```
# TIPE_traffic.py

0001| import numpy as np
0002| import matplotlib.pyplot as pl
0003| import random as rd
0004| import matplotlib.animation as animation
0005| import sqlite3 as sql
0006| from Donnees_traffic import variables,
correspondance_DB_traffic, correspondance_traffic_DB
0007| from matplotlib.widgets import Button
0008| from algorithmes_de_minimisation import minimum,
monteCarlo
0009| from time import time
0010| import copy
0011|
0012|
0013| route_id="810:B"
0014| ## Creation et initialisation du maillage
0015| """
0016| dt en minutes
0017|
0018| points=[[x1, y1], [x2, y2], ... ]
0019| distances= matrice n*n diagonale ij = distances entre i et
j
0020| si 0 alors non reliées
0021|
0022| gares=[[gare1], [gare2], ... ]
0023| gare=[occupations_des_voies, fréquentations]
0024| occupations_des_voies=[[direction(vers où), nombre de
voies occupées, nombre de voies total], [dir2, n2, nprim2]...]
0025| fréquentations=[direction, passagers_en_gare,
passagers_quotidiens, dernière heure de desserte]
0026|
0027| dest=[[feuille de route1], [feuille2], ...]
0028| feuille de route=[gares par lesquelles passer]
0029| direct= meme format que dest -> gares ou le train est sans
arret
0030|
0031| types=[[typ1], [typ2], ...]
0032|
0033| trains contient des typ
0034| typ=[position,vitesse_actuelle, acceleration,
tps_attendu_arret, gar_départ, gar_arrivée, remplissage,
no_feuille_route = no_feuille_direct]
0035|     si gar_depart=-1 alors le train est arrivé en
gar_arrivé
0036|     la pos est en proportion de la distance, la vit est
absolue
0037|     il y en a un par feuille de route
0038|
```

```
0039| remplissage=[[gare de destination, nombre de personnes],
[gare2, n2], ...]
0040|
0041|
0042| destinations contient des dest
0043| dest=[gare1, gare2...]
0044|
0045| sans_arret contient des direct
0046| direct=[gare_évitée1, gare_évitée2 ...]
0047|
0048| Les deux fonctionnent en paire
0049|
0050| arret contient des numtrains dont l'arret est contraint
par un autre train
0051| arret_force contient des numtrains dont l'arret est
contraint par une perturbation (dans le but d'en évaluer
l'impact)
0052|
0053| fin_de_service contient des  [numtrain,numgare] allant
être supprimés en numgare
0054|
0055| horaires=[[type de train, horaire de départ, (trip_id1)],
[typ2, horaire2, (trip_id2)], ...]
0056|
0057| Attention, stop_times et stop_times0 ne son pas au même
format
0058|
0059| stop_times=[[numtrain, 'trip_id', passages(liste)],
[numtrain2, 'trip_id2', passages(liste)], ... ]
0060| passages=[numgare, heure, voy_montants, minutes_attendues]
0061| stop_times_stock est similaire à stop_times mais avec des
id_groupe à la place des numgare
0062|
0063| stop_times0=[trip1, trip2 ...]
0064| trip=[trip_id, dep_times]
0065| dep_times=[h1, h2 ...]
0066|
0067| profil_suivi=[intergare1, intergare2, ...]
0068| intergare=[gare_dep, gare_ar, valeurs1, valeurs2, ...]
0069| valeur=[pos, heure, vit, accel, remplissage] avec
remplissage correspondant au contenu du train numtrain_course
0070|
0071| """
0072|
0073| def variables_neutres():
0074|     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
```

```
0075|     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
0076|
0077| #il faut copier le contenu de chaque sous_liste de gares
0078| #pas besoin pour horaires qui n'est pas modifié (seulement
suppr)
0079| def etat_initial():
0080|     variables_neutres()
0081|     global horaires, gares, zeros
0082|     gares = copy.deepcopy(gares0)
0083|     horaires = horaires0.copy()
0084|     zeros=[0]*len(gares)
0085|
0086| ## donnees GTFS
0087| #on peut choisir de lancer le modèle avec des paramètres
différents
0088| def donnees_GTFS(route_id = route_id):
0089|     global dimension, annote
0090|     dimension = 600
0091|     annote = False
0092|     global points, liste_id, distances, dest, types,
horaires0, gares0, direct, num_max_trains, cadre, stop_times0,
dest_full, plages
0093|     # try:
0094|     points, liste_id, distances, dest, types, horaires0,
gares0, direct, num_max_trains, cadre, stop_times0, dest_full,
plages = variables(route_id)
0095|     # except Exception:
0096|         # print('erreur de donnée')
0097|
0098|
0099| ##global
0100| # les données sont sauvegardees au débur pour ne pas les
recalculer
0101| def demarrage():
0102|     etat_initial()
0103|     global affichage, dessin, service
0104|     affichage=True; dessin=True; service=True
0105|     evolution_traffic(6, 0.1, 8, 20)
0106| '''
0107| num_h=4  départ à 6.0
trip_id_course='115072256-1_14393'; duree=0.2; tm=6.2; g1=34;
g2=37
0108| num_h=51  départ à 7.38
trip_id_course='115054849-1_16156'; duree=0.2; tm=7.5; g1=30;
```

```
g2=29
''' 
def profil_course():
    etat_initial()
    global profil, trip_id_course, dessin, service,
exemple
    profil = True; dessin=True; service=False;
exemple=False
    trip_id_course='115054849-1_16156'
    evolution_traffic(5, 0.1, 23, 200)
    comparaison(trip_id_course)


def evaluation_pert():
    etat_initial()
    global pert_on, dessin, service, profil,
trip_id_course, consigne_pert
    pert_on=True; dessin=False; service=False;
profil=True; consigne_pert=False
    trip_id_course='115054849-1_16156'; duree=0.2; tm=7.5;
g1=30; g2=29
    variables_de_pert(duree, tm, g1, g2)
    evolution_traffic(5.0, 0.1, 23, 20)
    comparaison(trip_id_course)
    # reecriture_DB()
    # ecart_db(True)


#fait une itération du modèle pour les paramètres données,
il s'agit de la fonction à minimiser
#on ne calcule qu'une seule course
def retour_ecart(variabs):
    etat_initial()
    global vitesse_nominale, vitesse_arrivee, contenance,
tps_arret, tps_population
    [vitesse_nominale, vitesse_arrivee, contenance,
tps_arret, tps_population]=variabs
    global affichage, horaires, service, dessin
    affichage=False; service=False; dessin=False
    horaires=[horaires0[51]] #train quelconque
'115054849-1_16156'
    evolution_traffic(7.2, 0.1, 9, 100)
    ec = ponctualite()
    return ec

def evolution_traffic(t0, dt1, t_fin, time_par_frame):
#dt en minutes, t_fin et t_depart en heures
    global t_depart, dt
    delta_t = t_fin-t0
    t_depart = t0
```

```
0148|     dt = dt1
