
0608|     patches, text, other = ax_pie.pie(sizes, autopct = lambda x: str(round(x, 2)) + '%')
```

```
    title='representation transit au départ de ' + str(nom_gare) + ' (' + str(int(total))+' voyageurs par jour)'
    ax_pie.set_title(title)
    ax_pie.legend(patches, labels, loc='lower right')
    ax_graph.set_title("évolution de l'impédance en fonction de la distance")
    X=np.arange(0.1, 10, 0.001)
    Y=[]
    for x in X:
        dist=x*1000
        impedance=dist**alpha*np.exp(beta*dist)
        Y.append(impedance)
    ax_graph.plot(X, Y)
    ax_graph.set_xlabel('distance en km')
    pl.savefig(title, dpi=300,  bbox_inches='tight')
    pl.show()


##remplissage et fréquentations
def remp_freq(dest, gares, liste_id, c):
    c.execute('select from_id_groupe, to_id_groupe, deplacements from flux_transit')
    for id_grp1, id_grp2, deplacements in c:
        print(id_grp1, id_grp2)
        [numgare1, numgare2] = correspondance_DB_traffic([id_grp1, id_grp2], liste_id)
        condition=False
        for destinat in dest:
            if numgare1 in destinat and numgare2 in destinat: #il existe un train qui emmenera les voyageurs de a à b
                condition=True
        if condition and deplacements>10: #on évite de surcharger avec des flux minimes
            gares[numgare1][1].append([numgare2, 0, deplacements, 0])



##outils
# transforme un id_groupe en son équivalent pour le traffic et réciproquement
def correspondance_traffic_DB(liste_id_traffic, liste_id):
    conn=sql.connect(r"F:/informatique/TIPE/database/produit exploitable/GTFS.db")
    c=conn.cursor()
    rep=[]
    for indice in liste_id_traffic:
        id_groupe=liste_id[indice]
        c.execute('select stop_name from stops_groupe where id_groupe={}'.format(id_groupe))
        rep.append([id_groupe, c.fetchone()[0]])
    conn.close()
    return rep

def correspondance_DB_traffic(liste_id_groupe, liste_id):
    rep=[]
    for id_groupe in liste_id_groupe:
        for indice in range(len(liste_id)):
            if id_groupe==liste_id[indice]:
                rep.append(indice)
    return rep

def verif_dessin(dest, points):
    ax=pl.axes(xlim=(630000,650000),ylim=(670000,680000))
    ax.axis('equal')
    for i in range(len(dest)):
```

```
0665|         ligne=dest[i]
0666|         for gare in ligne:
0667|             if i==0:
0668|                 col='blue'
0669|             else:
0670|                 col='red'
0671|             ax.plot(points[gare][0],points[gare][1],marker='+',color=col)
0672|     pl.show()
0673|
0674| def somme(liste):
0675|     somme=0
0676|     for elmt in liste:
0677|         somme+=elmt
0678|     return somme
0679|
0680| def relatif(liste):
0681|     total=somme(liste)
0682|     new_liste=[]
0683|     for elmt in liste:
0684|         new_liste.append(elmt/total)
0685|     return new_liste
0686|
0687| ##feuille de route et horaires
0688| '''' on créera d'abord un type/horaire par train existant puis on les rassemblera'''
0689|
0690| '''on ne prend que
0691| -les trains non exceptionnels (plus de 5 qui ont le même itinéraire)
0692| -circulant à la date donnée
0693| -pour le route_id'''
0694|
0695|
0696| def selection_trips_ids(route_id, c):
0697|     c.execute('''
0698|     select trips.trip_id
0699|     from stop_times
0700|     join trips
0701|         on trips.trip_id=stop_times.trip_id
0702|     join calendar
0703|         on calendar.service_id=trips.service_id
0704|             where trips.route_id="{}"
0705|             and stop_times.stop_sequence=1
0706|             and calendar.start_date<20200522 and calendar.stop_date>20200522
0707|     group by stop_times.stop_id, stop_times.departure_time
0708|     order by stop_times.stop_id, stop_times.departure_time
0709|         '''.format(route_id))
0710|     tup=()
0711|     for trip_id in c:
0712|         tup+=trip_id
0713|     print("trip_id selectionnées")
0714|     return tup
0715|
0716| #☻pour remplir les colonnes inutiles (route_type et route_id) on selectionnera des chiffres
0717| def creation_dest(c, selection_trips):
0718|     c.execute('''
0719|     select 1, 2, groupe.id_groupe, stop_times.stop_sequence
0720|     from stop_times
```

```
    join groupe
        on stop_times.stop_id=groupe.stop_id
    where stop_times.trip_id in {s}
    order by stop_times.trip_id desc, stop_times.stop_sequence asc

    '''.format(s=selection_trips))
    liste_dest=creation_liste_dest(c)
    dest=liste_dest[0][2]
    print('dest full ok\n')
    return dest

#on utilise également creation_liste_dest pour regrouper les horaires de passage d'un même trip
#l'horaire de départ sera utilisé dans horaires tandis que tous seront utilisés dans stop_times0 pour pouvoir calculer ponctualite()
def creation_horaires(c, selection_trips):
    c.execute('''
    select 1, stop_times.trip_id, stop_times.departure_time, stop_times.stop_sequence
    from stop_times
    where stop_times.trip_id in {s}
    order by stop_times.trip_id desc, stop_times.stop_sequence asc
    '''.format(s=selection_trips))
    liste_horaires=creation_liste_dest(c)
    horaires_full=[]
    stop_times0=[]
    for trip in liste_horaires:
        trip_id=trip[1]
        dep_times=trip[2][0]
        dep_times_corr=[]
        for h_m in dep_times:
            h_m=h_m.split(':')
            dep_times_corr.append(float(h_m[0])+float(h_m[1])/60)
        horaires_full.append([dep_times_corr[0], trip_id])
        stop_times0.append([trip_id, dep_times_corr])
    print('horaires full ok\n')
    return horaires_full, stop_times0

#on regroupe les dest par trip et on les joint à leurs horaires par ind qui deviendra le no_feuille_route
def reunion(dest_full, horaires_full):
    dest=[]; horaires=[]
    for indice_full in range(len(dest_full)):  # len(horaires_full)=len(dest_full)
        n=len(dest)
        indice=0
        while indice<n and dest_full[indice_full]!=dest[indice]:
            indice+=1
        if indice==n: #nouvelle feuille de route
            dest.append(dest_full[indice_full])
            horaires.append([len(dest)-1] + horaires_full[indice_full])
        else:
            horaires.append([indice] + horaires_full[indice_full])
    return dest, horaires


def creation_direct(dest):
    chgmt=1
    direct=[[] for k in range(len(dest))]
    while chgmt!=0:
        dest, chgmt, direct = correction_directs_ligne(dest, True, direct)
```

```
0777|     return dest, direct
0778|
0779| #on remplace tous les id_groupe par des numgare allant de 1 à n
0780| def convertir(dest, direct, liste_id):
0781|     n=len(dest)
0782|     dest_traffic=[]; direct_traffic=[]
0783|     for indice in range(n):
0784|         new_destin=correspondance_DB_traffic(dest[indice], liste_id)
0785|         new_destin.append(-1)
0786|         dest_traffic.append(new_destin)
0787|         direct_traffic.append(correspondance_DB_traffic(direct[indice], liste_id))
0788|     return dest_traffic, direct_traffic
0789|
0790| ##tri horaires
0791| #mise en place d'un quicksort qui agit in place en utilisant la première valeur comme pivot
0792| def partition(horaires, p, r):
0793|     piv = horaires[p][1]
0794|     j=p # horaires[j] est le dernier élément lu qui est plus petit que piv
0795|     for i in range(p+1, r):
0796|         if horaires[i][1]<=piv:
0797|             j+=1
0798|             echange(horaires, i, j)
0799|     echange(horaires, p, j)
0800|     return j
0801|
0802| def tri_rapide_rec(horaires, p, r):
0803|     if r>p+1:
0804|         q = partition(horaires, p, r)
0805|         tri_rapide_rec(horaires, p, q)
0806|         tri_rapide_rec(horaires, q+1, r)
0807|
0808| def quicksort(horaires):
0809|     n = len(horaires)
0810|     tri_rapide_rec(horaires, 0, n)
0811|
0812| def echange(liste, i, j):
0813|     if i!=j:
0814|         liste[i], liste[j] = liste[j], liste[i]
0815|
0816|
0817| ##types
0818| # [position,vitesse_actuelle, tps_attendu_arret, gar_départ, gar_arrivée, remplissage en personnes, no_feuille de route]
0819| def creation_types(dest):
0820|     modele_train=[0, 0, 0, 0, -1, 'terminus à remplir', [], 'no feuille']
0821|     n=len(dest)
0822|     types=[]; direct=[]
0823|     for no_feuille in range(n):
0824|         term_depart=dest[no_feuille][0]
0825|         dest[no_feuille].remove(term_depart)
0826|         new_type=modele_train.copy()
0827|         new_type[5]=term_depart; new_type[7]=no_feuille
0828|         types.append(new_type)
0829|     return types, dest
0830|
0831| ##affichage
0832| def cadre_num_max_trains(horaires, c, route_id):
```

```
0833|     num_max_trains=int(len(horaires)/10) #approximation du  nombre simultanné de trains
0834|     c.execute('''
0835|     select min(x), min(y), max(x), max(y)
0836|     from stops_groupe
0837|     join graphe
0838|         on graphe.from_id_groupe=stops_groupe.id_groupe
0839|     where graphe.route_id="{}"'''.format(route_id))
0840|     (minx, miny, maxx, maxy) = c.fetchone()
0841|     cadre=[[minx-2000, maxx+2000], [miny-20000, maxy+20000]]
0842|     return num_max_trains, cadre
0843|
0844| ##étalement horaire
0845| def etalement(horaires):
0846|     plages=[0]*24
0847|     N=len(horaires)
0848|     for hor in horaires:
0849|         t_dep=hor[1]
0850|         num_p=int(t_dep)
0851|         plages[num_p]+=1
0852|     for k in range(24):
0853|         plages[k]/=N
0854|     return plages
0855|
0856| ##variables
0857| def variables(route_id):
0858|     global liste_id
0859|     conn=sql.connect(r"F:/informatique/TIPE/database/produit exploitable/GTFS.db")
0860|     c=conn.cursor()
0861|     c2=conn.cursor()
0862|     liste_id, points, gares = creer_points_id(c, c2, route_id)
0863|     distances, gares = creer_distances_graphe_gares(liste_id, c, route_id, gares)
0864|     #on récupère toutes les informations de la ligne dans stop_times, on a : len(dest_full)=len(horaires_full)
0865|     selection_trips=selection_trips_ids(route_id, c)
0866|     dest_full=creation_dest(c, selection_trips)
0867|     horaires_full, stop_times0=creation_horaires(c, selection_trips)
0868|     #on les réduit
0869|     dest, horaires = reunion(dest_full, horaires_full)
0870|     #on trie les horaires
0871|     quicksort(horaires)
0872|     #on calcule la répartition des flux
0873|     plages = etalement(horaires)
0874|     #on modifie dest pour corriger les directs
0875|     dest, direct = creation_direct(dest)
0876|     #on transforme avec les id du traffic
0877|     dest, direct = convertir(dest, direct, liste_id)
0878|     #on crée les types et voies de garage
0879|     voies_garages(gares, dest, liste_id, c)
0880|     types, dest = creation_types(dest)
0881|     #affichage
0882|     num_max_trains, cadre = cadre_num_max_trains(horaires, c, route_id)
0883|     #on rajoute dans gares les freq
0884|     remp_freq(dest, gares, liste_id, c)
0885|     conn.close()
0886|     return points, liste_id, distances, dest, types, horaires, gares, direct, num_max_trains, cadre, stop_times0, dest_full, plages
0887|
0888|
```

```
0889|
0890|
0891|
0892|
0893|
0894|
0895| ##TIPE trafic.py
0896| import numpy as np
0897| import matplotlib.pyplot as pl
0898| import random as rd
0899| import matplotlib.animation as animation
0900| import sqlite3 as sql
0901| from Donnees_traffic import variables, correspondance_DB_traffic, correspondance_traffic_DB
0902| from matplotlib.widgets import Button
0903| from algorithmes_de_minimisation import minimum, monteCarlo
0904| from time import time
0905| import copy
0906|
0907|
0908| route_id="810:B"
0909| ## Creation et initialisation du maillage
0910| """
0911| dt en minutes
0912|
0913| points=[[x1, y1], [x2, y2], ... ]
0914| distances= matrice n*n diagonale ij = distances entre i et j
0915| si 0 alors non reliées
0916|
0917| gares=[[gare1], [gare2], ... ]
0918| gare=[occupations_des_voies, fréquentations]
0919| occupations_des_voies=[[direction(vers où), nombre de voies occupées, nombre de voies total], [dir2, n2, nprim2]...]
0920| fréquentations=[direction, passagers_en_gare,  passagers_quotidiens, dernière heure de desserte]
0921|
0922| dest=[[feuille de route1], [feuille2], ...]
0923| feuille de route=[gares par lesquelles passer]
0924| direct= meme format que dest -> gares ou le train est sans arret
0925|
0926| types=[[typ1], [typ2], ...]
0927|
0928| trains contient des typ
0929| typ=[position,vitesse_actuelle, acceleration, tps_attendu_arret, gar_départ, gar_arrivée, remplissage, no_feuille_route = no_feuille_direct]
0930|     si gar_depart=-1 alors le train est arrivé en gar_arrivé
0931|     la pos est en proportion de la distance, la vit est absolue
0932|     il y en a un par feuille de route
0933|
0934| remplissage=[[gare de destination, nombre de personnes], [gare2, n2], ...]
0935|
0936|
0937| destinations contient des dest
0938| dest=[gare1, gare2...]
0939|
0940| sans_arret contient des direct
0941| direct=[gare_évitée1, gare_évitée2 ...]
0942|
0943| Les deux fonctionnent en paire
0944|
```

```
arret contient des numtrains dont l'arret est contraint par un autre train
arret_force contient des numtrains dont l'arret est contraint par une perturbation (dans le but d'en évaluer l'impact)

fin_de_service contient des  [numtrain,numgare] allant être supprimés en numgare

horaires=[[type de train, horaire de départ, (trip_id1)], [typ2, horaire2, (trip_id2)], ...]