0149|     n = int(delta_t//(dt/60))
0150|     if affichage:
0151|         print(n, 'frames')
0152|     initialisation_horaires(t0)
0153|     if dessin:
0154|         creation_artists(num_max_trains, cadre[0],
cadre[1])
0155|         global ani
0156|         ani = animation.FuncAnimation(fig1, anim,
fargs=(5, ), init_func = init, frames = n, blit = False,
interval = time_par_frame, repeat = False)
0157|         pl.show()
0158|     else:
0159|         for i in range(n):
0160|             if condition_arret_prgm:
0161|                 break
0162|             heure = t_depart+i*dt/60
0163|             temps_suivant(heure)
0164|             if i%50==0 and affichage:
0165|                 print('temps:', conversion_h_m(heure))
0166|     stop_times_stock.extend(stop_times)
0167|     statistiques_attente()
0168|
0169| ##Avancement d'une durée dt
0170| def temps_suivant(heure):
0171|     nb_trains = len(trains)
0172|     mise_en_service(heure)
0173|     for numtrain in range(nb_trains):
0174|         if est_en_gare(numtrain):
0175|             train_gare(numtrain, heure)
0176|         else:
0177|             train_voie(numtrain, heure)
0178|     execute_fin_service()
0179|     gestion_arret_force(heure)
0180|     if profil:
0181|         gestion_profil(heure)
0182|     if pert_on:
0183|         mise_en_place_pert(heure)
0184|
0185|
0186| ##gares
0187| def est_en_gare(numtrain):
0188|     if trains[numtrain][4]==-1:
0189|         return True
0190|     else:
0191|         return False
0192|
0193| def train_gare(numtrain, heure):
0194|     gare1 = trains[numtrain][5]
```

```
0195|     if len(destinations[numtrain])<=1:  #fin de service à
la fin (modif des numtrains)
0196|         fin_de_service.append([numtrain, gare1])
0197|     else:
0198|         gare2 = destinations[numtrain][0]
0199|         ind = recherche_indice_quai(gare1,gare2)
0200|         if ind is None:
0201|             print('suppression du train', numtrain)
0202|             erreurs.append([numtrain, 'destinations
incohérentes de', gare1, 'à', gare2])
0203|             fin_de_service.append([numtrain, gare1])
0204|         else:
0205|             personnes = voyageurs_quai(gare1, numtrain)
0206|             if condition_redemarrage_gare(numtrain,
personnes, gare1, gare2):
0207|                 execute_train_redemarrage(numtrain, gare1,
gare2, personnes, ind, heure)
0208|             else:
0209|                 execute_train_attente(numtrain)
0210|
0211| #condition de tps d'attente: 30 sec +1 min toute les 1000
pers en gare
0212| #condition voie libre: pas de trains dans la premiere
moitié
0213| def condition_redemarrage_gare(numtrain, personnes, gare1,
gare2):
0214|     pos_liaison = distance_train_proche(gare1, gare2,
numtrain)
0215|     if len(pos_liaison)>0:
0216|         pos_proche = min(pos_liaison)
0217|     else:
0218|         pos_proche = 1 #pas de train
0219|     if pos_proche>0.6 and trains[numtrain]
[3]>tps_arret+personnes*tps_population and (numtrain not in
arret_force):
0220|         return True
0221|     else:
0222|         return False
0223|
0224|
0225| #on récupère les positions de tous les trains sur la
liaison excepté le numtrain considéré (utile pour un train en
voie)
0226| def distance_train_proche(gare1, gare2, numtrain1):
0227|     pos_proche=[]
0228|     for numtrain in range(len(trains)):
0229|         if trains[numtrain][4]==gare1 and trains[numtrain]
[5]==gare2 and numtrain!=numtrain1:
0230|             pos = trains[numtrain][0]
0231|             pos_proche.append(pos)
```

```
0232|     return pos_proche
0233|
0234|
0235| def recherche_indice_quai(gare1,gare2):
0236|     occup=gares[gare1][0]
0237|     for ind in range(len(occup)):
0238|         if occup[ind][0]==gare2:
0239|             return ind
0240|     print('erreur, ce train souhaite aller de', gare1,
'à', gare2)
0241|
0242| #on ne fait monter et descendre les voyageurs qu'au
redémarrage car ces derniers interviennent dans les calculs de
temps d'attente en gare
0243| def execute_train_redemarrage(numtrain, gare1, gare2,
personnes, ind, heure):
0244|     trains[numtrain][4:6]=[gare1,gare2]
0245|     trains[numtrain][1]=0     #sans vitesse initiale
0246|     trains[numtrain][3]=heure
#reset temps d'attente
0247|     gares[gare1][0][ind][1]-=1  #libère un quai
0248|     monter_voyageurs(numtrain,gare1, heure)
0249|
0250| def execute_train_attente(numtrain):
0251|     trains[numtrain][3]+=dt
0252|
0253| ##voies
0254| def train_voie(numtrain, heure):
0255|     [pos,vit] = trains[numtrain][0:2]
0256|     gare1, gare2 = int(trains[numtrain][4]),
int(trains[numtrain][5])
0257|     dist = distances[gare1, gare2]
0258|     gare3 = destinations[numtrain][1]
0259|     ind = recherche_indice_quai(gare2,gare3)
0260|     if ind is None or gare1==gare2:
0261|         print('suppr')
0262|         erreurs.append([numtrain, 'destinations
incohérentes de', gare2, 'à', gare3])
0263|     else:
0264|         test_securite(numtrain, pos, gare1, gare2, gare3,
ind, dist)
0265|         #traite tous les cas d'arret, de ralentissement et
d'arret forcé par la pert
0266|         vit = trains[numtrain][1]
0267|         accel = trains[numtrain][2]
0268|         if (numtrain not in arret) and (numtrain not in
ralentissement) and (numtrain not in arret_force):
0269|             if arrive_en_gare(pos, vit, dist):
0270|                 if gare2 in sans_arret[numtrain]:
0271|
```

```
execute_train_direct_gare(numtrain,gare2,gare3)
0272|                 else:
0273|                     execute_train_arrive_gare(numtrain,
gare2, ind, heure, gare3)
0274|             else:
0275|                 execute_train_avancer(numtrain, pos, vit,
accel, dist)
0276|
0277|
0278| def execute_train_arrive_gare(numtrain, gare2, ind, heure,
gare3):
0279|     augmenter_voyageurs_gares(gare2, numtrain, heure)
0280|     descendre_voyageurs(numtrain,gare2)
0281|     trains[numtrain][0:3]=[0,0,0]    #arret
0282|     trains[numtrain][3]=0  #tps attente
0283|     trains[numtrain][4]=-1
0284|     trains[numtrain][5]=gare2
0285|     remplissage = trains[numtrain][6]
0286|     gare3 = int(destinations[numtrain][1])
0287|     gares[gare2][0][ind][1]+=1 #occupe un quai