Attention, stop_times et stop_times0 ne son pas au même format

stop_times=[[numtrain, 'trip_id', passages(liste)], [numtrain2, 'trip_id2', passages(liste)], ... ]
passages=[numgare, heure, voy_montants, minutes_attendues]
stop_times_stock est similaire à stop_times mais avec des id_groupe à la place des numgare

stop_times0=[trip1, trip2 ...]
trip=[trip_id, dep_times]
dep_times=[h1, h2 ...]

profil_suivi=[intergare1, intergare2, ...]
intergare=[gare_dep, gare_ar, valeurs1, valeurs2, ...]
valeur=[pos, heure, vit, accel, remplissage] avec remplissage correspondant au contenu du train numtrain_course

"""
def variables_neutres():
    global destinations, arret, ralentissement, trains, fin_de_service, sans_arret, timer, pause, service, erreurs, stop_times, stop_times_stock, arret_force,
affluence, annote, profil, profil_suivi, condition_arret_prgm, numtrain_course, pert_on, dessin, affichage, consigne_pert, exemple
    destinations=[]; arret=[]; ralentissement=[]; trains=[]; fin_de_service=[]; sans_arret=[]; timer=False; pause=False; service=True; erreurs=[]; stop_times=[];
stop_times_stock=[];  arret_force=[]; annote=False; affluence=False; profil=False; profil_suivi=[]; condition_arret_prgm=False; numtrain_course=-1; pert_on=False;
dessin=False; affichage=True; consigne_pert=False; exemple=False

#il faut copier le contenu de chaque sous_liste de gares
#pas besoin pour horaires qui n'est pas modifié (seulement suppr)
def etat_initial():
    variables_neutres()
    global horaires, gares, zeros
    gares = copy.deepcopy(gares0)
    horaires = horaires0.copy()
    zeros=[0]*len(gares)

## donnees GTFS
#on peut choisir de lancer le modèle avec des paramètres différents
def donnees_GTFS(route_id = route_id):
    global dimension, annote
    dimension = 600
    annote = False
    global points, liste_id, distances, dest, types, horaires0, gares0, direct, num_max_trains, cadre, stop_times0, dest_full, plages
    # try:
    points, liste_id, distances, dest, types, horaires0, gares0, direct, num_max_trains, cadre, stop_times0, dest_full, plages = variables(route_id)
    # except Exception:
        # print('erreur de donnée')


##global
# les données sont sauvegardees au débur pour ne pas les recalculer
def demarrage():
    etat_initial()
```

```
0998|     global affichage, dessin, service
0999|     affichage=True; dessin=True; service=True
1000|     evolution_traffic(6, 0.1, 8, 20)
1001| '''
1002| num_h=4  départ à 6.0   trip_id_course='115072256-1_14393'; duree=0.2; tm=6.2; g1=34; g2=37
1003| num_h=51  départ à 7.38  trip_id_course='115054849-1_16156'; duree=0.2; tm=7.5; g1=30; g2=29
1004| '''
1005| def profil_course():
1006|     etat_initial()
1007|     global profil, trip_id_course, dessin, service, exemple
1008|     profil = True; dessin=True; service=False; exemple=False
1009|     trip_id_course='115054849-1_16156'
1010|     evolution_traffic(5, 0.1, 23, 200)
1011|     comparaison(trip_id_course)
1012|
1013|
1014| def evaluation_pert():
1015|     etat_initial()
1016|     global pert_on, dessin, service, profil, trip_id_course, consigne_pert
1017|     pert_on=True; dessin=False; service=False; profil=True; consigne_pert=False
1018|     trip_id_course='115054849-1_16156'; duree=0.2; tm=7.5; g1=30; g2=29
1019|     variables_de_pert(duree, tm, g1, g2)
1020|     evolution_traffic(5.0, 0.1, 23, 20)
1021|     comparaison(trip_id_course)
1022|     # reecriture_DB()
1023|     # ecart_db(True)
1024|
1025|
1026| #fait une itération du modèle pour les paramètres données, il s'agit de la fonction à minimiser
1027| #on ne calcule qu'une seule course
1028| def retour_ecart(variabs):
1029|     etat_initial()
1030|     global vitesse_nominale, vitesse_arrivee, contenance, tps_arret, tps_population
1031|     [vitesse_nominale, vitesse_arrivee, contenance, tps_arret, tps_population]=variabs
1032|     global affichage, horaires, service, dessin
1033|     affichage=False; service=False; dessin=False
1034|     horaires=[horaires0[51]] #train quelconque '115054849-1_16156'
1035|     evolution_traffic(7.2, 0.1, 9, 100)
1036|     ec = ponctualite()
1037|     return ec
1038|
1039| def evolution_traffic(t0, dt1, t_fin, time_par_frame):  #dt en minutes, t_fin et t_depart en heures
1040|     global t_depart, dt
1041|     delta_t = t_fin-t0
1042|     t_depart = t0
1043|     dt = dt1
1044|     n = int(delta_t//(dt/60))
1045|     if affichage:
1046|         print(n, 'frames')
1047|     initialisation_horaires(t0)
1048|     if dessin:
1049|         creation_artists(num_max_trains, cadre[0], cadre[1])
1050|         global ani
1051|         ani = animation.FuncAnimation(fig1, anim, fargs=(5, ), init_func = init, frames = n, blit = False, interval = time_par_frame, repeat = False)
1052|         pl.show()
1053|     else:
```

```
        for i in range(n):
            if condition_arret_prgm:
                break
            heure = t_depart+i*dt/60
            temps_suivant(heure)
            if i%50==0 and affichage:
                print('temps:', conversion_h_m(heure))
    stop_times_stock.extend(stop_times)
    statistiques_attente()

##Avancement d'une durée dt
def temps_suivant(heure):
    nb_trains = len(trains)
    mise_en_service(heure)
    for numtrain in range(nb_trains):
        if est_en_gare(numtrain):
            train_gare(numtrain, heure)
        else:
            train_voie(numtrain, heure)
    execute_fin_service()
    gestion_arret_force(heure)
    if profil:
        gestion_profil(heure)
    if pert_on:
        mise_en_place_pert(heure)


##gares
def est_en_gare(numtrain):
    if trains[numtrain][4]==-1:
        return True
    else:
        return False

def train_gare(numtrain, heure):
    gare1 = trains[numtrain][5]
    if len(destinations[numtrain])<=1:  #fin de service à la fin (modif des numtrains)
        fin_de_service.append([numtrain, gare1])
    else:
        gare2 = destinations[numtrain][0]
        ind = recherche_indice_quai(gare1,gare2)
        if ind is None:
            print('suppression du train', numtrain)
            erreurs.append([numtrain, 'destinations incohérentes de', gare1, 'à', gare2])
            fin_de_service.append([numtrain, gare1])
        else:
            personnes = voyageurs_quai(gare1, numtrain)
            if condition_redemarrage_gare(numtrain, personnes, gare1, gare2):
                execute_train_redemarrage(numtrain, gare1, gare2, personnes, ind, heure)
            else:
                execute_train_attente(numtrain)

#condition de tps d'attente: 30 sec +1 min toute les 1000 pers en gare
#condition voie libre: pas de trains dans la premiere moitié
def condition_redemarrage_gare(numtrain, personnes, gare1, gare2):
    pos_liaison = distance_train_proche(gare1, gare2, numtrain)
```

```
    if len(pos_liaison)>0:
        pos_proche = min(pos_liaison)
    else:
        pos_proche = 1 #pas de train
    if pos_proche>0.6 and trains[numtrain][3]>tps_arret+personnes*tps_population and (numtrain not in arret_force):
        return True
    else:
        return False


#on récupère les positions de tous les trains sur la liaison excepté le numtrain considéré (utile pour un train en voie)
def distance_train_proche(gare1, gare2, numtrain1):
    pos_proche=[]
    for numtrain in range(len(trains)):
        if trains[numtrain][4]==gare1 and trains[numtrain][5]==gare2 and numtrain!=numtrain1:
            pos = trains[numtrain][0]
            pos_proche.append(pos)
    return pos_proche


def recherche_indice_quai(gare1,gare2):
    occup=gares[gare1][0]
    for ind in range(len(occup)):
        if occup[ind][0]==gare2:
            return ind
    print('erreur, ce train souhaite aller de', gare1, 'à', gare2)

#on ne fait monter et descendre les voyageurs qu'au redémarrage car ces derniers interviennent dans les calculs de temps d'attente en gare
def execute_train_redemarrage(numtrain, gare1, gare2, personnes, ind, heure):
    trains[numtrain][4:6]=[gare1,gare2]
    trains[numtrain][1]=0     #sans vitesse initiale
    trains[numtrain][3]=heure                          #reset temps d'attente
    gares[gare1][0][ind][1]-=1  #libère un quai
    monter_voyageurs(numtrain,gare1, heure)

def execute_train_attente(numtrain):
    trains[numtrain][3]+=dt

##voies
def train_voie(numtrain, heure):
    [pos,vit] = trains[numtrain][0:2]
    gare1, gare2 = int(trains[numtrain][4]), int(trains[numtrain][5])
    dist = distances[gare1, gare2]
    gare3 = destinations[numtrain][1]
    ind = recherche_indice_quai(gare2,gare3)
    if ind is None or gare1==gare2:
        print('suppr')
        erreurs.append([numtrain, 'destinations incohérentes de', gare2, 'à', gare3])
    else:
        test_securite(numtrain, pos, gare1, gare2, gare3, ind, dist)
        #traite tous les cas d'arret, de ralentissement et d'arret forcé par la pert
        vit = trains[numtrain][1]
        accel = trains[numtrain][2]
        if (numtrain not in arret) and (numtrain not in ralentissement) and (numtrain not in arret_force):
            if arrive_en_gare(pos, vit, dist):
                if gare2 in sans_arret[numtrain]:
```

```
                    execute_train_direct_gare(numtrain,gare2,gare3)
                else:
                    execute_train_arrive_gare(numtrain, gare2, ind, heure, gare3)
            else:
                execute_train_avancer(numtrain, pos, vit, accel, dist)


def execute_train_arrive_gare(numtrain, gare2, ind, heure, gare3):
    augmenter_voyageurs_gares(gare2, numtrain, heure)
    descendre_voyageurs(numtrain,gare2)
    trains[numtrain][0:3]=[0,0,0]    #arret
    trains[numtrain][3]=0  #tps attente
    trains[numtrain][4]=-1
    trains[numtrain][5]=gare2
    remplissage = trains[numtrain][6]
    gare3 = int(destinations[numtrain][1])
    gares[gare2][0][ind][1]+=1 #occupe un quai
    destinations[numtrain].remove(gare2)
    suivi_ajout_passage(numtrain, gare2, heure)

def execute_train_direct_gare(numtrain,gare2,gare3):
    trains[numtrain][4:6]=[gare2,gare3]
    trains[numtrain][0]=0
    destinations[numtrain].remove(gare2)
    sans_arret[numtrain].remove(gare2) #pas utile mais plus clair

#la régulation de la dynamique du train se fait sur le couple qui est proportionnel à l'accélértion
#couple de démarrage constant-> montée linéaire de couple jusqu'à vitesse_nominale ou couple_nominal -> couple nul-> couple négatif de freinage
def execute_train_avancer(numtrain, pos, vit, accel, dist):
    distance=pos*dist
    if distance<distance_demarrage:
        accel1=accel_nominale
        vit1=vit+accel1*dt
        if vit1>vitesse_nominale:
            vit1=vitesse_nominale
    elif (dist-distance)<distance_freinage: #proche de l'arrivée
        vit1=vit+freinage_nominal*dt
        if vit1>vitesse_arrivee: #on ne veut pas qu'il s'arrete mais seulement qu'il ralentisse
            accel1=freinage_nominal
        else:
            vit1=vitesse_arrivee
            accel1=0
    else: #milieu de course
        accel1=0
        vit1=vitesse_nominale
    avance=vit1*dt
    pos1=pos+avance/dist
    trains[numtrain][0:3]=[pos1, vit1, accel1]

##sécurité
#deux trains proches
def test_securite_train(numtrain, pos, gare1, gare2, dist):
    liste_pos = distance_train_proche(gare1, gare2, numtrain)
    arret_demande = False; ralentissement_demande = False
    for pos1 in liste_pos:
        if abs((pos-pos1)*dist)<distance_securite and pos<pos1:
```

```
            ralentissement_demande=True
        if abs((pos-pos1)*dist)<distance_securite_mini and pos<pos1:
            arret_demande = True
    return ralentissement_demande, arret_demande

#voie non dispo
def test_securite_gare(numtrain, pos, gare2, gare3, ind, dist):
    arret_demande = False; ralentissement_demande = False
    if not voie_disponible_gare(gare2,gare3, ind):
        if (1-pos)*dist<distance_securite:
            ralentissement_demande = True
        if (1-pos)*dist<distance_securite_mini:
            arret_demande = True
    return ralentissement_demande, arret_demande

#on compile les raisons de s'arreter
def test_securite(numtrain, pos, gare1, gare2, gare3, ind, dist):
    r1, a1 = test_securite_train(numtrain, pos, gare1, gare2, dist)
    r2, a2 = test_securite_gare(numtrain, pos, gare2, gare3, ind, dist)
    arret_du_train((a1 or a2), numtrain)
    ralentissement_du_train((r1 or r2), numtrain, dist)

def arret_du_train(condition, numtrain):
    #arret force correspond à une perturbation qui impose un arret qui n'est pas lié aux conditions de sécurité énoncées ci-dessus
    #cet arret prime sur les autres
    if arret_force_de(numtrain):
        execute_train_attente(numtrain) #le train est déjà arreté
    else:
        if condition:
            if numtrain in arret:
                execute_train_attente(numtrain)
            else:
                execute_arret(numtrain)
        else:
            if numtrain in arret:
                arret.remove(numtrain)

def ralentissement_du_train(condition, numtrain, dist):
    if condition:
        if numtrain not in ralentissement:
            execute_ralentissement(numtrain, dist)
    else:
        if numtrain in ralentissement:
            ralentissement.remove(numtrain)

def arrive_en_gare(pos, vit, dist):
    if pos+vit*dt/dist>1:
        return True
    else:
        return False

def voie_disponible_gare(gare2,gare3, ind):
    if gares[gare2][0][ind][1]<gares[gare2][0][ind][2]:
        return True
    else:
        return False
```