0288|     destinations[numtrain].remove(gare2)
0289|     suivi_ajout_passage(numtrain, gare2, heure)
0290|
0291| def execute_train_direct_gare(numtrain,gare2,gare3):
0292|     trains[numtrain][4:6]=[gare2,gare3]
0293|     trains[numtrain][0]=0
0294|     destinations[numtrain].remove(gare2)
0295|     sans_arret[numtrain].remove(gare2) #pas utile mais
plus clair
0296|
0297| #la régulation de la dynamique du train se fait sur le
couple qui est proportionnel à l'accélértion
0298| #couple de démarrage constant-> montée linéaire de couple
jusqu'à vitesse_nominale ou couple_nominal -> couple nul->
couple négatif de freinage
0299| def execute_train_avancer(numtrain, pos, vit, accel,
dist):
0300|     distance=pos*dist
0301|     if distance<distance_demarrage:
0302|         accel1=accel_nominale
0303|         vit1=vit+accel1*dt
0304|         if vit1>vitesse_nominale:
0305|             vit1=vitesse_nominale
0306|     elif (dist-distance)<distance_freinage: #proche de
l'arrivée
0307|         vit1=vit+freinage_nominal*dt
0308|         if vit1>vitesse_arrivee: #on ne veut pas qu'il
s'arrete mais seulement qu'il ralentisse
0309|             accel1=freinage_nominal
0310|         else:
```

```
0311|             vit1=vitesse_arrivee
0312|             accel1=0
0313|     else: #milieu de course
0314|         accel1=0
0315|         vit1=vitesse_nominale
0316|     avance=vit1*dt
0317|     pos1=pos+avance/dist
0318|     trains[numtrain][0:3]=[pos1, vit1, accel1]
0319|
0320| ##sécurité
0321| #deux trains proches
0322| def test_securite_train(numtrain, pos, gare1, gare2,
dist):
0323|     liste_pos = distance_train_proche(gare1, gare2,
numtrain)
0324|     arret_demande = False; ralentissement_demande = False
0325|     for pos1 in liste_pos:
0326|         if abs((pos-pos1)*dist)<distance_securite and
pos<pos1:
0327|             ralentissement_demande=True
0328|         if abs((pos-pos1)*dist)<distance_securite_mini and
pos<pos1:
0329|             arret_demande = True
0330|     return ralentissement_demande, arret_demande
0331|
0332| #voie non dispo
0333| def test_securite_gare(numtrain, pos, gare2, gare3, ind,
dist):
0334|     arret_demande = False; ralentissement_demande = False
0335|     if not voie_disponible_gare(gare2,gare3, ind):
0336|         if (1-pos)*dist<distance_securite:
0337|             ralentissement_demande = True
0338|         if (1-pos)*dist<distance_securite_mini:
0339|             arret_demande = True
0340|     return ralentissement_demande, arret_demande
0341|
0342| #on compile les raisons de s'arreter
0343| def test_securite(numtrain, pos, gare1, gare2, gare3, ind,
dist):
0344|     r1, a1 = test_securite_train(numtrain, pos, gare1,
gare2, dist)
0345|     r2, a2 = test_securite_gare(numtrain, pos, gare2,
gare3, ind, dist)
0346|     arret_du_train((a1 or a2), numtrain)
0347|     ralentissement_du_train((r1 or r2), numtrain, dist)
0348|
0349| def arret_du_train(condition, numtrain):
0350|     #arret force correspond à une perturbation qui impose
un arret qui n'est pas lié aux conditions de sécurité énoncées
ci-dessus
```

```
0351|     #cet arret prime sur les autres
0352|     if arret_force_de(numtrain):
0353|         execute_train_attente(numtrain) #le train est déjà
arreté
0354|     else:
0355|         if condition:
0356|             if numtrain in arret:
0357|                 execute_train_attente(numtrain)
0358|             else:
0359|                 execute_arret(numtrain)
0360|         else:
0361|             if numtrain in arret:
0362|                 arret.remove(numtrain)
0363|
0364| def ralentissement_du_train(condition, numtrain, dist):
0365|     if condition:
0366|         if numtrain not in ralentissement:
0367|             execute_ralentissement(numtrain, dist)
0368|     else:
0369|         if numtrain in ralentissement:
0370|             ralentissement.remove(numtrain)
0371|
0372| def arrive_en_gare(pos, vit, dist):
0373|     if pos+vit*dt/dist>1:
0374|         return True
0375|     else:
0376|         return False
0377|
0378| def voie_disponible_gare(gare2,gare3, ind):
0379|     if gares[gare2][0][ind][1]<gares[gare2][0][ind][2]:
0380|         return True
0381|     else:
0382|         return False
0383|
0384| def execute_arret(numtrain):
0385|     arret.append(numtrain)
0386|     trains[numtrain][1:3]=[0,0]   #arret
0387|     print('train ', numtrain, 'arreté')
0388|
0389| def execute_ralentissement(numtrain, dist):
0390|     ralentissement.append(numtrain)
0391|     pos, vit = trains[numtrain][0:2]
0392|     trains[numtrain][2]=freinage_nominal
0393|     if vit>vitesse_arrivee:
0394|         vit1=vit+freinage_nominal*dt
0395|         print('train', numtrain, 'ralenti')
0396|     else:
0397|         vit1=vitesse_arrivee
0398|     pos1=pos+vit1*dt/dist
0399|     trains[numtrain][0:2]=[pos1, vit1]
```

```
0400|
0401|
0402| ##mise en service
0403|
0404| #permet de commencer à n'importe quelle heure en
élimninant les trains passés
0405| def initialisation_horaires(heure):
0406|     global horaires
0407|     num_horaire=0
0408|     temps_deb=horaires[0][1]
0409|     while temps_deb<=heure and len(horaires)>1:
0410|         num_horaire+=1
0411|         temps_deb=horaires[num_horaire][1]
0412|     horaires=horaires[num_horaire:]
0413|     if affichage:
0414|         print(num_horaire, "horaires en dehors de la
zone")
0415|     for gare in gares:
0416|         for direction in gare[1]:
0417|             direction[3]=heure
0418|
0419| def mise_en_service(heure):
0420|     if len(horaires)>0:
0421|         num_horaire=0
0422|         temps_deb=horaires[num_horaire][1]
0423|         while temps_deb<=heure and
len(horaires)>num_horaire+1:
0424|             temps_deb=horaires[num_horaire+1][1]
0425|             num_horaire+=1
0426|         #num_horaire a atteint un train en dehors de la
zone des horaires alors num_horaire ne doit pas être mis en
place
0427|         if temps_deb>heure:
0428|             j=1
0429|         else: #num_horaire a atteint la fin de la liste
0430|             j=0
0431|         for k in range(num_horaire-j, -1, -1):
0432|             execute_mise_service(k, heure)
0433|
0434|
0435| def execute_mise_service(num_horaire, heure):