```
1278|
1279| def execute_arret(numtrain):
1280|     arret.append(numtrain)
1281|     trains[numtrain][1:3]=[0,0]   #arret
1282|     print('train ', numtrain, 'arreté')
1283|
1284| def execute_ralentissement(numtrain, dist):
1285|     ralentissement.append(numtrain)
1286|     pos, vit = trains[numtrain][0:2]
1287|     trains[numtrain][2]=freinage_nominal
1288|     if vit>vitesse_arrivee:
1289|         vit1=vit+freinage_nominal*dt
1290|         print('train', numtrain, 'ralenti')
1291|     else:
1292|         vit1=vitesse_arrivee
1293|     pos1=pos+vit1*dt/dist
1294|     trains[numtrain][0:2]=[pos1, vit1]
1295|
1296|
1297| ##mise en service
1298|
1299| #permet de commencer à n'importe quelle heure en élimninant les trains passés
1300| def initialisation_horaires(heure):
1301|     global horaires
1302|     num_horaire=0
1303|     temps_deb=horaires[0][1]
1304|     while temps_deb<=heure and len(horaires)>1:
1305|         num_horaire+=1
1306|         temps_deb=horaires[num_horaire][1]
1307|     horaires=horaires[num_horaire:]
1308|     if affichage:
1309|         print(num_horaire, "horaires en dehors de la zone")
1310|     for gare in gares:
1311|         for direction in gare[1]:
1312|             direction[3]=heure
1313|
1314| def mise_en_service(heure):
1315|     if len(horaires)>0:
1316|         num_horaire=0
1317|         temps_deb=horaires[num_horaire][1]
1318|         while temps_deb<=heure and len(horaires)>num_horaire+1:
1319|             temps_deb=horaires[num_horaire+1][1]
1320|             num_horaire+=1
1321|         #num_horaire a atteint un train en dehors de la zone des horaires alors num_horaire ne doit pas être mis en place
1322|         if temps_deb>heure:
1323|             j=1
1324|         else: #num_horaire a atteint la fin de la liste
1325|             j=0
1326|         for k in range(num_horaire-j, -1, -1):
1327|             execute_mise_service(k, heure)
1328|
1329|
1330| def execute_mise_service(num_horaire, heure):
1331|     typ = horaires[num_horaire][0]
1332|     trip_id = horaires[num_horaire][2]
1333|     train = copy.deepcopy(types[typ])
```

```
    no_feuille_route = train[7]
    desti = dest[no_feuille_route].copy()
    destinations.append(desti)
    s_a=direct[no_feuille_route].copy()
    sans_arret.append(s_a)
    numtrain = len(trains)
    trains.append(train)
    gare1 = train[5]
    gare2 = desti[0]
    ind = recherche_indice_quai(gare1,gare2)
    gares[gare1][0][ind][1]+=1 #un train de plus en gare
    tps_litteral = conversion_h_m(heure)
    augmenter_voyageurs_gares(gare1, numtrain, heure)
    if service: #affichage
        nom_gare = correspondance_traffic_DB([gare1], liste_id)[0][1]
        print('train', trip_id, 'mis en service à', nom_gare)
    stop_times.append([numtrain, trip_id, [[gare1, heure, 0, 0]]])
    if profil and trip_id==trip_id_course:
        global numtrain_course
        numtrain_course = numtrain
        print('numtrain_course:', numtrain_course, heure)
        gestion_profil(heure)
    horaires.remove(horaires[num_horaire])  #intérêt du compteur descendant de mise_en_service

##fin de service
#il faut un compteur descendant
def execute_fin_service():
    global fin_de_service
    for k in range(len(fin_de_service)-1, -1, -1):
        [numtrain, numgare] = fin_de_service[k]
        execute_suppr_suivi(numtrain, numgare)
        execute_changement_indice_trains(numtrain)
    fin_de_service=[]


def execute_suppr_suivi(numtrain, numgare):
    global trains, destinations, stop_times, numtrain_course, sans_arret
    index = recherche_indice_suivi(numtrain)
    stop_times_stock.append(stop_times[index])
    trip_id=stop_times[index][1]
    stop_times = stop_times[:index]+stop_times[index+1:] #on enlève ce train des trains suivis
    if service:
        print('il restait',int(total_voyageurs_dans_train(numtrain)),'voyageurs', 'fin de service de', trip_id, numtrain)
    if profil and numtrain_course==numtrain:
        global condition_arret_prgm
        condition_arret_prgm = True  #il ne sert à rien de continuer
        numtrain_course= -1
    trains = trains[:numtrain]+trains[numtrain+1:]
    sans_arret = sans_arret[:numtrain]+sans_arret[numtrain+1:]
    destinations = destinations[:numtrain]+destinations[numtrain+1:]
    ind = recherche_indice_quai(numgare, -1)
    gares[numgare][0][ind][1]-=1 #le train libère le quai

#il faut faire correspondre les numtrains suivis/arretés/course avec les nouveaux numtrains
def execute_changement_indice_trains(numtrain):
    n = len(arret)
```

```
    for k in range(n):
        if arret[k]>numtrain:
            arret[k]-=1

    n=len(ralentissement)
    for k in range(n):
        if ralentissement[k]>numtrain:
            ralentissement[k]-=1

    n = len(stop_times)
    for i in range(n):
        if stop_times[i][0]>numtrain:
            stop_times[i][0]-=1

    global numtrain_course
    if numtrain_course>numtrain:
        numtrain_course-=1


##animation

def creation_artists(num_max_trains, xlim1, ylim1):
    global plot_trains,annotations_trains,plot_gares,annotations_gares,plot_trains_voie,fig1,train_ax
    fig1 = pl.figure(figsize=(8, 8))
    train_ax = pl.axes([0.2,0.1,0.7,0.8], xlim = xlim1, ylim = ylim1)
    activation_boutons()
    train_ax.set_title('traffic RER')
    plot_trains=[]; plot_gares=[]; annotations_trains=[];  annotations_gares=[]; plot_trains_voie=[]
    for i in range(num_max_trains):
        plot_trains.extend(train_ax.plot([], [], marker='+', color='green',markeredgewidth = 8, markersize = 10))
        plot_trains_voie.extend(train_ax.plot([], [],color='red',linewidth = 5))
        annotations_trains.append(train_ax.annotate(i,xy=(-1,-1), color='green', annotation_clip = False, fontsize = 6))
    for k in range(len(points)):
        plot_gares.extend(train_ax.plot([], [],marker='o',color='black',markersize = 1))
        # if exemple and k%6==0:
        #     texte = str(k) + correspondance_traffic_DB([k], liste_id)[0][1]
        # elif exemple:
        #     texte = ''
        texte = str(k)
        annotations_gares.append(train_ax.annotate(texte,xy=(-1,-1), xycoords='data',color='black', annotation_clip = False, fontsize = 9))


def init():
    n = len(points)
    dessiner_gares(n)
    dessiner_graphe(n)
    annoter_gare(n)
    return plot_gares, annotations_gares   #ajouter l'augmentation de la taille des gares en fct des voyageurs

def anim(i, instance):
    global pause
    heure = t_depart+i*dt/60
    if timer:
        print('temps:', conversion_h_m(heure))
    if pause:
        print("l'éxecution est en pause, appuyer sur une touche")
```

```
1446|         pl.waitforbuttonpress(-1)
1447|         pause = not pause
1448|     temps_suivant(heure)
1449|     n = len(points)
1450|     nb_trains = len(trains)
1451|     coef = 0.5
1452|     numeroVoie = 0
1453|     for numtrain in range(nb_trains):
1454|         if est_en_gare(numtrain):
1455|             numgare1 = int(trains[numtrain][5])
1456|             x,y = dessiner_trains_gare(numgare1,numtrain)
1457|         else:
1458|             numgare1, numgare2 = int(trains[numtrain][4]), int(trains[numtrain][5])
1459|             x,y = dessiner_trains_voie(numgare1, numgare2, numtrain, coef, numeroVoie, plot_trains_voie)
1460|             numeroVoie+=1
1461|         dessiner_trains(x,y,numtrain)
1462|     for k in range(numeroVoie, num_max_trains):
1463|         plot_trains_voie[k].set_data([],[])
1464|     for k in range(nb_trains, num_max_trains):
1465|         plot_trains[k].set_data([],[])
1466|         annotations_trains[k].set_position((-1,-1))
1467|     taille_des_gares()
1468|     annoter_gare(n)
1469|     return plot_trains, annotations_trains, plot_trains_voie, plot_gares
1470|
1471|
1472| ##dessin
1473| def dessiner_trains_gare(numgare, numtrain):
1474|     pt = points[numgare]
1475|     return pt[0],pt[1]
1476|
1477| def dessiner_trains_voie(numgare1, numgare2, numtrain, coef, numeroVoie, plot_trains_voie):
1478|     pt1, pt2 = points[numgare1], points[numgare2]
1479|     X,Y=[pt1[0],pt2[0]],[pt1[1],pt2[1]]
1480|     pos,vit = trains[numtrain][0:2]
1481|     vect = X[1]-X[0],Y[1]-Y[0]
1482|     x1,y1 = X[0]+pos*vect[0],Y[0]+pos*vect[1]
1483|     pos0 = pos-vit*dt/distances[numgare1,numgare2]*coef
1484|     x0,y0 = X[0]+pos0*vect[0],Y[0]+pos0*vect[1]
1485|     plot_trains_voie[numeroVoie].set_data([x0,x1],[y0,y1])
1486|     return x1,y1
1487|
1488| def dessiner_trains(x,y,numtrain):
1489|     if annote:
1490|         annotations_trains[numtrain].set_position((x+dimension,y+dimension))  #eviter la  supersposition
1491|     else:
1492|         annotations_trains[numtrain].set_position((-1,-1))
1493|     if exemple:
1494|         plot_trains[numtrain].set_color('red')
1495|     elif total_voyageurs_dans_train(numtrain)>1700:
1496|         plot_trains[numtrain].set_color('red')
1497|     else:
1498|         plot_trains[numtrain].set_color('blue')
1499|     plot_trains[numtrain].set_data(x,y)
1500|
1501| def dessiner_gares(n):
```

```
    for numgare in range(n):
        pt = points[numgare]
        plot_gares[numgare].set_data(pt[0],pt[1])
        scalaire = total_voyageurs_en_gare(numgare)/100
        if scalaire>10:
            scalaire = 10
            plot_gares[numgare].set_color('orange')
        if scalaire<5:
            scalaire = 5
        plot_gares[numgare].set_markersize(scalaire)

def annoter_gare(n):
    for numgare in range(n):
        pt = points[numgare]
        if annote:
            annotations_gares[numgare].set_position((pt[0]+dimension, pt[1]-dimension))
        else:
            annotations_gares[numgare].set_position((-1,-1))


def dessiner_graphe(n):
    for lig in range(n):
        for col in range(n):
            if distances[lig,col]!=0:
                pt1 = points[lig]
                pt2 = points[col]
                X,Y=[pt1[0],pt2[0]],[pt1[1],pt2[1]]
                train_ax.plot(X,Y,color='gray',linewidth = 2.5)

def taille_des_gares():
    n = len(points)
    for numgare in range(n):
        if exemple:
            scalaire = 7
        elif affluence:
            scalaire = total_voyageurs_en_gare(numgare)/500
            if scalaire<5:
                scalaire = 5
            elif scalaire>10:
                scalaire = 10
                plot_gares[numgare].set_color('orange')
        else:
            scalaire = 5
        plot_gares[numgare].set_markersize(scalaire)



## interaction (animation)
def changer_etat(nom_var):
    g = globals()
    g[nom_var]= not g[nom_var]

nom_func=["annote", "pause", "timer", "service", "affluence"]

#on ajoute au dictinnaire des variables global les axes et les boutons
#on les associe au changement d'état de la variable du dictionnaire func_boutons
```

```
1558| def activation_boutons():
1559|     g = globals()
1560|     for i in range(len(nom_func)):
1561|         ax = pl.axes([0.01,0.1*i,0.1,0.1], xticks=[], yticks=[])
1562|         btn = Button(ax, label = nom_func[i])
1563|         nom_button="btn_{}".format(i)
1564|         g[nom_button]=btn
1565|         globals()[nom_button].on_clicked(lambda event,i = i:changer_etat(nom_func[i]))
1566|
1567|
1568|
1569|
1570| ##outil voyageurs
1571| #calcul du montant de voyageurs à numgare sur le quai du train numtrain
1572| #permet d'affiner le temps d'arret
1573| def voyageurs_quai(numgare, numtrain):
1574|     total=0
1575|     freq_gare=gares[numgare][1]
1576|     directions_desservies=destinations[numtrain]
1577|     for destination in freq_gare:
1578|         if destination[0] in directions_desservies:
1579|             total+=destination[1]
1580|     return total
1581|
1582| #calcul du montant total de voyageurs dans la gare toutes directions confondues, a un instant donné
1583| def total_voyageurs_en_gare(numgare):
1584|     total = 0
1585|     gare = gares[numgare]
1586|     frequentations = gare[1]
1587|     for destination in frequentations:
1588|         total+=destination[1]
1589|     return total
1590|
1591| #calcul du montant de voyageurs qui transitent par numgare en une journée
1592| def total_voyageurs_jour(numgare):
1593|     total = 0
1594|     gare = gares[numgare]
1595|     frequentations = gare[1]
1596|     for destination in frequentations:
1597|         total+=destination[2]
1598|     return total
1599|
1600| #calcul du montant de voyageurs à bord
1601| def total_voyageurs_dans_train(numtrain):
1602|     total = 0
1603|     remplissage = trains[numtrain][6]
1604|     for destination in remplissage:
1605|         total+=destination[1]
1606|     return total
1607|
1608|
1609| ##voyageurs et remplissage du train
1610|
1611| #même si tous les passagers en gare ne peuvent pas monter on reset le tps d'attente sur toutes les directions desservies (il ne sert qu'à comparer à la situation
théorique dans laquelle le traffic n'est pas)
1612| #les temps sont ici en minutes
```

```python
def monter_voyageurs(numtrain, numgare, heure):
    freq=gares[numgare][1]
    remplissage=trains[numtrain][6]
    fraction, passagers_montants = calcul_passagers_montant(numtrain, numgare)
    directions_desservies=destinations[numtrain]
    tps_total=0
    nombre=0
    if fraction>0:
        for direction in freq:
            gare_voulue=direction[0]
            if gare_voulue in directions_desservies: #si le train passe par la gare voulue
                passagers_montants_dest=round(direction[1]*fraction, 3)
                minutes_attendues=(heure-direction[3])*60
                tps_total+=minutes_attendues
                nombre+=1
                execute_monter_passagers(passagers_montants_dest, remplissage, gare_voulue)
            direction[3]=heure #reset temps d'attente
    if nombre==0:
        tps_attente_moyen=0
    else:
        tps_attente_moyen=round(tps_total/nombre, 4)
    index=recherche_indice_suivi(numtrain)
    stop_times[index][2][-1][2:4]=[passagers_montants, tps_attente_moyen]


#calcule la fraction des passagers qui va monter dans numtrain en numgare
def calcul_passagers_montant(numtrain, numgare):
    passagers_voulant_monter=voyageurs_quai(numgare, numtrain)
    voyageurs_a_bord=total_voyageurs_dans_train(numtrain)
    if passagers_voulant_monter==0:
        return 0, 0
    if voyageurs_a_bord+passagers_voulant_monter<contenance:
        passagers_montants=passagers_voulant_monter
    else:
        passagers_montants=contenance-voyageurs_a_bord
    if passagers_montants>10: #permet d'alléger les calculs sans vraiment fausser les résultats
        fraction=passagers_montants/passagers_voulant_monter
    else:
        fraction = 0
    return fraction, round(passagers_montants)

def execute_monter_passagers(passagers_montants_dest, remplissage, gare_voulue):
    rajout_destin=True
    for destination in remplissage:  #il y-a-t-il des passagers pour cette direction
        gare_dir=destination[0]
        if gare_voulue==gare_dir:
            destination[1]+=passagers_montants_dest
            rajout_destin=False
    if rajout_destin:
        remplissage.append([gare_voulue, passagers_montants_dest])

def descendre_voyageurs(numtrain, numgare):
    remplissage=trains[numtrain][6]
    for destination in remplissage:
        if destination[0]==numgare:  #les voyageurs sont arrivés à destination
            remplissage.remove(destination)
```

```
            break #ne sert à rien de continuer

## voyageurs et remplissage des gares
'''plages liste les freq relatives des flux de voyageurs découpée par heures'''

#on fait arriver les voyageurs juste avant l'arrivée du train numtrain pour optimiser les calculs et donc que selon les directions desservies
def augmenter_voyageurs_gares(numgare, numtrain, heure):  #deltat est en heures, longueur de plage aussi
    num_p = int(heure)
    directions_desservies=destinations[numtrain]
    freq_gare=gares[numgare][1]
    for destination in freq_gare:
        if destination[0] in directions_desservies:
            voyageurs_par_jour = destination[2]
            deltat = heure- destination[3]
            destination[1]+=plages[num_p]*voyageurs_par_jour*deltat



##outils temps
#int pour enlever les 1.0 -> 1
def conversion_h_m(heure):
    h = int(np.floor(heure))
    mins = int((heure-h)*60)
    return str(h)+':'+str(mins)

def conversion_minutes(liste_h_m):
    liste_h_m = liste_h_m.split(':')
    heure = float(liste_h_m[0])*60+float(liste_h_m[1])
    return heure

def conversion(vit, accel):
    return vit/1000*60, accel/60

##statistiques
'''
temps_cumulé = somme(temps d'attente sur une gare avant de monter dans un train * nb_voyageurs montant dans le train)
temps_cumule_gare
voyageurs_desservis_total
voyageurs_desservis_gare = voyageurs_desservis par gare (liste)
tps_moyen_total = temps_cumulé/voyageurs_desservis_total
tps_moyen_gare = le temps_d'attente moyen par gare (liste)
'''

def statistiques_attente():
    global temps_cumule_gare, voyageurs_desservis_gare, tps_moyen_gare
    temps_cumule, voyageurs_desservis_total = ponctualite_charge()

    #on calcule aussi des stats plus détaillées par gare:
    tps_moyen_gare = zeros.copy()
    for numgare in range(len(gares)):
        if voyageurs_desservis_gare[numgare]!=0:
            tps_moyen_gare[numgare]=temps_cumule_gare[numgare]/voyageurs_desservis_gare[numgare]
        else:
            tps_moyen_gare[numgare]=-1  #non desservie

    if voyageurs_desservis_total!=0:
```

```
        tps_moyen_total = temps_cumule/voyageurs_desservis_total
    else:
        tps_moyen_total = 0
    if affichage:
        print('temps cumulé', round(temps_cumule/60, 2), "heures")
        print(round(voyageurs_desservis_total), 'voyageurs desservis')
        print("soit un tps d'attente moyen de", round(tps_moyen_total,2), "minutes")
        print("pour le detail par gare voir temps_cumule_gare, tps_moyen_gare et voyageurs_desservis_gare")

# calcule le temps attendu par les voyageurs*nombre de voyageurs qui ont attendu
#tps_attendu est en minutes
def ponctualite_charge():
    global temps_cumule_gare, voyageurs_desservis_gare
    voyageurs_desservis_total = 0; temps_cumule = 0
    temps_cumule_gare = zeros.copy(); voyageurs_desservis_gare = zeros.copy()
    for train in stop_times_stock:
        for passage in train[2]:
            [numgare, heure, voy,tps_attendu] = passage
            tps = voy*tps_attendu
            temps_cumule+=tps
            voyageurs_desservis_total+=voy
            temps_cumule_gare[numgare]+=tps
            voyageurs_desservis_gare[numgare]+=voy
    return temps_cumule, voyageurs_desservis_total

## écarts aux horaires en traffic normal - critère de ponctualité
'''il faut faire correspondre stop_times0 qui est [trip_id, [horaire1, horaire2, ...] et dont les trip_id sont ordonnées de manière décroissante
avec stop_times qui est [numtrain, trip_id, [groupe_id1, horaire1], [groupe_id2, horaire2], ...]
il arrive que stop_times contiennent moins d'horaires que stop_times0 lorsque les trains ne finissent pas leur parcours'''
'''dans ces modules il faut avoir au préalable converti les id_traffic id_groupe'''

def ponctualite():
    recuperation_groupe_id()
    carre_ecart = 0
    ecart_flat = 0
    for trip in stop_times_stock:
        trip_id = trip[1]
        index = recherche_index_dichotomie(trip_id)
        trip0 = stop_times0[index]
        dep_times0 = trip0[1]
        passages = trip[2]
        n=len(passages)
        N=len(dep_times0)
        for i in range(N):
            h1 = float(dep_times0[i])
            if i<n:
                h2 = float(passages[i][1])
                carre_ecart+=(h2-h1)**2 #écart en heure^2
                ecart_flat+=h2-h1
    # if ecart_flat<0:
    #     print('modèle en avance sur la théorie', ecart_flat)
    # else:
    #     print('modèle en retard sur la théorie', ecart_flat)
    return np.sqrt(carre_ecart)

```

```
#renvoie l'index dans stop_times0 de trip_id
def recherche_index_dichotomie(trip_id):
    N = len(stop_times0)
    mini = 0; maxi = N-1
    index = int((mini+maxi)/2)
    while mini!=index and maxi!=index:
        if trip_id>stop_times0[index][0]:
            maxi = index
        else:
            mini = index
        index = int((mini+maxi)/2)
    if trip_id!=stop_times0[index][0]:
        print('erreur')
    else:
        return index

##suivi des trains
#numtrain1 arrive en gare2, s'il est suivi on ajoute l'horaire de passage à trains suivis
def suivi_ajout_passage(numtrain1, gare2, heure):
    index = recherche_indice_suivi(numtrain1)
    stop_times[index][2].append([gare2, heure, 0, 0])

#donne les trains dans la gare numgare
def recherche_train_dans_gare(numgare):
    rep=[]
    for numtrain in range(len(trains)):
        train = trains[numtrain]
        if train[4]==-1 and train[5]==numgare:
            rep.append(numtrain)
    return rep

#retourne l'index dans la liste stop_times du train numtrain
def recherche_indice_suivi(numtrain):
    for index in range(len(stop_times)):
        if stop_times[index][0]==numtrain:
            return index
    print('le train', numtrain, 'circulait sans être suivi en suivi_global')

#on remplace dans la liste préexistente le numgare par sa correspondance dans la DB en terme de id_groupe
def recuperation_groupe_id():
    for train in stop_times_stock:
        n=len(train[2])
        for index_passage in range(n):
            numgare = train[2][index_passage][0]
            stop_groupe = liste_id[numgare]
            train[2][index_passage][0]=stop_groupe

##suivi d'un profil de course
#on crée profil_suivi qui est plus précis que juste stoptimes des trains: on ne regarde pas les horaires de passage en gare mais la position à toute heure
#on n'utilisera pas ce module pour calculer ponctualite car on n'est pas sur du profil theorique
def gestion_profil(heure):
    if numtrain_course!=-1:
        train_suiv=trains[numtrain_course]
        pos, vit, accel, tps, gare_dep, gare_ar, remplissage = train_suiv[0:7]
        if gare_ar!=-1:
            vit, accel = conversion(vit, accel)
```

```
            if len(profil_suivi)>0 and profil_suivi[-1][0]==gare_dep:
                #toujours sur le même intergare
                profil_suivi[-1].append([pos, heure, vit, accel, []])
            else:
                profil_suivi.append([gare_dep, gare_ar, [pos, heure, vit, accel, copy.deepcopy(remplissage)]])


def profil_theorique(trip_id, pos_ax):
    global num_horaire_course
    num_horaire_course = recherche_indice_horaire(trip_id)
    index = recherche_index_dichotomie(trip_id)
    stop_time = stop_times0[index][1]
    #il faut récupérer les destinations pour avoir les distances
    num_typ = horaires0[num_horaire_course][0]
    typ = types[num_typ]
    gare_dep = typ[5]
    feuille_route = [gare_dep] + dest[num_typ][:-1]   #on enlève le -1 (fictif) et on rajoute le terminus de départ
    direc = direct[num_typ]
    profil_th = []
    dist_tot = 0
    indice_temps = 0
    n = len(feuille_route)
    for indice_feuille in range(n-1):
        gare_dep = feuille_route[indice_feuille]
        gare_ar = feuille_route[indice_feuille+1]
        t_dep = stop_time[indice_temps]
        if gare_dep not in direc: #le train s'arrete bien et donc il est pris en compte dans stop_time
            pos_ax.scatter(x=t_dep, y=dist_tot, s=8, color='blue')
            indice_temps+=1
        #dans le cas contraire, on ne modifie pas indice_temps (les directs ne sont pas pris en compte dans stop_times) et on ne plot pas
        dist_tot+= distances[gare_dep][gare_ar]
    t_ar = stop_time[indice_temps]
    pos_ax.scatter(x=t_ar, y=dist_tot, s=8, color='blue')



def dessin_course(vit_ax, pos_ax, remp_ax, accel_ax, color):
    n=len(gares)
    global remp_liste
    t_liste=[]; t_cut_liste=[]; pos_liste=[]; vit_liste=[]; accel_liste=[]; bins=[]
    remp_liste=[[] for k in range(n)]; weight_liste=[[] for k in range(n)]; labels=['']*n
    distance_parcourue = 0
    compt = 0
    # lim_remp = 17; lim_vit = 40; inf = 15 #pour l'étude de trip_id_course='115054849-1_16156'
    lim_remp = np.inf; lim_vit = np.inf; inf = 0 #pour l'étude de trip_id_course='115072256-1_14393'
    t0=profil_suivi[0][2][1]
    t0_cut=t0
    for intergare in profil_suivi:
        gare_dep = intergare[0]
        gare_ar = intergare[1]
        # print(correspondance_traffic_DB([gare_dep], liste_id)[0][1])
        # print(gare_dep)
        if gare_dep!=-1:
            dist_gares = distances[gare_dep][gare_ar]
            if dist_gares==0:
                print('ce train est direct')
```

```
        else:
            dist_gares = 0

        for pos, heure, vit, accel, remplissage in intergare[2:]:
            dist = pos*dist_gares
            pos_liste.append(dist+distance_parcourue)
            t_liste.append(heure)
            if compt<inf:
                t0_cut=heure
            else:
                if compt<lim_vit:
                    t_cut_liste.append(heure)
                    vit_liste.append(vit)
                    accel_liste.append(accel)
                    tf_cut=heure
        if gare_dep!=-1:
            pos, heure, vit, accel, remplissage = intergare[2]
            if compt<lim_remp:
                bins.append(heure)
                tot=0
                for numgare, passagers in remplissage:
                    if passagers>40:
                        remp_liste[numgare].append(heure)
                        weight_liste[numgare].append(passagers)
                        remp_ax.text(heure, tot+passagers/2, str(int(passagers)), fontsize=7)
                        name = correspondance_traffic_DB([numgare], liste_id)[0][1]
                        if passagers>200:
                            labels[numgare]=name
                        tot+=passagers
        tf = intergare[-1][1]
        bins.append(tf)
        distance_parcourue+=dist_gares
        pos_ax.plot([tf], [distance_parcourue], marker='o', markersize = 4, color=color)
        compt+=1
    pos_ax.plot(t_liste, pos_liste, color=color)
    vit_ax.plot(t_cut_liste, vit_liste, marker='o', markersize = 4)
    accel_ax.plot(t_cut_liste, accel_liste)
    remp_ax.hist(remp_liste, weights=weight_liste, bins=bins, histtype='barstacked', label=labels)
    remp_ax.legend(loc='upper right', fontsize=8)
    return t0, tf, t0_cut, tf_cut


def reperes(remp_ax, vit_ax, accel_ax, t0, tf, t0_cut, tf_cut):
    v_nom, ac_nom = conversion(vitesse_nominale, accel_nominale)
    v_ar, fr_nom = conversion(vitesse_arrivee, freinage_nominal)
    lims=[t0, tf]
    lims_cut=[t0_cut, tf_cut]
    vit_ax.plot(lims, [v_nom, v_nom])
    vit_ax.plot(lims, [v_ar, v_ar])
    accel_ax.plot(lims, [ac_nom, ac_nom])
    accel_ax.plot(lims, [fr_nom, fr_nom])
    vit_ax.set_xlim(lims)
    accel_ax.set_xlim(lims)
    remp_ax.plot(lims, [contenance, contenance])
    remp_ax.set_xlim(lims)

```

```
1949|
1950|
1951| def comparaison(trip_id):
1952|     fig2, axes = pl.subplots(2, 2, figsize=[23, 16])
1953|     for i in range(2):
1954|         for j in range(2):
1955|             axes[i][j].set_xlabel('temps (heures)')
1956|     [[pos_ax, remp_ax], [vit_ax, accel_ax]] = axes
1957|     pos_ax.set_title('position modèle et théorique')
1958|     pos_ax.set_ylabel('distance depuis terminus (m)')
1959|     remp_ax.set_title('remplissage modèle')
1960|     remp_ax.set_ylabel("nombre de passagers en fonction de la gare d'arrivée")
1961|     accel_ax.set_title('acceléréation modèle m/s^2')
1962|     vit_ax.set_title('vitesse modèle')
1963|     vit_ax.set_ylabel('vitesse (km/h)')
1964|     profil_theorique(trip_id, pos_ax)
1965|     color = 'black'
1966|     t0, tf, t0_cut, tf_cut = dessin_course(vit_ax, pos_ax, remp_ax, accel_ax, color)
1967|     reperes(remp_ax, vit_ax, accel_ax, t0, tf, t0_cut, tf_cut)
1968|     pl.savefig('calage du modèle (profil course de traffic)', dpi=300,  bbox_inches='tight')
1969|     pl.show()
1970|
1971| # renvoie l'indice de trip_id dans horaires0
1972| def recherche_indice_horaire(trip_id):
1973|     n=len(horaires0)
1974|     for k in range(n):
1975|         if trip_id==horaires0[k][2]:
1976|             return k
1977|     print('erreur de trip_id, inexistant dans horaires0')
1978|
1979|
1980| ## utilisation dans la db
1981| #on remplit avec notre modèle stop_times_modif
1982| #il y a des petites approximations sur l'heure dans la conversion en h_m c'est pourquoi la db n'est utilisée que pour du debug et pas pour le calcul de l'écart
pour la descente de gradient
1983| def reecriture_DB():
1984|     conn = sql.connect(r"F:/informatique/TIPE/database/produit exploitable/GTFS.db")
1985|     c = conn.cursor()
1986|     c.execute('delete from stop_times_modif')
1987|     for train in stop_times_stock:
1988|         trip_id = train[1]
1989|         for passage in train[2]:
1990|             [id_groupe, departure_time, voy, tps]=passage
1991|             departure_time = conversion_h_m(departure_time)
1992|             c.execute('''
1993|             insert into stop_times_modif(route_id, trip_id, departure_time, id_groupe, stop_sequence)
1994|             values(?,?,?,?,?)''', (route_id, trip_id, departure_time, id_groupe, stop_sequence))
1995|     conn.commit()
1996|     conn.close()
1997|
1998| def ecart_db(affichage=False):
1999|     if affichage:
2000|         ec_fig, ax=pl.subplots(figsize=[23, 16])
2001|     conn = sql.connect(r"F:/informatique/TIPE/database/produit exploitable/GTFS.db")
2002|     c = conn.cursor()
2003|     c.execute('''
```