0436|     typ = horaires[num_horaire][0]
0437|     trip_id = horaires[num_horaire][2]
0438|     train = copy.deepcopy(types[typ])
0439|     no_feuille_route = train[7]
0440|     desti = dest[no_feuille_route].copy()
0441|     destinations.append(desti)
0442|     s_a=direct[no_feuille_route].copy()
0443|     sans_arret.append(s_a)
0444|     numtrain = len(trains)
```

```
0445|     trains.append(train)
0446|     gare1 = train[5]
0447|     gare2 = desti[0]
0448|     ind = recherche_indice_quai(gare1,gare2)
0449|     gares[gare1][0][ind][1]+=1 #un train de plus en gare
0450|     tps_litteral = conversion_h_m(heure)
0451|     augmenter_voyageurs_gares(gare1, numtrain, heure)
0452|     if service: #affichage
0453|         nom_gare = correspondance_traffic_DB([gare1],
liste_id)[0][1]
0454|         print('train', trip_id, 'mis en service à',
nom_gare)
0455|     stop_times.append([numtrain, trip_id, [[gare1, heure,
0, 0]]])
0456|     if profil and trip_id==trip_id_course:
0457|         global numtrain_course
0458|         numtrain_course = numtrain
0459|         print('numtrain_course:', numtrain_course, heure)
0460|         gestion_profil(heure)
0461|     horaires.remove(horaires[num_horaire])  #intérêt du
compteur descendant de mise_en_service
0462|
0463| ##fin de service
0464| #il faut un compteur descendant
0465| def execute_fin_service():
0466|     global fin_de_service
0467|     for k in range(len(fin_de_service)-1, -1, -1):
0468|         [numtrain, numgare] = fin_de_service[k]
0469|         execute_suppr_suivi(numtrain, numgare)
0470|         execute_changement_indice_trains(numtrain)
0471|     fin_de_service=[]
0472|
0473|
0474| def execute_suppr_suivi(numtrain, numgare):
0475|     global trains, destinations, stop_times,
numtrain_course, sans_arret
0476|     index = recherche_indice_suivi(numtrain)
0477|     stop_times_stock.append(stop_times[index])
0478|     trip_id=stop_times[index][1]
0479|     stop_times = stop_times[:index]+stop_times[index+1:]
#on enlève ce train des trains suivis
0480|     if service:
0481|         print('il
restait',int(total_voyageurs_dans_train(numtrain)),'voyageurs',
'fin de service de', trip_id, numtrain)
0482|     if profil and numtrain_course==numtrain:
0483|         global condition_arret_prgm
0484|         condition_arret_prgm = True  #il ne sert à rien de
continuer
0485|         numtrain_course= -1
```

```
0486|     trains = trains[:numtrain]+trains[numtrain+1:]
0487|     sans_arret = sans_arret[:numtrain]
+sans_arret[numtrain+1:]
0488|     destinations = destinations[:numtrain]
+destinations[numtrain+1:]
0489|     ind = recherche_indice_quai(numgare, -1)
0490|     gares[numgare][0][ind][1]-=1 #le train libère le quai
0491|
0492| #il faut faire correspondre les numtrains suivis/arretés/
course avec les nouveaux numtrains
0493| def execute_changement_indice_trains(numtrain):
0494|     n = len(arret)
0495|     for k in range(n):
0496|         if arret[k]>numtrain:
0497|             arret[k]-=1
0498|
0499|     n=len(ralentissement)
0500|     for k in range(n):
0501|         if ralentissement[k]>numtrain:
0502|             ralentissement[k]-=1
0503|
0504|     n = len(stop_times)
0505|     for i in range(n):
0506|         if stop_times[i][0]>numtrain:
0507|             stop_times[i][0]-=1
0508|
0509|     global numtrain_course
0510|     if numtrain_course>numtrain:
0511|         numtrain_course-=1
0512|
0513|
0514| ##animation
0515|
0516| def creation_artists(num_max_trains, xlim1, ylim1):
0517|     global
plot_trains,annotations_trains,plot_gares,annotations_gares,plot
_trains_voie,fig1,train_ax
0518|     fig1 = pl.figure(figsize=(8, 8))
0519|     train_ax = pl.axes([0.2,0.1,0.7,0.8], xlim = xlim1,
ylim = ylim1)
0520|     activation_boutons()
0521|     train_ax.set_title('traffic RER')
0522|     plot_trains=[]; plot_gares=[]; annotations_trains=[];
annotations_gares=[]; plot_trains_voie=[]
0523|     for i in range(num_max_trains):
0524|         plot_trains.extend(train_ax.plot([], [],
marker='+', color='green',markeredgewidth = 8, markersize = 10))
0525|         plot_trains_voie.extend(train_ax.plot([],
[],color='red',linewidth = 5))
0526|
```

```
annotations_trains.append(train_ax.annotate(i,xy=(-1,-1),
color='green', annotation_clip = False, fontsize = 6))
0527|     for k in range(len(points)):
0528|         plot_gares.extend(train_ax.plot([],
[],marker='o',color='black',markersize = 1))
0529|         # if exemple and k%6==0:
0530|         #     texte = str(k) +
correspondance_traffic_DB([k], liste_id)[0][1]
0531|         # elif exemple:
0532|         #     texte = ''
0533|         texte = str(k)
0534|
annotations_gares.append(train_ax.annotate(texte,xy=(-1,-1),
xycoords='data',color='black', annotation_clip = False, fontsize
= 9))
0535|
0536|
0537| def init():
0538|     n = len(points)
0539|     dessiner_gares(n)
0540|     dessiner_graphe(n)
0541|     annoter_gare(n)
0542|     return plot_gares, annotations_gares   #ajouter
l'augmentation de la taille des gares en fct des voyageurs
0543|
0544| def anim(i, instance):
0545|     global pause
0546|     heure = t_depart+i*dt/60
0547|     if timer:
0548|         print('temps:', conversion_h_m(heure))
0549|     if pause:
0550|         print("l'éxecution est en pause, appuyer sur une
touche")
0551|         pl.waitforbuttonpress(-1)
0552|         pause = not pause
0553|     temps_suivant(heure)
0554|     n = len(points)
0555|     nb_trains = len(trains)
0556|     coef = 0.5
0557|     numeroVoie = 0
0558|     for numtrain in range(nb_trains):
0559|         if est_en_gare(numtrain):
0560|             numgare1 = int(trains[numtrain][5])
0561|             x,y = dessiner_trains_gare(numgare1,numtrain)
0562|         else:
0563|             numgare1, numgare2 = int(trains[numtrain][4]),
int(trains[numtrain][5])
0564|             x,y = dessiner_trains_voie(numgare1, numgare2,
numtrain, coef, numeroVoie, plot_trains_voie)
0565|             numeroVoie+=1
```

```
0566|             dessiner_trains(x,y,numtrain)
0567|         for k in range(numeroVoie, num_max_trains):
0568|             plot_trains_voie[k].set_data([],[])
0569|         for k in range(nb_trains, num_max_trains):