```
    select s1.departure_time, s2.departure_time, s1.trip_id
    from stop_times as s1
    join stop_times_modif as s2
        on s1.trip_id = s2.trip_id and s1.stop_sequence = s2.stop_sequence
    ''')
    carre_ecart = 0; ecart_flat = 0
    for h1, h2, trip_id in c:
        h1 = conversion_minutes(h1); h2 = conversion_minutes(h2)
        carre_ecart+=(h2-h1)**2 #écart en min^2
        ecart_flat+=h2-h1
        if affichage:
            ax.scatter(h1, 0); pl.scatter(h2, 1)
            ax.annotate(s='', xy=(h1, 0), xytext=(h2, 1),  arrowprops=dict(arrowstyle="->", lw=0.5, mutation_scale=1))
    conn.close()
    print('écart réduit ', np.sqrt(carre_ecart))
    if affichage:
        pl.savefig('régulation', dpi=300, bbox_inches='tight')
        pl.show()



##perturbation
def execute_arret_force(numtrain, tfin):
    arret_force.append([numtrain, tfin])
    if trains[numtrain][4]!=-1: #pas en gare
        execute_arret(numtrain)
    #si en gare, le train ne redémarre tout simplement pas tant qu'il est en arret_force

def gestion_arret_force(heure):
    n=len(arret_force)
    for k in range(n-1, -1, -1):
        if arret_force[k][1]<heure:
            arret_force.remove(arret_force[k])

# teste si numtrain est en arret forcé
def arret_force_de(numtrain):
    for train in arret_force:
        if train[0]==numtrain:
            return True
    return False


##régulation
#on détermine l'horizon de régulation en supposant connaitre la durée de la régulation

#heure est en heures
# 0.2 h = 12 min
def variables_de_pert(duree, tm, g1, g2):
    global duree_pert_est, tmin, gare_pert1, gare_pert2, horizon_retarde
    duree_pert_est=duree
    tmin=tm
    facteur_pert=4
    gare_pert1=g1; gare_pert2=g2
    horizon_retarde = duree_pert_est*facteur_pert

```

```
#on veut que le dernier train à partir ne soit pas retardé
def retarder_departs(heure):
    global trains_retardes
    [name1, name2] = correspondance_traffic_DB([gare_pert1, gare_pert2], liste_id)
    conv_tmin = conversion_h_m(tmin)
    conv_tmax =  conversion_h_m(tmin+duree_pert_est)
    print("\n \n perturbation se déroulant depuis", conv_tmin, "jusqu'à", conv_tmax, "entre ", name1[1], gare_pert1, "et", name2[1], gare_pert2, '\n \n',
'numtrain course=', numtrain_course)
    print(destinations[numtrain_course])
    trains_retardes = []
    N=len(trains)
    # print(numtrain_course, trains[numtrain_course], destinations[numtrain_course], gare_pert1, gare_pert2)

#on arrete les trains
    for numtrain in range(N):
        ret = retard(heure)
        destin = destinations[numtrain]
        if test_destination(destin):
            trains_retardes.append(numtrain)
            execute_arret_force(numtrain, heure+ret)
    if consigne_pert:
        dessin_regulation()
        pl.plot([heure, heure+ret], [1, 2], marker='o', color='black')
    print('la perturbation a occasioné le retard des trains ', trains_retardes, " qui sont en service ainsi que le retard des départs d'autres trains passant
entre ", name1[1], "et", name2[1])
#on retarde directement le départ des trains considérés
    num_h=0; dep_time=0 #initialisation
    print('horizon retardé : ', conversion_h_m(horizon_retarde), ' min \n')
    while dep_time<horizon_retarde + tmin:
        destin = dest[num_h]
        dep_time=horaires[num_h][1]
        # print(correspondance_traffic_DB(destin, liste_id), test_destination(destin), '\n')
        if test_destination(destin):
            ret = retard(dep_time)
            hor_dep = horaires[num_h][1]
            new_hor = hor_dep + ret
            horaires[num_h][1] = new_hor
            if consigne_pert:
                pl.plot([hor_dep, new_hor], [1, 2], marker='o', color='black')

        num_h+=1
    if consigne_pert:
        pl.savefig('retards en consigne des trains', dpi=300,  bbox_inches='tight')
        pl.show()


def retard(heure):
    return (tmin+horizon_retarde-heure)*duree_pert_est/horizon_retarde

#on regarde si la direction définie par gare_pert1 et gare_pert2 est dans destination
def test_destination(destin):
    n = len(destin)
    for k in range(n-1):
        if gare_pert1==destin[k] and gare_pert2==destin[k+1]:
            return True
    return False
```

```
2114|
2115| def mise_en_place_pert(heure):
2116|     if heure<tmin and heure+dt/60>=tmin:
2117|         retarder_departs(heure)
2118|
2119| def dessin_regulation():
2120|     pl.figure(figsize=[23, 16])
2121|     pl.xlabel('temps (heures)')
2122|     pl.title('retards en consigne des trains')
2123|     pl.plot([tmin, horizon_retarde + tmin], [1, 1])
2124|     pl.plot([tmin, horizon_retarde + tmin], [2, 2])
2125|
2126|
2127|
2128|
2129| ##paramètres du modèle et calage
2130| '''Tout est mis en m/min
2131| vit commerciale : 50 km.h-1
2132| vit_nominale : 65 kmh
2133| capacité totale 1700
2134| on calcule accel nominale=3912
2135| '''
2136|
2137| vitesse_nominale = round(65000/60)  #65km/h en m/min
2138| vitesse_arrivee=round(vitesse_nominale/3) #le train doit arriver en gare à allure raisonnable
2139| contenance = 1700
2140| tps_arret = 0.5 #30 sec
2141| tps_population = 1/10000 #pour 10000 voyageurs en gare, le train attendra une minute de plus (à ajuster en fonction des paramètres de flux)
2142| distance_freinage = 300
2143| distance_demarrage = 150 #longueur d'une gare
2144| distance_securite = 800
2145| distance_securite_mini = 500 #deux trains ne doivent jamais être à moins de 500m
2146| accel_nominale=round((vitesse_nominale)**2/2/distance_demarrage) #on choisit accel_dem de facon à satisfaire dist_dem et vit_dem qui sont des params plausibles
2147| freinage_nominal=round(-accel_nominale/1.5)
2148|
2149| var=[vitesse_nominale, vitesse_arrivee, contenance, tps_arret, tps_population, distance_freinage, distance_demarrage, accel_nominale, freinage_nominal]
2150| delt0=300 #m/min
2151| delt1=300 #m/min
2152| delt2=300 #passagers
2153| delt3=0.3 #min
2154| delt4=tps_population/2
2155| delt=[delt0, delt1, delt2, delt3, delt4]
2156|
2157| bornes=[[var[k]-delt[k], var[k]+delt[k]] for k in range(5)]
2158| variabs=[var[k] for k in range(5)]
2159| infinitesimaux=[50, 0.05, 0.1, 30, 30]
2160| '''' minimum(retour_ecart, var, bornes, infinitesimaux, 1, 0.01, False)
2161| la derive_partielle de tps_population est nul ...
2162|
2163| Lorsque dt est petit (9 sec), l'exécution du programme est plus lente et les trains sont en avance sur la théorie
2164| pour dt grand (20 sec), les trains sont en retard sur la théorie
2165| Les paramètres dépendent du dt choisi...
2166|
2167| monteCarlo(retour_ecart, bornes, 10)
2168|
2169| '''
```

```
2170|
2171| 'résultat du monte carlo :'
2172| # vitesse_nominale = 1150 #contre 1083 m/min dans le modèle original.
2173| #cela corrrespond à 69 km/h
2174| # vitesse_arrivee = 300 #contre 361 m/min --> faible différence
2175| #surtout il y a une forte dispertsion de vitesse arrivée aux différents minima --> peu d'impact dans la modélisation
2176|
2177| #trop de fluctuation pour tps_arret
2178|
2179| #on ne peut pas conclure sur contenance et sur tps_population car le modèle de minimisation ne prend en compte qu'un seul train --> les flux à bord du train et à
quai ne sont donc pas réalistes
2180|
2181|
2182| ## initialisation
2183| try:
2184|     test = horaires0[0]
2185| except NameError: #si c'est la première fois
2186|     donnees_GTFS("810:B")
2187|
2188|
2189|
2190|
2191|
2192|
2193|
2194| ##traitement BDD.py
2195| import sqlite3 as sql
2196| import numpy as np
2197| import time
2198| import matplotlib.pyplot as pl
2199| from algorithmes_de_minimisation import minimiser, minimum
2200|
2201| #nous utiliserons deux connections pour pouvoir effectuer des actions à l'intérieur d'une boucle générée par la première connection
2202| #(écrire avec c2 à l'intérieur de c provoque des erreurs)
2203| conn = sql.connect(r"F:/informatique/TIPE/database/produit exploitable/GTFS.db")
2204| c = conn.cursor()
2205| c2 = conn.cursor()
2206|
2207| ##outils
2208| def traitement_tuple(L):
2209|     rep=[]
2210|     for tupl in L:
2211|         rep.append(tupl[0])
2212|     return rep
2213|
2214| def somme(L, ind):
2215|     s=0
2216|     for l in L:
2217|         s+=l[ind]
2218|     return s
2219|
2220| def reverse(liste):
2221|     N = len(liste)
2222|     reverse_liste=[]
2223|     for num in range(N-1,-1,-1):
2224|         reverse_liste.append(liste[num])
```

```
    return reverse_liste

def recuperer_id_groupe(x, y):
    c.execute('''
    select id_groupe
    from stops_groupe
    where (stops_groupe.x between {a}- 500 and {a}+ 500) and (stops_groupe.y between {b}- 500 and {b}+ 500)
    '''.format(a=x, b=y))
    return c.fetchall()

def recuperer_liaisons(x, y):
    c.execute('''
    select stops_groupe.stop_name, stops_groupe.id_groupe
    from stops_groupe
    join graphe
        on graphe.to_id_groupe=stops_groupe.id_groupe
    where graphe.from_id_groupe in(
        select id_groupe
        from stops_groupe
            where (stops_groupe.x between {a}- 300 and {a}+ 300) and (stops_groupe.y between {b}- 300 and {b}+ 300))
    '''.format(a=x, b=y))
    return c.fetchall()

def recuperer_x_y(id_groupe):
    c2.execute('select x,y from stops_groupe where id_groupe={}'.format(id_groupe))
    return c2.fetchone()

def calcul_distance(dernier_id_groupe, id_groupe):
    XY=[]
    for id_g in [dernier_id_groupe, id_groupe]:
        c.execute('select x, y from stops_groupe where id_groupe={}'.format(id_g))
        XY.append(c.fetchone())
    [(x1,y1), (x2,y2)]=XY
    d=dist(x1, y1, x2, y2)
    return d

def dist(x1, y1, x2, y2):
    return np.sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2))

def random_color():
    return [np.random.rand()*0.8 for k in range(3)]

##paramètres du système lambert93
def conversion_rad(angle_degre):
    return angle_degre*np.pi/180

def latitude_iso(lat_rad):  #reparamétrage symétrique de la latitude
    sin=np.sin(lat_rad)
    p1=np.arctanh(sin)
    p2=np.arctanh(e*sin)
    result=p1-e*p2
    return result

xs=700000; ys=12655612.05 #coordonnées du pole nord dans le système lambert93
phi1=conversion_rad(44); ss1=np.sin(phi1); cs1=np.cos(phi1) # Premier parallèle automécoïque ie lattitude de sécance
phi2=conversion_rad(49); ss2=np.sin(phi2); cs2=np.cos(phi2) # Deuxième parallèle automécoïque
```

```
phi0=(phi1+phi2)/2
a=6378388 #grand axe de l'ellipse en m de la Terre ie rayon à l'équateur
e=0.08248325676 #excentricité de l'ellipse de la Terre
gN1=a/np.sqrt(1-(e*ss1)**2); gl1=latitude_iso(phi1)  #grande normales aux sécantes (meilleur rayon de courbure)
gN2=a/np.sqrt(1-(e*ss2)**2); gl2=latitude_iso(phi2)
lon0=conversion_rad(3)  #méridien de greenwitch par rapport à celui de Paris en degrés
expo=np.log(gN2/gN1*cs2/cs1) / (gl1-gl2) #exposant de la projection
C=((gN1 * cs1) / expo) * np.exp(expo * gl1)

##conversion WGS84_to_lambert93
def WGS84_to_lambert93(lat, lon):
    lat_rad, lon_rad = conversion_rad(lat), conversion_rad(lon)
    gl=latitude_iso(lat_rad)
    R = C*np.exp(-expo*gl)
    teta = expo*(lon_rad-lon0)
    X = xs + R*np.sin(teta)
    Y = ys - R*np.cos(teta)
    return X, Y

##fonction commune aux flux
#il faut avoir au préalable rempli le cursor
def representation_flux(nb_title, seuil, xn=600, xm=715, yn=6800, ym=6905):
    fig_mob=pl.figure(figsize=[23, 16])
    mob_ax=pl.axes(title='flux de personnes émis par la ville par jour pour motif professionnel(Lambert 93 en km)')
    mob_ax.axis('equal')
    X, Y, S = [], [], []
    compt=0; compt2=0
    for x, y, flux, name in c:
        x_k, y_k = x/1000, y/1000
        if x_k>xn and x_k<xm and y_k>yn and y_k<ym:
            S.append(flux/1000)
            X.append(x/1000)
            Y.append(y/1000)
            compt2+=1
            if flux/1000>seuil:
                compt+=1
                if compt%4==0:
                    mob_ax.text(x=x_k, y=y_k, s=name)
            else:
                if compt2%nb_title==0:
                    mob_ax.text(x=x_k, y=y_k, s=name)
    scat = mob_ax.scatter(X, Y, s=S)
    grid(mob_ax)
    handles, labels = scat.legend_elements(prop="sizes")
    mob_ax.legend(handles, labels, loc='lower right', title="nombre d'habitants \n(en milliers)")



##mobilites pro -flux emis par VILLE
def conversion_mobilites():
    c.execute('select cle, lat, lon from mobilites_pro')
    compt=0
    for cle, lat, lon in c:
        x, y = WGS84_to_lambert93(lat, lon)
        c2.execute('update mobilites_pro set x={x}, y={y} where cle={c}'.format(x=x, y=y, c=cle))
        compt+=1
```

```
2337|         if compt%100==0:
2338|             print(compt)
2339|     conn.commit()
2340|
2341| def representation_mobilites():
2342|     c.execute('select x, y, flux_emis_jour, name from mobilites_pro')
2343|     representation_flux(45, 20)
2344|     pl.savefig('flux émis par ville (mobilités pro)', dpi=300, bbox_inches='tight')
2345|     pl.show()
2346|
2347| ##flux emis par POLE-ROUTE_ID (frequentation_m)
2348|
2349| '''on connait les flux émis par chaque ville, on répartit ce flux entre les différents poles (frequentation_m)
2350| puis on suppose que le flux dans les transports en commun est proportionnel a ce flux tous transports confondus
2351| on va donc faire en sorte que ce facteur de prop donne un flux émis par jour dans chaque gare le plus proche de la fréquentation de référence donnée par RATP'''
2352|
2353| def remplir_nb_trips():
2354|     c.execute('''
2355|     select trips.route_id, count(trips.trip_id)
2356|     from trips
2357|     join calendar
2358|         on calendar.service_id=trips.service_id
2359|         where calendar.start_date<20200522 and calendar.stop_date>20200522
2360|     group by trips.route_id''')
2361|     for route_id, nb_trips in c:
2362|         c2.execute('update routes set nb_trips={n} where route_id="{r}"'.format(n=nb_trips, r=route_id))
2363|     conn.commit()
2364|
2365| modes=[0.8, 4.2, 2.7, 3.9] #en millions de déplacements par jour selon les routes_types
2366| # 0 tramway; 1métro; 2  RER; 3    bus
2367| #on va donc comparer les contenances relatives
2368| remplissage_contenance=[0]*4
2369| def contenances():
2370|     c.execute('select sum(nb_trips) from routes group by route_type')
2371|     repart = c.fetchall()
2372|     for k in range(4):
2373|         remplissage_contenance[k] = modes[k]*10**6/repart[k][0]
2374| #les contenances trouvés sont nettement inférieures à la réalité mais elles donnent un poids relatif
2375| #certaines courses ont propablement été comptées en multiple
2376|
2377| def remplir_flux_emis():
2378|     c.execute('update flux_emis set frequentation_m=0')
2379|     contenances()
2380|     c.execute('''
2381|     select mobilites_pro.code_commune, mobilites_pro.flux_emis_jour
2382|     from stops_groupe
2383|     join mobilites_pro
2384|         on mobilites_pro.code_commune=stops_groupe.code_commune
2385|     group by mobilites_pro.code_commune
2386|     ''')
2387|     #on ne sélectionne que les codes communes qui ont des pôles
2388|     compt=0
2389|     for code_commune, flux_ville in c:
2390|         execute_repartition_flux(code_commune, flux_ville)
2391|         compt+=1
2392|         if compt%50==0:
```

```
            print(compt)
            conn.commit()
    conn.commit()

def execute_repartition_flux(code_commune, flux_ville, affichage=False, ville='à remplir'):
    if affichage:
        contenances()
    c2.execute('''
    select s.id_groupe, graphe.route_id, graphe.route_type, nb_trips, routes.route_short_name
        from graphe
        join (select id_groupe
                from stops_groupe
                where code_commune={}) as s
            on s.id_groupe=graphe.from_id_groupe
        join routes
            on routes.route_id=graphe.route_id
            where graphe.route_type>=0 and graphe.route_type<=3
        order by graphe.route_type, graphe.route_id'''.format(code_commune))
    #on récupère, toutes les liaisons dont le departure_id est dans la ville de code_commune donné
    weight=[]
    dernier_route_id='init'
    #on répartit le flux_ville entre les différentes lignes de la ville
    #le poids de chaque ligne est nb_trips*contenance
    #fact est donc le nombre de voyageurs qui transitent par la ligne route_id
    for id_groupe, route_id, route_type, nb_trips, name in c2:
        if route_id==dernier_route_id:
            weight[-1][-1]+=(id_groupe,)
        else:
            dernier_route_id=route_id
            fact=remplissage_contenance[int(route_type)]*nb_trips
            weight.append([fact, route_type, name, route_id, (id_groupe,)])
        print(id_groupe, route_id, nb_trips, name, fact)
    tot_modes=somme(weight, 0)
    if affichage:
        print(weight)
        repr_distrib_ville(weight, flux_ville, ville, tot_modes)
    #puis pour chaque ligne, on répartit ce flux entre les poles (de manière équitable puisqu'on suppose que l'attractivité se joue à l'echelle d'une ville)
    for ligne in weight:
        fact, route_type, name, route_id, poles = ligne
        freq_ligne = fact/tot_modes*flux_ville
        freq_pole = freq_ligne/len(poles)
        # print('\n'+name)
        for pol in poles:
            update_freq_m(pol, freq_pole, route_id)

def update_freq_m(pol, freq_pole, route_id):
    c2.execute('select frequentation_m from flux_emis where id_groupe={i} and route_id="{rid}"'.format(i=pol, rid=route_id))
    freq=c2.fetchone()
    if freq==None:
        c2.execute('insert into flux_emis (id_groupe, route_id, frequentation_m, frequentation_c) values (?,?,?,?)', (pol, route_id, freq_pole, 0))
    else:
        freq=freq[0]
        c2.execute('''
        update flux_emis
        set frequentation_m={f}
            where id_groupe={p} and route_id="{rid}"
```

```
        '''.format(p=pol, f=freq_pole+freq, rid=route_id))


def repr_distrib_ville(weight, flux_ville, ville, tot_modes):
    sizes=[[] for k in range(4)]
    labels=[[] for k in range(4)]
    labels_globaux=['tramway', 'metro', 'RER', 'bus']
    fig_rep = pl.figure(figsize=[23, 16])
    X = [(1, 3, 1), (2, 3, 2), (2, 3, 3), (2, 3, 5), (2, 3, 6)]
    axes=[]
    for nrows, ncols, plot_number in X:
        sb=fig_rep.add_subplot(nrows, ncols, plot_number)
        axes.append(sb)
    rep_ax=axes[0]
    rep_ax.set_title('répartition des flux émis ('+str(flux_ville)+' voyageurs) en fonctions des différents modes à '+ville)
    for ligne in weight:
        fact, route_type, name, route_id, poles = ligne
        freq_ligne = fact/tot_modes*flux_ville
        print(freq_ligne)
        if route_type<4 and freq_ligne>1:
            sizes[route_type].append(freq_ligne)
            labels[route_type].append(name)
    t=[sum(sizes[k]) for k in range(4)]
    print(sizes, sum(t))
    sizes_globaux=[sum(sizes[k]) for k in range(4)]
    for k in range(3, -1, -1):
        axes[k+1].pie(sizes[k], labels=labels[k], autopct = lambda x: str(int(x)) + '%')
        if len(sizes[k])>0:
            axes[k+1].set_title('répartition par lignes de '+labels_globaux[k])
        if sizes_globaux[k]==0:
            sizes_globaux=sizes_globaux[:k]+sizes_globaux[k+1:]
            labels_globaux=labels_globaux[:k]+labels_globaux[k+1:]
    rep_ax.pie(sizes_globaux, labels=labels_globaux, autopct= lambda x : int(x*flux_ville/100))
    pl.savefig('répartition modale test', dpi=300, bbox_inches='tight')
    pl.show()
#execute_repartition_flux(75105, 16842.0, affichage=True, ville='Paris 05')

def representation_flux_emis():
    c.execute('''
    select x, y, sum(frequentation_m), ""
    from flux_emis
    join stops_groupe
        on stops_groupe.id_groupe=flux_emis.id_groupe
    group by flux_emis.id_groupe''')
    representation_flux(500, 10)
    pl.savefig('flux émis par pôle', dpi=300, bbox_inches='tight')
    pl.show()




##normalisation du flux emis
'''deux approches:
1) on veut minimiser les écarts au carré de flux émis en choisissant le facteur multiplicatuif
2) on veut que la totalité des flux émis par les gares de la RATP corresponde aux données
'''
```

```
#valeur de référence donnée par RATP (frequentation_c)

def calcul_ecart(facteur_population):
    c.execute('''
    select sum(frequentation_m), sum(frequentation_c)
    from flux_emis
    group by id_groupe''')
    ecart=0; ecart_flat=0
    for frequentation_m, frequentation_c in c:
        if frequentation_c!=0:
            # ecart_flat+=abs(frequentation_m*facteur_population-frequentation_c)
            ecart+=np.sqrt((frequentation_m*facteur_population-frequentation_c)**2)
    return np.sqrt(ecart)

def calcul_ecart_par_ligne(facteur_population):
    c.execute('''
    select frequentation_m, frequentation_c
    from flux_emis
    where frequentation_c!=0''')
    ecart=0; ecart_flat=0
    for frequentation_m, frequentation_c in c:
        if frequentation_c!=0:
            # ecart_flat+=abs(frequentation_m*facteur_population-frequentation_c)
            ecart+=np.sqrt((frequentation_m*facteur_population-frequentation_c)**2)
    return np.sqrt(ecart_flat)

def calcul_facteur_normalisation():
    c.execute('''
    select sum(frequentation_m), sum(frequentation_c)
    from flux_emis
    group by id_groupe''')
    sum_c=0; sum_m=0
    for frequentation_m, frequentation_c in c:
        if frequentation_c!=0:
            sum_c+=frequentation_c
            sum_m+=frequentation_m
    K = sum_c/sum_m
    return K

# calcul_facteur_normalisation()
# >> 4.169361260527865

#fonction vectorielle pour le gradient descent
def calcul_ecart_vecto(facteur_population):
    f=facteur_population[0]
    return calcul_ecart(f)

def repr_ecarts():
    X=np.arange(0, 5, 0.05)
    Y=[]
    for x in X:
        Y.append(calcul_ecart_par_ligne(x))
    fig, ax = pl.subplots(figsize=[24, 12])
    title='valeur absolue des écarts entre les flux émis selon les données RATP et selon le modèle 1'
    ax.set_title(title)
    ax.plot(X, Y)
```

```
    pl.xlabel('facteur_population')
    pl.savefig(title, dpi=300, bbox_inches='tight')
    pl.show()

'''
minimiser(0, 5, 0.0001, calcul_ecart)=1.9097520275672006 par méthode du nombre d'or
minimum(calcul_ecart_vecto, [1], [[0, 5]], [0.001], 0.001, 0.01, False) par descente de gradient
'''

facteur_population=1.9
# facteur_population=2.1 #par ligne carré
# facteur_population=3.34 #carré
# facteur_population=1.9 #flat


def scalairisation_flux_emis():
    c.execute('select id_groupe, route_id, frequentation_m from flux_emis')
    for id_groupe, route_id, frequentation_m in c:
        c2.execute('update flux_emis set frequentation_m={}'.format(facteur_population*frequentation_m))
    conn.commit()

def repr_ecarts_ligne(par_ligne=False, xn=630000, xm=670000, yn=6845000, ym=6875000):
    fig_emis, axes = pl.subplots(1,2, figsize=[24, 12])
    ax_m, ax_c = axes
    ax_m.axis('equal'); ax_c.axis('equal'); grid(ax_m); grid(ax_c)
    ax_c.set_title("flux emis par jour d'après les données RATP \n soit 289 pôles (Lambert 93 en km)")
    ax_m.set_title("flux modélisés sur ces mêmes pôles")
    #1=ax_m (modèle) ; 2=ax_c (théorie)
    if par_ligne:
        c.execute('''
        select stops_groupe.x, stops_groupe.y,  frequentation_m, frequentation_c, stop_name
        from stops_groupe
        join flux_emis
            on flux_emis.id_groupe=stops_groupe.id_groupe
        where (stops_groupe.x between {xn} and {xm} ) and (stops_groupe.y between {yn} and {ym} )
        '''.format(xn=xn, xm=xm, yn=yn, ym=ym))
    else:
        c.execute('''
        select stops_groupe.x, stops_groupe.y,  sum(frequentation_m), sum(frequentation_c), stop_name
        from stops_groupe
        join flux_emis
            on flux_emis.id_groupe=stops_groupe.id_groupe
        where (stops_groupe.x between {xn} and {xm} ) and (stops_groupe.y between {yn} and {ym} )
        group by stops_groupe.id_groupe
        '''.format(xn=xn, xm=xm, yn=yn, ym=ym))
    X, Y, S1, S2 = [], [], [], []
    compt=0
    for x, y, f1, f2, stop_name in c:
        if f2!=0:
            compt+=1
            xs, ys = x/1000, y/1000
            X.append(xs)
            Y.append(ys)
            s1, s2 = f1*facteur_population/1000, f2/1000
            # print(s1, s2)
            S1.append(s1)
```

```
            S2.append(s2)
            if s1==0:
                ax_m.scatter(xs, ys, s=4, color='grey')
            if s1>500 or s2>300:
                ax_m.text(xs, ys, s="")
                ax_c.text(xs, ys, s="")
    scat1 = ax_m.scatter(X, Y, s=S1)
    scat2 = ax_c.scatter(X, Y, s=S2)
    handles1, labels1 = scat1.legend_elements(prop="sizes", num=np.arange(10, 60, 10))
    handles2, labels2 = scat2.legend_elements(prop="sizes", num=np.arange(10, 60, 10))
    ax_m.legend(handles1, labels1, loc='lower right', title='flux emis par jour (milliers de déplacements)')
    ax_c.legend(handles2, labels2, loc='lower right', title='flux emis par jour (milliers de déplacements)')
    pl.savefig('flux emis sur la ligne B '.format(facteur_population), dpi=300, bbox_inches='tight')
    pl.show()


## suppression des exceptions
#on prend les trains rajoutés n'assurant pas un trajet régulier et on les supprime
def exceptions():
    c.execute('''
    delete from stop_times
    where stop_times.trip_id in(
    select trips.trip_id
    from trips
    join calendar_dates
        on calendar_dates.service_id=trips.service_id
    where exception_type=1 and calendar_dates.service_id not in(
            select service_id
                from calendar))''')
    c.execute('''
    delete from trips
    where trip_id in(
    select trips.trip_id
    from trips
    join calendar_dates
        on calendar_dates.service_id=trips.service_id
    where exception_type=1 and calendar_dates.service_id not in(
            select service_id
                from calendar))''')
    conn.commit()