0570|             plot_trains[k].set_data([],[])
0571|             annotations_trains[k].set_position((-1,-1))
0572|         taille_des_gares()
0573|         annoter_gare(n)
0574|         return plot_trains, annotations_trains,
plot_trains_voie, plot_gares
0575|
0576|
0577| ##dessin
0578| def dessiner_trains_gare(numgare, numtrain):
0579|     pt = points[numgare]
0580|     return pt[0],pt[1]
0581|
0582| def dessiner_trains_voie(numgare1, numgare2, numtrain,
coef, numeroVoie, plot_trains_voie):
0583|     pt1, pt2 = points[numgare1], points[numgare2]
0584|     X,Y=[pt1[0],pt2[0]],[pt1[1],pt2[1]]
0585|     pos,vit = trains[numtrain][0:2]
0586|     vect = X[1]-X[0],Y[1]-Y[0]
0587|     x1,y1 = X[0]+pos*vect[0],Y[0]+pos*vect[1]
0588|     pos0 = pos-vit*dt/distances[numgare1,numgare2]*coef
0589|     x0,y0 = X[0]+pos0*vect[0],Y[0]+pos0*vect[1]
0590|     plot_trains_voie[numeroVoie].set_data([x0,x1],[y0,y1])
0591|     return x1,y1
0592|
0593| def dessiner_trains(x,y,numtrain):
0594|     if annote:
0595|
annotations_trains[numtrain].set_position((x+dimension,y+dimensi
on))  #eviter la  supersposition
0596|     else:
0597|         annotations_trains[numtrain].set_position((-1,-1))
0598|     if exemple:
0599|         plot_trains[numtrain].set_color('red')
0600|     elif total_voyageurs_dans_train(numtrain)>1700:
0601|         plot_trains[numtrain].set_color('red')
0602|     else:
0603|         plot_trains[numtrain].set_color('blue')
0604|     plot_trains[numtrain].set_data(x,y)
0605|
0606| def dessiner_gares(n):
0607|     for numgare in range(n):
0608|         pt = points[numgare]
0609|         plot_gares[numgare].set_data(pt[0],pt[1])
0610|         scalaire = total_voyageurs_en_gare(numgare)/100
0611|         if scalaire>10:
```

```
0612|             scalaire = 10
0613|             plot_gares[numgare].set_color('orange')
0614|         if scalaire<5:
0615|             scalaire = 5
0616|         plot_gares[numgare].set_markersize(scalaire)
0617|
0618| def annoter_gare(n):
0619|     for numgare in range(n):
0620|         pt = points[numgare]
0621|         if annote:
0622|             annotations_gares[numgare].set_position((pt[0]
+dimension, pt[1]-dimension))
0623|         else:
0624|
annotations_gares[numgare].set_position((-1,-1))
0625|
0626|
0627| def dessiner_graphe(n):
0628|     for lig in range(n):
0629|         for col in range(n):
0630|             if distances[lig,col]!=0:
0631|                 pt1 = points[lig]
0632|                 pt2 = points[col]
0633|                 X,Y=[pt1[0],pt2[0]],[pt1[1],pt2[1]]
0634|                 train_ax.plot(X,Y,color='gray',linewidth =
2.5)
0635|
0636| def taille_des_gares():
0637|     n = len(points)
0638|     for numgare in range(n):
0639|         if exemple:
0640|             scalaire = 7
0641|         elif affluence:
0642|             scalaire = total_voyageurs_en_gare(numgare)/
500
0643|             if scalaire<5:
0644|                 scalaire = 5
0645|             elif scalaire>10:
0646|                 scalaire = 10
0647|                 plot_gares[numgare].set_color('orange')
0648|         else:
0649|             scalaire = 5
0650|         plot_gares[numgare].set_markersize(scalaire)
0651|
0652|
0653|
0654| ## interaction (animation)
0655| def changer_etat(nom_var):
0656|     g = globals()
0657|     g[nom_var]= not g[nom_var]
```

```
0658|
0659| nom_func=["annote", "pause", "timer", "service",
"affluence"]
0660|
0661| #on ajoute au dictinnaire des variables global les axes et
les boutons
0662| #on les associe au changement d'état de la variable du
dictionnaire func_boutons
0663| def activation_boutons():
0664|     g = globals()
0665|     for i in range(len(nom_func)):
0666|         ax = pl.axes([0.01,0.1*i,0.1,0.1], xticks=[],
yticks=[])
0667|         btn = Button(ax, label = nom_func[i])
0668|         nom_button="btn_{}".format(i)
0669|         g[nom_button]=btn
0670|         globals()[nom_button].on_clicked(lambda event,i =
i:changer_etat(nom_func[i]))
0671|
0672|
0673|
0674|
0675| ##outil voyageurs
0676| #calcul du montant de voyageurs à numgare sur le quai du
train numtrain
0677| #permet d'affiner le temps d'arret
0678| def voyageurs_quai(numgare, numtrain):
0679|     total=0
0680|     freq_gare=gares[numgare][1]
0681|     directions_desservies=destinations[numtrain]
0682|     for destination in freq_gare:
0683|         if destination[0] in directions_desservies:
0684|             total+=destination[1]
0685|     return total
0686|
0687| #calcul du montant total de voyageurs dans la gare toutes
directions confondues, a un instant donné
0688| def total_voyageurs_en_gare(numgare):
0689|     total = 0
0690|     gare = gares[numgare]
0691|     frequentations = gare[1]
0692|     for destination in frequentations:
0693|         total+=destination[1]
0694|     return total
0695|
0696| #calcul du montant de voyageurs qui transitent par numgare
en une journée
0697| def total_voyageurs_jour(numgare):
0698|     total = 0
0699|     gare = gares[numgare]
```

```
0700|     frequentations = gare[1]
0701|     for destination in frequentations:
0702|         total+=destination[2]
0703|     return total
0704|
0705| #calcul du montant de voyageurs à bord
0706| def total_voyageurs_dans_train(numtrain):
0707|     total = 0
0708|     remplissage = trains[numtrain][6]
0709|     for destination in remplissage:
0710|         total+=destination[1]
0711|     return total
0712|
0713|
0714| ##voyageurs et remplissage du train
0715|
0716| #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)
0717| #les temps sont ici en minutes
0718| def monter_voyageurs(numtrain, numgare, heure):
0719|     freq=gares[numgare][1]
0720|     remplissage=trains[numtrain][6]
0721|     fraction, passagers_montants =
calcul_passagers_montant(numtrain, numgare)
0722|     directions_desservies=destinations[numtrain]
0723|     tps_total=0
0724|     nombre=0
0725|     if fraction>0:
0726|         for direction in freq:
0727|             gare_voulue=direction[0]
0728|             if gare_voulue in directions_desservies: #si
le train passe par la gare voulue
0729|
passagers_montants_dest=round(direction[1]*fraction, 3)
0730|                 minutes_attendues=(heure-direction[3])*60
0731|                 tps_total+=minutes_attendues
0732|                 nombre+=1
0733|
execute_monter_passagers(passagers_montants_dest, remplissage,
gare_voulue)
0734|             direction[3]=heure #reset temps d'attente
0735|     if nombre==0:
0736|         tps_attente_moyen=0
0737|     else:
0738|         tps_attente_moyen=round(tps_total/nombre, 4)
0739|     index=recherche_indice_suivi(numtrain)
0740|     stop_times[index][2][-1][2:4]=[passagers_montants,
tps_attente_moyen]
```

```
0741|
0742|
0743| #calcule la fraction des passagers qui va monter dans
numtrain en numgare
0744| def calcul_passagers_montant(numtrain, numgare):
0745|     passagers_voulant_monter=voyageurs_quai(numgare,
numtrain)
0746|     voyageurs_a_bord=total_voyageurs_dans_train(numtrain)
0747|     if passagers_voulant_monter==0:
0748|         return 0, 0
0749|     if
voyageurs_a_bord+passagers_voulant_monter<contenance:
0750|         passagers_montants=passagers_voulant_monter
0751|     else:
0752|         passagers_montants=contenance-voyageurs_a_bord
0753|     if passagers_montants>10: #permet d'alléger les
calculs sans vraiment fausser les résultats
0754|         fraction=passagers_montants/
passagers_voulant_monter
0755|     else:
0756|         fraction = 0
0757|     return fraction, round(passagers_montants)
0758|
0759| def execute_monter_passagers(passagers_montants_dest,
remplissage, gare_voulue):
0760|     rajout_destin=True
0761|     for destination in remplissage:  #il y-a-t-il des
passagers pour cette direction
0762|         gare_dir=destination[0]
0763|         if gare_voulue==gare_dir:
0764|             destination[1]+=passagers_montants_dest
0765|             rajout_destin=False
0766|     if rajout_destin:
0767|         remplissage.append([gare_voulue,
passagers_montants_dest])
0768|
0769| def descendre_voyageurs(numtrain, numgare):
0770|     remplissage=trains[numtrain][6]
0771|     for destination in remplissage:
0772|         if destination[0]==numgare:  #les voyageurs sont
arrivés à destination
0773|             remplissage.remove(destination)
0774|             break #ne sert à rien de continuer
0775|
0776| ## voyageurs et remplissage des gares
0777| '''plages liste les freq relatives des flux de voyageurs
découpée par heures'''
0778|
0779| #on fait arriver les voyageurs juste avant l'arrivée du
train numtrain pour optimiser les calculs et donc que selon les
```

```
directions desservies
0780| def augmenter_voyageurs_gares(numgare, numtrain, heure):
#deltat est en heures, longueur de plage aussi
0781|     num_p = int(heure)
0782|     directions_desservies=destinations[numtrain]
0783|     freq_gare=gares[numgare][1]
0784|     for destination in freq_gare:
0785|         if destination[0] in directions_desservies:
0786|             voyageurs_par_jour = destination[2]
0787|             deltat = heure- destination[3]
0788|             destination[1]
+=plages[num_p]*voyageurs_par_jour*deltat
0789|
0790|
0791|
0792| ##outils temps
0793| #int pour enlever les 1.0 -> 1
0794| def conversion_h_m(heure):
0795|     h = int(np.floor(heure))
0796|     mins = int((heure-h)*60)
0797|     return str(h)+':'+str(mins)
0798|
0799| def conversion_minutes(liste_h_m):
0800|     liste_h_m = liste_h_m.split(':')
0801|     heure = float(liste_h_m[0])*60+float(liste_h_m[1])
0802|     return heure
0803|
0804| def conversion(vit, accel):
0805|     return vit/1000*60, accel/60
0806|
0807| ##statistiques
0808| '''
0809| temps_cumulé = somme(temps d'attente sur une gare avant de
monter dans un train * nb_voyageurs montant dans le train)
0810| temps_cumule_gare
0811| voyageurs_desservis_total
0812| voyageurs_desservis_gare = voyageurs_desservis par gare
(liste)
0813| tps_moyen_total = temps_cumulé/voyageurs_desservis_total
0814| tps_moyen_gare = le temps_d'attente moyen par gare (liste)
0815| '''
0816|
0817| def statistiques_attente():
0818|     global temps_cumule_gare, voyageurs_desservis_gare,
tps_moyen_gare
0819|     temps_cumule, voyageurs_desservis_total =
ponctualite_charge()
0820|
0821|     #on calcule aussi des stats plus détaillées par gare:
0822|     tps_moyen_gare = zeros.copy()
```

```
0823|     for numgare in range(len(gares)):
0824|         if voyageurs_desservis_gare[numgare]!=0:
0825|
tps_moyen_gare[numgare]=temps_cumule_gare[numgare]/
voyageurs_desservis_gare[numgare]
0826|         else:
0827|             tps_moyen_gare[numgare]=-1  #non desservie
0828|
0829|     if voyageurs_desservis_total!=0:
0830|         tps_moyen_total = temps_cumule/
voyageurs_desservis_total
0831|     else:
0832|         tps_moyen_total = 0
0833|     if affichage:
0834|         print('temps cumulé', round(temps_cumule/60, 2),
"heures")
0835|         print(round(voyageurs_desservis_total), 'voyageurs
desservis')
0836|         print("soit un tps d'attente moyen de",
round(tps_moyen_total,2), "minutes")
0837|         print("pour le detail par gare voir
temps_cumule_gare, tps_moyen_gare et voyageurs_desservis_gare")
0838|
0839| # calcule le temps attendu par les voyageurs*nombre de
voyageurs qui ont attendu
0840| #tps_attendu est en minutes
0841| def ponctualite_charge():
0842|     global temps_cumule_gare, voyageurs_desservis_gare
0843|     voyageurs_desservis_total = 0; temps_cumule = 0
0844|     temps_cumule_gare = zeros.copy();
voyageurs_desservis_gare = zeros.copy()
0845|     for train in stop_times_stock:
0846|         for passage in train[2]:
0847|             [numgare, heure, voy,tps_attendu] = passage
0848|             tps = voy*tps_attendu
0849|             temps_cumule+=tps
0850|             voyageurs_desservis_total+=voy
0851|             temps_cumule_gare[numgare]+=tps
0852|             voyageurs_desservis_gare[numgare]+=voy
0853|     return temps_cumule, voyageurs_desservis_total
0854|
0855| ## écarts aux horaires en traffic normal - critère de
ponctualité
0856| '''il faut faire correspondre stop_times0 qui est
[trip_id, [horaire1, horaire2, ...] et dont les trip_id sont
ordonnées de manière décroissante
0857| avec stop_times qui est [numtrain, trip_id, [groupe_id1,
horaire1], [groupe_id2, horaire2], ...]