##creation de groupe
#dans la DB, plusieurs stop_id designent la même gare (un stop_id par ligne de la gare + un stop_id par stop_area (=entrée de la gare) )
def enlever_proche(liste):
    compt = 0
    non_traite = liste
    stopid_groupeid=[]
    while len(non_traite)>1:
        stop_id1, lat1, lon1 = non_traite[0]; stop_id2, lat2, lon2 = non_traite[1]
        non_traite.remove(non_traite[0])
        stopid_groupeid.append([compt, stop_id1])
        indice = 0
        while  coordonnee_proche(lat1, lat2, eps_lat) and indice<len(non_traite)-1:  #la liste est classée par latitude
            if coordonnee_proche(lon1, lon2, eps_lon):  #je ne compare longitude que pour deux latitude proches
                non_traite.remove((stop_id2, lat2, lon2))
                stopid_groupeid.append([compt, stop_id2])  #meme id_groupe (=compt) si proches
```

```
            else:
                indice+=1
            stop_id2, lat2, lon2 = non_traite[indice]
        compt+=1
        if compt%1000==0:
            print(compt)
    stopid_groupeid.append([compt, non_traite[0][0]])
    return stopid_groupeid, compt


def coordonnee_proche(lat_long1, lat_long2, eps):
    if abs(lat_long1-lat_long2)<eps:
        return True
    return False

lat_ref, lon_ref = 48.8, 2.3
rayon_reel=100 #distance en m
def choix_eps(distance):
    x_ref, y_ref = WGS84_to_lambert93(lat_ref, lon_ref)
    x, y = x_ref, y_ref
    lat, lon = lat_ref, lon_ref
    while dist(x, y, x_ref, y_ref)<distance:
        lat+=0.00001
        x, y = WGS84_to_lambert93(lat, lon_ref)
    eps_lat=lat-lat_ref

    x, y = x_ref, y_ref
    lat, lon = lat_ref, lon_ref
    while dist(x, y, x_ref, y_ref)<distance:
        lon+=0.00001
        x, y = WGS84_to_lambert93(lat_ref, lon)
    eps_lon=lon-lon_ref
    return eps_lat, eps_lon

#on trouve
eps_lat, eps_lon = choix_eps(rayon_reel)

def regroupement():
    c.execute('delete from groupe')
    conn.commit()
    c.execute('select stop_id, stop_lat, stop_lon from stops order by stop_lat')
    liste = c.fetchall()
    stopid_groupeid, compt = enlever_proche(liste)
    print('il y a', compt, 'id_groupe')
    compt = 0
    for id_groupe, stop_id in stopid_groupeid:
        c.execute('insert into groupe(id_groupe, stop_id) values (?,?)', (id_groupe, stop_id))
        if compt%3000==0:
            print(compt)
        compt+=1
    conn.commit()
    print('done\n')
    remplissage_stops_groupe()


## validité du regroupement
```

```python
#on regarde le nombre de stop_id regroupés et à quelle gare ça correspond
def verification_nombre_par_regroupement(condition):
    c.execute('select count(stop_id) as compt, id_groupe from groupe group by id_groupe having compt>3 order by compt desc limit 5')
    liste=c.fetchall()
    print(liste)
    time.sleep(3)
    if condition:
        for count, id_groupe in liste:
            c.execute('''
            select stop_name
            from stops
            join groupe
                on stops.stop_id = groupe.stop_id
            where groupe.id_groupe={}'''.format(id_groupe))
            print(id_groupe, traitement_tuple(c.fetchall()), '\n')

# on regarde s'il y a des transfers (=correspondance) de moins d'une minute entre deux stop_id non regroupés
def verification_regroupements_transfers():
    c.execute('''select g1.id_groupe, t1.stop_name, g2.id_groupe, t2.stop_name, transfers.transfer_time
    from transfers
    join groupe as g1
        on g1.stop_id=transfers.from_stop_id
    join groupe as g2
        on g2.stop_id=transfers.to_stop_id

    join stops_groupe as t1
        on t1.id_groupe=g1.id_groupe
    join stops_groupe as t2
        on t2.id_groupe=g2.id_groupe

    where g1.id_groupe!=g2.id_groupe and transfers.transfer_time<=60
    group by g1.id_groupe, g2.id_groupe
    order by transfer_time
    limit 50''')
    return c.fetchall()


def validation_regroupement(xmin=660000, xmax=662000, ymin=6870000, ymax=6872000):
    fig_GTFS, axes = pl.subplots(1,2, figsize=[24, 12])
    ax2, ax3 = axes
    ax2.axis('equal'); ax3.axis('equal')
    c.execute('select count(stop_id) from stops')
    N1 = c.fetchone()[0]
    c.execute('select count(id_groupe) from stops_groupe')
    N2 = c.fetchone()[0]
    c.execute('select stop_lat, stop_lon from stops')
    n1=0
    for lat, lon in c:
        x, y = WGS84_to_lambert93(lat, lon)
        if x<xmax and x>xmin and y<ymax and y>ymin:
            ax2.plot(x/1000, y/1000, marker='o')
            n1+=1
    c.execute('''
    select x, y
    from stops_groupe
```

```
    where x<{xmax} and x>{xmin} and y<{ymax} and y>{ymin}'''.format(xmax=xmax, xmin=xmin, ymax=ymax, ymin=ymin))
    n2=0
    for x,y in c:
        n2+=1
        ax2.plot(x/1000, y/1000, marker='+')
        ax3.plot(x/1000, y/1000, marker='o', markersize=8)
        centre=x/1000, y/1000
        creation_cercle(centre, (rayon_reel+10)/1000, ax2)
    grid(ax3)
    ax2.set_title('position de ' + str(n1) + ' pôles avant regroupement dans une zone de 2 km \n (lambert93 en km)\n total: ' + str(N1))
    ax3.set_title('position de ' + str(n2) + ' pôles après regroupement dans cette même zone \n total: ' + str(N2))
    pl.savefig('efficacité du regroupement spatial', dpi=300, bbox_inches='tight')
    pl.show()

def creation_cercle(centre, rayon, ax):
    x, y = centre
    X=[x+rayon*np.cos(teta) for teta in np.linspace(0, 2*np.pi, 30)]
    Y=[y+rayon*np.sin(teta) for teta in np.linspace(0, 2*np.pi, 30)]
    ax.plot(X, Y)

##Stops_groupe
#on utilise les coordonnes de lambert (projection conique) pour pouvoir représenter le réseau
def remplissage_stops_groupe():
    c.execute('delete from stops_groupe')
    c.execute('''
    select stops.stop_lat, stops.stop_lon, groupe.id_groupe, stop_name, sum(frequentation_c)
        from stops
        join groupe
            on stops.stop_id = groupe.stop_id
        group by groupe.id_groupe''')
    for lat, lon, id_groupe, stop_name, freq in c:
        x, y = WGS84_to_lambert93(lat, lon)
        c2.execute('insert into stops_groupe(id_groupe, stop_name, x, y, frequentation_c) values (?,?,?,?,?)', (id_groupe, stop_name, x, y, freq))
    conn.commit()
    print('done \n')

#on se donne un carré n*n que l'on fait grandir jusqu'a trouver une ville correspondant à la gare
def remplissage_code_commune():
    c.execute('select id_groupe, x, y, n_maillage from stops_groupe')
    compt=0
    for id_groupe, x, y, n_maillage in c:
        data=trouver_ville(n_maillage)
        code_commune=plus_proche(data, x, y)
        c2.execute('update stops_groupe set code_commune={c} where id_groupe={i}'.format(c=code_commune, i=id_groupe))
        compt+=1
        if compt%1000==0:
            print(compt)
    conn.commit()


def trouver_ville(n_maillage):
    carre=carre_n_maillage(n_maillage-3*m-3, 7)
    c2.execute('select x, y, code_commune, n_maillage from mobilites_pro where n_maillage in {}'.format(carre))
    data=c2.fetchall()
    if len(data)==0:
        print('non trouvé') #on agrandit suffisement la séléction
```

```
        carre=carre_n_maillage(n_maillage-4*m-4, 9)
        c2.execute('select x, y, code_commune, n_maillage from mobilites_pro where n_maillage in {}'.format(carre))
        data=c2.fetchall()
    return data

def plus_proche(data, x, y):
    d=np.inf
    for x1, y1, code_commune, n_maillage in data:
        d1=dist(x, y, x1, y1)
        if d1<d:
            d=d1
            code=code_commune
    return code

def repr_attribution_ville(xmin=660000, xmax=675000, ymin=6870000, ymax=6885000):
    fig_comm=pl.figure(figsize=[22, 15])
    comm_ax=pl.axes(title='attribution des pôles aux villes adjacentes dans un rayon de 15 km (Lambert 93 en km)')
    comm_ax.axis('equal')
    c.execute('''
    select stops_groupe.x, stops_groupe.y, mobilites_pro.x, mobilites_pro.y, mobilites_pro.name
    from stops_groupe
    join mobilites_pro
        on mobilites_pro.code_commune=stops_groupe.code_commune
    where stops_groupe.x<{xmax} and stops_groupe.x>{xmin} and stops_groupe.y<{ymax} and stops_groupe.y>{ymin}
    '''.format(xmax=xmax, xmin=xmin, ymax=ymax, ymin=ymin))
    for x1, y1, x2, y2, ville_name in c:
        x1k, y1k, x2k, y2k = x1/1000, y1/1000, x2/1000, y2/1000
        comm_ax.plot(x1k, y1k, marker='o')
        comm_ax.plot(x2k, y2k, marker='s', markersize=10, color='red')
        comm_ax.text(x=x2k, y=y2k, s=str(ville_name))
        comm_ax.annotate(s='', xy=(x2k, y2k), xytext=(x1k,y1k),  arrowprops=dict(arrowstyle="->", lw=0.5, mutation_scale=1))
    grid(comm_ax)
    pl.savefig('attribution_ville', dpi=300, bbox_inches='tight')
    pl.show()

##maillage
#pour limiter le coût des calculs de gares proches, on associe à chaque id_groupe un numero correspondant à un cadrillage n*m
#chaque case est de coté 1000m
# n:lignes, m:colonnes --> numerotation par ligne en partant de bas à gauche

def limites():
    global minx, miny, maxx, maxy, n, m
    c.execute('select min(x), min(y), max(x), max(y) from mobilites_pro')
    [(minx1, miny1, maxx1, maxy1)] = c.fetchall()
    c.execute('select min(x), min(y), max(x), max(y) from stops_groupe')
    [(minx2, miny2, maxx2, maxy2)] = c.fetchall()
    minx=min(minx1, minx2); miny=min(miny1, miny2); maxx=max(maxx1, maxx2); maxy=max(maxy1, maxy2)
    n=(maxy-miny)//1000+2; m=(maxx-minx)//1000+2
    n, m = int(n), int(m)
    c.execute('delete from global_data')
    c.execute('insert into global_data (n_ligne, m_colonne, minx, miny, maxx, maxy) values (?,?,?,?,?,?)',(n,m, minx, miny, maxx, maxy ))
    conn.commit()


def maillage():
    limites()
```

```
    compt=0
    print("maillage jusqu'à", n*m)
    for lig in range(n):
        for col in range(m): # case de 0 --> n*m-1
            x=(minx//1000)*1000+col*1000
            y=(miny//1000)*1000+lig*1000
            c.execute('''update stops_groupe
            set n_maillage={num}
            where x between {x0} and {x0}+1000
            and y between {y0} and {y0}+1000
            '''.format(x0=x, y0=y, num=compt))
            c.execute('''update mobilites_pro
            set n_maillage={num}
            where x between {x0} and {x0}+1000
            and y between {y0} and {y0}+1000
            '''.format(x0=x, y0=y, num=compt))
            compt+=1
            if compt%1000==0:
                print(compt)
    conn.commit()


##maj de l'importance
#on prend le carré 3*3 pour etre plus exact
#le centre est donc n_maillage+m+1
def importance(liste, n_maillage):
    c.execute('insert into importance(n_maillage, n_gares) values(?,?)', (n_maillage+m+1, len(liste)))

#on veut aussi remplir les bordures
def completer_importance():
    for col in range(m):
        c.execute('insert into importance(n_maillage, n_gares) values(?,?)', (col, 0))
        c.execute('insert into importance(n_maillage, n_gares) values(?,?)', (col+(n-1)*m, 0))
    for lig in range(1, n-1):
        c.execute('insert into importance(n_maillage, n_gares) values(?,?)', (lig*m, 0))
        c.execute('insert into importance(n_maillage, n_gares) values(?,?)', (lig*m-1, 0))

##graphe du reseau
"il est plus rapide de tout récuperer depuis stop_times et de sélectionner progressivement les infos intéressantes. La requête SQL trop complexe n'aboutissait
pas"

#on retrouve le graphe du réseau à partir de stop_times (=horaires de chaque train)
#on traite par route_id pour supprimer les trip directs
#calcul des distances seulement si il y a une liaison (=voie) ou si les gares sont proches (à pied)
def selection_graphe():
    print('recuperation de stop_times')
    c.execute('''
    select routes.route_type, selection_trip.route_id, groupe.id_groupe, stop_times.stop_sequence
    from stop_times
    join (
            select trips.trip_id, trips.route_id, trips.service_id
            from (
                    select service_id, count(service_id) as nombre
                    from trips
                    group by service_id) as selection_ser
            join trips
```

```
                on trips.service_id=selection_ser.service_id
            where selection_ser.nombre>5) as selection_trip
        on selection_trip.trip_id = stop_times.trip_id
    join groupe
        on groupe.stop_id=stop_times.stop_id
    join calendar
        on calendar.service_id=selection_trip.service_id
    join routes
        on routes.route_id=selection_trip.route_id
    where selection_trip.route_id!="800:TER" and calendar.start_date<20200522 and calendar.stop_date>20200522
    order by selection_trip.route_id, selection_trip.trip_id, stop_times.stop_sequence
    ''')
    print('liste recupere')


# regroupe est ordonne selon cette forme
# liste_dest=[[route_id, dest], [route_id2, dest2]]
# dest=[[id_groupe1, id_groupe2, id_groupe3], [id_groupe7, id_groupe8]]
#c est params car cette fonction est réutilisée dans données_traffic
def creation_liste_dest(c):
    route_id_en_cours="init"
    route_type_en_cours="init"
    dernier_stop_seq="init"
    compt = 0
    liste_dest=[]
    dest=[]
    for route_type, route_id, id_groupe, stop_sequence in c:
        if route_id==route_id_en_cours:
            if stop_sequence!=dernier_stop_seq+1:  #nouveau trip
                dest.append([id_groupe])
                dernier_stop_seq=stop_sequence
            else: #je continue un trip_id
                dest[-1].append(id_groupe)
                dernier_stop_seq+=1
        else: #fin de la ligne (route_id)
            if compt!=0:
                liste_dest.append([route_type_en_cours, route_id_en_cours, dest])
            dest=[[id_groupe]] #on réinitialise et on ajoute le depart
            route_id_en_cours=route_id
            route_type_en_cours=route_type
            dernier_stop_seq=stop_sequence
        compt+=1
        if compt%100000==0:
            print(compt)
    liste_dest.append([route_type_en_cours, route_id_en_cours, dest]) #terminer
    print('liste_dest crée')
    return liste_dest


def remplissage_graphe():
    c.execute('delete from graphe')
    conn.commit()
    selection_graphe()  #je recupere les stop_times
    liste_dest=creation_liste_dest(c)
    liste_dest=prep_suppr_double(liste_dest)
    liste_dest=traitement_directs(liste_dest)
```