0858| il arrive que stop_times contiennent moins d'horaires que
stop_times0 lorsque les trains ne finissent pas leur parcours'''
```

```
0859| '''dans ces modules il faut avoir au préalable converti
les id_traffic id_groupe'''
0860|
0861| def ponctualite():
0862|     recuperation_groupe_id()
0863|     carre_ecart = 0
0864|     ecart_flat = 0
0865|     for trip in stop_times_stock:
0866|         trip_id = trip[1]
0867|         index = recherche_index_dichotomie(trip_id)
0868|         trip0 = stop_times0[index]
0869|         dep_times0 = trip0[1]
0870|         passages = trip[2]
0871|         n=len(passages)
0872|         N=len(dep_times0)
0873|         for i in range(N):
0874|             h1 = float(dep_times0[i])
0875|             if i<n:
0876|                 h2 = float(passages[i][1])
0877|                 carre_ecart+=(h2-h1)**2 #écart en heure^2
0878|                 ecart_flat+=h2-h1
0879|     # if ecart_flat<0:
0880|     #     print('modèle en avance sur la théorie',
ecart_flat)
0881|     # else:
0882|     #     print('modèle en retard sur la théorie',
ecart_flat)
0883|     return np.sqrt(carre_ecart)
0884|
0885|
0886| #renvoie l'index dans stop_times0 de trip_id
0887| def recherche_index_dichotomie(trip_id):
0888|     N = len(stop_times0)
0889|     mini = 0; maxi = N-1
0890|     index = int((mini+maxi)/2)
0891|     while mini!=index and maxi!=index:
0892|         if trip_id>stop_times0[index][0]:
0893|             maxi = index
0894|         else:
0895|             mini = index
0896|         index = int((mini+maxi)/2)
0897|     if trip_id!=stop_times0[index][0]:
0898|         print('erreur')
0899|     else:
0900|         return index
0901|
0902| ##suivi des trains
0903| #numtrain1 arrive en gare2, s'il est suivi on ajoute
l'horaire de passage à trains suivis
0904| def suivi_ajout_passage(numtrain1, gare2, heure):
```

```
0905|     index = recherche_indice_suivi(numtrain1)
0906|     stop_times[index][2].append([gare2, heure, 0, 0])
0907|
0908| #donne les trains dans la gare numgare
0909| def recherche_train_dans_gare(numgare):
0910|     rep=[]
0911|     for numtrain in range(len(trains)):
0912|         train = trains[numtrain]
0913|         if train[4]==-1 and train[5]==numgare:
0914|             rep.append(numtrain)
0915|     return rep
0916|
0917| #retourne l'index dans la liste stop_times du train
numtrain
0918| def recherche_indice_suivi(numtrain):
0919|     for index in range(len(stop_times)):
0920|         if stop_times[index][0]==numtrain:
0921|             return index
0922|     print('le train', numtrain, 'circulait sans être suivi
en suivi_global')
0923|
0924| #on remplace dans la liste préexistente le numgare par sa
correspondance dans la DB en terme de id_groupe
0925| def recuperation_groupe_id():
0926|     for train in stop_times_stock:
0927|         n=len(train[2])
0928|         for index_passage in range(n):
0929|             numgare = train[2][index_passage][0]
0930|             stop_groupe = liste_id[numgare]
0931|             train[2][index_passage][0]=stop_groupe
0932|
0933| ##suivi d'un profil de course
0934| #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
0935| #on n'utilisera pas ce module pour calculer ponctualite
car on n'est pas sur du profil theorique
0936| def gestion_profil(heure):
0937|     if numtrain_course!=-1:
0938|         train_suiv=trains[numtrain_course]
0939|         pos, vit, accel, tps, gare_dep, gare_ar,
remplissage = train_suiv[0:7]
0940|         if gare_ar!=-1:
0941|             vit, accel = conversion(vit, accel)
0942|             if len(profil_suivi)>0 and profil_suivi[-1]
[0]==gare_dep:
0943|                 #toujours sur le même intergare
0944|                 profil_suivi[-1].append([pos, heure, vit,
accel, []])
0945|             else:
```

```
0946|                 profil_suivi.append([gare_dep, gare_ar,
[pos, heure, vit, accel, copy.deepcopy(remplissage)]])
0947|
0948|
0949| def profil_theorique(trip_id, pos_ax):
0950|     global num_horaire_course
0951|     num_horaire_course = recherche_indice_horaire(trip_id)
0952|     index = recherche_index_dichotomie(trip_id)
0953|     stop_time = stop_times0[index][1]
0954|     #il faut récupérer les destinations pour avoir les
distances
0955|     num_typ = horaires0[num_horaire_course][0]
0956|     typ = types[num_typ]
0957|     gare_dep = typ[5]
0958|     feuille_route = [gare_dep] + dest[num_typ][:-1]   #on
enlève le -1 (fictif) et on rajoute le terminus de départ
0959|     direc = direct[num_typ]
0960|     profil_th = []
0961|     dist_tot = 0
0962|     indice_temps = 0
0963|     n = len(feuille_route)
0964|     for indice_feuille in range(n-1):
0965|         gare_dep = feuille_route[indice_feuille]
0966|         gare_ar = feuille_route[indice_feuille+1]
0967|         t_dep = stop_time[indice_temps]
0968|         if gare_dep not in direc: #le train s'arrete bien
et donc il est pris en compte dans stop_time
0969|             pos_ax.scatter(x=t_dep, y=dist_tot, s=8,
color='blue')
0970|             indice_temps+=1
0971|         #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
0972|         dist_tot+= distances[gare_dep][gare_ar]