```
3008|     print('directs traités', len(liste_dest), 'lignes')
3009|     for route_type, route_id, dest in liste_dest:
3010|         print(route_id)
3011|         for destination in dest:
3012|             insertion_une_destination(destination, route_type, route_id)
3013|     conn.commit()
3014|     suppression_doublons_graphe()
3015|     print('graphe recopie \n')
3016|
3017| def insertion_une_destination(destination, route_type, route_id):
3018|     compt=0
3019|     dernier_id_groupe='init'
3020|     for id_groupe in destination:
3021|         if compt!=0:
3022|             dist=calcul_distance(dernier_id_groupe, id_groupe)
3023|             c.execute('insert into graphe(from_id_groupe, to_id_groupe, route_type, route_id, distance) values (?,?,?,?,?)', (dernier_id_groupe, id_groupe,
route_type, route_id, dist))
3024|         dernier_id_groupe = id_groupe
3025|         compt+=1
3026|
3027| #on veut que le graphe soit symetrique, si les gares sont reliées elles le sont dans les deux sens
3028| #redondance des informations dans la DB mais requetes SQL plus faciles
3029| def retablir_symetrie():
3030|     print('symetrisation')
3031|     c.execute('''
3032|     insert into graphe (from_id_groupe, to_id_groupe, route_type, route_id, distance)
3033|         select graphe.to_id_groupe, graphe.from_id_groupe, route_type, route_id, distance from graphe
3034|         join (select from_id_groupe, to_id_groupe from graphe
3035|                 except
3036|             select to_id_groupe, from_id_groupe from graphe) as selection
3037|             on selection.from_id_groupe=graphe.from_id_groupe and selection.to_id_groupe=graphe.to_id_groupe
3038|             ''')
3039|     conn.commit()
3040|
3041| def prep_suppr_double(liste_dest):
3042|     liste_dest_corrige=[]
3043|     for route_type, route_id, dest in liste_dest:
3044|         liste_dest_corrige.append([route_type, route_id, suppr_double(dest)])
3045|     return liste_dest_corrige
3046|
3047| def suppr_double(dest):
3048|     new_dest=[]
3049|     for destination in dest:
3050|         if destination not in new_dest:
3051|             new_dest.append(destination)
3052|     return new_dest
3053|
3054| ##traitement directs
3055| '''je ne veut pas dans le graphe de liaisons virtuelles créées par un train direct
3056| je regarde pour chaque liaison si il n'y en a pas une plus longue'''
3057| #j'applique correction_directs_ligne à toutes les lignes
3058| #parfois un direct est emboité dans un plus direct encore et il faut plusieurs itérations de correction_directs_ligne
3059| def traitement_directs(liste_dest):
3060|     liste_dest_corrige=[]
3061|     for route_type, route_id, dest in liste_dest:
3062|         nombre_chgmts=1
```

```
3063|         if route_type!=3: #les bus n'ont pas ce problème
3064|             while nombre_chgmts!=0:
3065|                 dest, nombre_chgmts = correction_directs_ligne(dest, False)
3066|         liste_dest_corrige.append([route_type, route_id, dest])
3067|     return liste_dest_corrige
3068|
3069| #remplace dans dest (pour un route_id donc) les trajets directs par les omnibus
3070| #ce programme est utilisé dans donnees traffic pour recuperer directs, dans ce cas cond_creation_directs=True et on peut renseigner directs
3071| def correction_directs_ligne(dest, cond_creation_directs, directs=[]):
3072|     nombre_chgmts=0
3073|     N=len(dest)
3074|     for numdest in range(N):
3075|         feuille_route=dest[numdest]
3076|         for k in range(len(feuille_route)-2, -1, -1):
3077|             gare1=feuille_route[k]; gare2=feuille_route[k+1]
3078|             liste_recherche=dest[:numdest]+dest[numdest+1:]
3079|             intermediaires=recherche_trajet_equivalent(gare1, gare2, liste_recherche)
3080|             if verification_boucle(intermediaires, feuille_route):
3081|                 feuille_route=feuille_route[:k+1]+intermediaires+feuille_route[k+1:]  #compteur descendant
3082|                 if cond_creation_directs:
3083|                     directs[numdest].extend(intermediaires)
3084|                 nombre_chgmts+=len(intermediaires)
3085|         dest[numdest] = feuille_route
3086|     if cond_creation_directs:
3087|         return dest, nombre_chgmts, directs
3088|     else:
3089|         return dest, nombre_chgmts
3090|
3091| #il ne faut pas créer de boucle --> infini sinon
3092| #cette situation arrive sur les lignes ou il y a des boucles (RER C par ex)
3093| def verification_boucle(intermediaires, destination):
3094|     for id_groupe in intermediaires:
3095|         if id_groupe in destination:
3096|             return False
3097|     return True
3098|
3099| #je recherche s'il y a un train non-direct entre gare1, gare2
3100| def recherche_trajet_equivalent(gare1, gare2, liste_dest):
3101|     parties_changer=[]
3102|     for k in range(len(liste_dest)):
3103|         destination = liste_dest[k]
3104|         resultats=[-1, -1]
3105|         for index in range(len(destination)):
3106|             if destination[index]==gare1:
3107|                 resultats[0]=index
3108|             elif destination[index]==gare2:
3109|                 resultats[1]=index
3110|         if resultats[0]!=-1 and resultats[1]!=-1 and abs(resultats[0]-resultats[1])>1: #il existe un trajet moins direct
3111|             # print(gare1, gare2, k)
3112|             parties_changer = remetre_ordre(destination, resultats, parties_changer)
3113|     if len(parties_changer)==0:
3114|         return []
3115|     else:
3116|         plus_long = selection_plus_long(gare1, gare2, parties_changer)
3117|     return plus_long
3118|
```

```
#on veut le plus long (=le moins direct) et pas de boucle
def selection_plus_long(gare1, gare2, parties_changer):
    plus_long=[]
    for liste in parties_changer:
        if len(liste)>len(plus_long) and gare1 not in liste and gare2 not in liste:
            plus_long = liste
    return plus_long

#le trajet intermediaire trouvé peut etre de gare2-->gare1 et pas dans le bon sens
def remetre_ordre(destination, resultats, parties_changer):
    if resultats[0]<resultats[1]: #bon ordre
        parties_changer.append(destination[min(resultats)+1:max(resultats)])
    else:
        parties_changer.append(reverse(destination[min(resultats)+1:max(resultats)]))  #ordre inverse
    return parties_changer

#après remplacement des directs par omnibus on a des redondances
def suppression_doublons_graphe():
    print('suppression doublons')
    c.execute('''
    delete from graphe
    where cle not in (
    select  cle
        from graphe
        where from_id_groupe!=to_id_groupe
        group by from_id_groupe, to_id_groupe, route_type, route_id )''')
    conn.commit()
    print('sans doublons graphe \n')

##gares accessibles à pied
'''route_id=-1 si à moins de 500m
route_id=-2 si à moins de 1km
le maillage permet de reduire le cout n*n en séquencant l'espace
on en profite pour calculer importance'''
def marche():
    print('ligne', n, 'colonne', m, 'total', n*m)
    for lig in range(n-2):
        for col in range(m-2):
            n_maillage=lig*m + col
            carre=carre_n_maillage(n_maillage, 3)
            c.execute('select id_groupe, x, y from stops_groupe where n_maillage in {}'.format(carre))
            liste=c.fetchall()
            traitement_graphe_marche(liste)
            importance(liste, n_maillage)
            if n_maillage%1000==0:
                print(n_maillage)
    suppression_doublons_graphe()  #plus efficace de le faire une fois à la fin
    completer_importance()
    conn.commit()
    print('marche rempli\n')

#on renseigne en bas à gauche et renvoie le carrée de k*k
def carre_n_maillage(n_maillage, k):
    rep=()
    for ligne in range(k):
        for col in range(k):
```

```
3175|             rep+=(n_maillage+ligne*m+col,)
3176|     return rep
3177|
3178| #il faudrait ne remplir qu'un demi et recopier pour gagner du temps
3179| def traitement_graphe_marche(liste):
3180|     for id_groupe1, x1, y1 in liste:
3181|         for id_groupe2, x2, y2 in liste:
3182|             execute_marche(id_groupe1, x1, y1, id_groupe2, x2, y2)
3183|
3184| def execute_marche(id_groupe1, x1, y1, id_groupe2, x2, y2):
3185|     deltx=abs(x1-x2); delty=abs(x1-x2)
3186|     distance=np.sqrt(deltx**2+delty**2)
3187|     if distance<rayon_reel*3:
3188|         if distance<rayon_reel+10:
3189|             route_type=-1
3190|         else:
3191|             route_type=-2
3192|         c.execute('insert into graphe (from_id_groupe, to_id_groupe, route_type, route_id, distance) values (?,?,?,?,?)', (id_groupe1, id_groupe2, route_type,
"marche", distance))
3193|
3194|
3195| ##représentation graphe
3196| def grid(ax):
3197|     xmin, xmax, ymin, ymax = ax.axis()
3198|     x_ticks=np.arange(int(xmin), xmax+1, 1); y_ticks=np.arange(int(ymin), ymax+1, 1)
3199|     ax.set_xticks(x_ticks, minor=True); ax.set_yticks(y_ticks, minor=True)
3200|     ax.grid(which='both', alpha=0.2)
3201|
3202| titles=['tramway', 'métro', 'RER', 'bus']
3203| def representation(route_type, condition_global):  #on choisis de tracer ligne par ligne ou tout d'un coup
3204|     fig_maill=pl.figure(figsize=[23, 16])
3205|     maill_ax=pl.axes()
3206|     maill_ax.axis('equal')
3207|     c.execute('select count(route_id) from routes where route_type={}'.format(route_type))
3208|     N=c.fetchone()[0]
3209|     title = str(N) + ' lignes de ' + titles[route_type] + ' positionnés en km dans le système de coordonnées Lambert'
3210|     maill_ax.set_title(title)
3211|     c.execute('''
3212|     select from_id_groupe, to_id_groupe, graphe.route_type, graphe.route_id, routes.route_long_name, s1.x, s1.y, s2.x, s2.y
3213|     from graphe
3214|     join routes
3215|         on routes.route_id=graphe.route_id
3216|     join stops_groupe as s1
3217|         on s1.id_groupe=from_id_groupe
3218|     join stops_groupe as s2
3219|         on s2.id_groupe=to_id_groupe
3220|     where graphe.route_type={} and from_id_groupe<to_id_groupe
3221|     order by graphe.route_id, s1.x, s1.y'''.format(route_type))
3222|     dernier_route_id='init'
3223|     compt=0
3224|     for id1, id2, route_type, route_id, route_long_name, x1, y1, x2, y2 in c:
3225|         if route_id!=dernier_route_id:
3226|             compt+=1
3227|             if not condition_global:
3228|                 pl.show()
3229|             if route_type!=3:
```

```
                color=random_color()
                maill_ax.plot([x1/1000, x2/1000], [y1/1000, y2/1000], color=color, linewidth=3, label=route_long_name)
                print(dernier_route_id)
            else:
                color=[0, 0, 1]
                if compt%20==0:
                    print(compt)
        else:
            maill_ax.plot([x1/1000, x2/1000], [y1/1000, y2/1000], color=color, linewidth=3)
        dernier_route_id = route_id
    if route_type!=3:
        maill_ax.legend(loc='lower right', title='nom des lignes')
    grid(maill_ax)
    print(dernier_route_id)
    pl.savefig('maillage ' + titles[route_type], dpi=400,  bbox_inches='tight')
    pl.show()

def representation_une_ligne(route_id, ax):
    c.execute('''select s1.x, s1.y, s2.x, s2.y, s1.stop_name, s1.id_groupe
    from graphe
    join stops_groupe as s1
        on s1.id_groupe=graphe.from_id_groupe
    join stops_groupe as s2
        on s2.id_groupe=graphe.to_id_groupe
    where graphe.route_id="{}" and s1.id_groupe>s2.id_groupe
    group by s1.stop_name, s2.stop_name
    '''.format(route_id))
    ax.axis('equal')
    for x1, y1, x2, y2, name, id_groupe in c:
        ax.plot([x1/1000, x2/1000], [y1/1000, y2/1000], color='black')
        # ax.annotate(name+' '+str(id_groupe), xy=(x1, y1), size=6)

## outils de debug
def liaisons(route_id):
    c.execute('''
    select s1.stop_name, s1.id_groupe, s2.stop_name, s2.id_groupe
    from graphe
    join stops_groupe as s1
        on s1.id_groupe=graphe.from_id_groupe
    join stops_groupe as s2
        on s2.id_groupe=graphe.to_id_groupe
    where graphe.route_id="{}"
    group by s1.stop_name, s2.stop_name
    order by s1.stop_name'''.format(route_id))
    return c

def trip_id_passant_par(id_groupe):
    c.execute('''
    select trips.route_id, trips.trip_id, trips.service_id
    from stop_times
    join trips
        on trips.trip_id=stop_times.trip_id
    where stop_times.stop_id
    in (select stops.stop_id
            from stops
            join groupe
```

```
3286|                 on groupe.stop_id=stops.stop_id
3287|             where groupe.id_groupe="{}")
3288|     group by trips.route_id''''.format(id_groupe))
3289|     return c
3290|
3291|
3292| def representation_maillage():
3293|     c.execute('select stop_lat, stop_lon from stops')
3294|     for lat, lon in c:
3295|         x,y = WGS84_to_lambert93(lat, lon)
3296|         pl.plot(x, y, marker='o')
3297|     pl.show()
3298|     c.execute('select x,y from stops_groupe')
3299|     for x,y in c:
3300|         pl.plot(x, y, marker='o')
3301|     pl.show()
3302|
3303| ##global
3304| def reseau():
3305|     print('remplissage de nb_trips\n')
3306|     remplir_nb_trips()
3307|     print('conversion Lambert93 mobilites\n')
3308|     conversion_mobilites()
3309|     print('suppression des exceptions\n')
3310|     exceptions()
3311|     print('regroupement\n')
3312|     regroupement()
3313|     maillage()
3314|     print('remplissage_graphe\n')
3315|     remplissage_graphe()
3316|     print('symétrisation du graphe\n')
3317|     retablir_symetrie()
3318|     print('attribution des villes\n')
3319|     remplissage_code_commune()
3320|     print('frequentations modèlisés\n')
3321|     remplir_flux_emis()
3322|     print('liaisons marche\n')
3323|     marche()
3324|     for route_type in range(3, -1, -1):
3325|         representation(route_type, True)
3326|     #il faut aussi actualiser les flux du RERB pour le modèle
3327|
3328|
3329|
```