0973|     t_ar = stop_time[indice_temps]
0974|     pos_ax.scatter(x=t_ar, y=dist_tot, s=8, color='blue')
0975|
0976|
0977|
0978| def dessin_course(vit_ax, pos_ax, remp_ax, accel_ax,
color):
0979|     n=len(gares)
0980|     global remp_liste
0981|     t_liste=[]; t_cut_liste=[]; pos_liste=[];
vit_liste=[]; accel_liste=[]; bins=[]
0982|     remp_liste=[[] for k in range(n)]; weight_liste=[[]
for k in range(n)]; labels=['']*n
0983|     distance_parcourue = 0
0984|     compt = 0
0985|     # lim_remp = 17; lim_vit = 40; inf = 15 #pour l'étude
```

```
de trip_id_course='115054849-1_16156'
0986|     lim_remp = np.inf; lim_vit = np.inf; inf = 0 #pour
l'étude de trip_id_course='115072256-1_14393'
0987|     t0=profil_suivi[0][2][1]
0988|     t0_cut=t0
0989|     for intergare in profil_suivi:
0990|         gare_dep = intergare[0]
0991|         gare_ar = intergare[1]
0992|         # print(correspondance_traffic_DB([gare_dep],
liste_id)[0][1])
0993|         # print(gare_dep)
0994|         if gare_dep!=-1:
0995|             dist_gares = distances[gare_dep][gare_ar]
0996|             if dist_gares==0:
0997|                 print('ce train est direct')
0998|         else:
0999|             dist_gares = 0
1000|
1001|         for pos, heure, vit, accel, remplissage in
intergare[2:]:
1002|             dist = pos*dist_gares
1003|             pos_liste.append(dist+distance_parcourue)
1004|             t_liste.append(heure)
1005|             if compt<inf:
1006|                 t0_cut=heure
1007|             else:
1008|                 if compt<lim_vit:
1009|                     t_cut_liste.append(heure)
1010|                     vit_liste.append(vit)
1011|                     accel_liste.append(accel)
1012|                     tf_cut=heure
1013|         if gare_dep!=-1:
1014|             pos, heure, vit, accel, remplissage =
intergare[2]
1015|             if compt<lim_remp:
1016|                 bins.append(heure)
1017|                 tot=0
1018|                 for numgare, passagers in remplissage:
1019|                     if passagers>40:
1020|                         remp_liste[numgare].append(heure)
1021|
weight_liste[numgare].append(passagers)
1022|                         remp_ax.text(heure, tot+passagers/
2, str(int(passagers)), fontsize=7)
1023|                         name =
correspondance_traffic_DB([numgare], liste_id)[0][1]
1024|                         if passagers>200:
1025|                             labels[numgare]=name
1026|                         tot+=passagers
1027|         tf = intergare[-1][1]
```

```
1028|         bins.append(tf)
1029|         distance_parcourue+=dist_gares
1030|         pos_ax.plot([tf], [distance_parcourue],
marker='o', markersize = 4, color=color)
1031|         compt+=1
1032|     pos_ax.plot(t_liste, pos_liste, color=color)
1033|     vit_ax.plot(t_cut_liste, vit_liste, marker='o',
markersize = 4)
1034|     accel_ax.plot(t_cut_liste, accel_liste)
1035|     remp_ax.hist(remp_liste, weights=weight_liste,
bins=bins, histtype='barstacked', label=labels)
1036|     remp_ax.legend(loc='upper right', fontsize=8)
1037|     return t0, tf, t0_cut, tf_cut
1038|
1039|
1040| def reperes(remp_ax, vit_ax, accel_ax, t0, tf, t0_cut,
tf_cut):
1041|     v_nom, ac_nom = conversion(vitesse_nominale,
accel_nominale)
1042|     v_ar, fr_nom = conversion(vitesse_arrivee,
freinage_nominal)
1043|     lims=[t0, tf]
1044|     lims_cut=[t0_cut, tf_cut]
1045|     vit_ax.plot(lims, [v_nom, v_nom])
1046|     vit_ax.plot(lims, [v_ar, v_ar])
1047|     accel_ax.plot(lims, [ac_nom, ac_nom])
1048|     accel_ax.plot(lims, [fr_nom, fr_nom])
1049|     vit_ax.set_xlim(lims)
1050|     accel_ax.set_xlim(lims)
1051|     remp_ax.plot(lims, [contenance, contenance])
1052|     remp_ax.set_xlim(lims)
1053|
1054|
1055|
1056|
1057|
1058|
1059|
1060|
1061|
1062| def comparaison(trip_id):
1063|     fig2, axes = pl.subplots(2, 2, figsize=[23, 16])
1064|     for i in range(2):
1065|         for j in range(2):
1066|             axes[i][j].set_xlabel('temps (heures)')
1067|     [[pos_ax, remp_ax], [vit_ax, accel_ax]] = axes
1068|     pos_ax.set_title('position modèle et théorique')
1069|     pos_ax.set_ylabel('distance depuis terminus (m)')
1070|     remp_ax.set_title('remplissage modèle')
1071|     remp_ax.set_ylabel("nombre de passagers en fonction de
```

```
la gare d'arrivée")
1072|     accel_ax.set_title('acceléréation modèle m/s^2')
1073|     vit_ax.set_title('vitesse modèle')
1074|     vit_ax.set_ylabel('vitesse (km/h)')
1075|     profil_theorique(trip_id, pos_ax)
1076|     color = 'black'
1077|     t0, tf, t0_cut, tf_cut = dessin_course(vit_ax, pos_ax,
remp_ax, accel_ax, color)
1078|     reperes(remp_ax, vit_ax, accel_ax, t0, tf, t0_cut,
tf_cut)
1079|     pl.savefig('calage du modèle (profil course de
traffic)', dpi=300,  bbox_inches='tight')
1080|     pl.show()
1081|
1082| # renvoie l'indice de trip_id dans horaires0
1083| def recherche_indice_horaire(trip_id):
1084|     n=len(horaires0)
1085|     for k in range(n):
1086|         if trip_id==horaires0[k][2]:
1087|             return k
1088|     print('erreur de trip_id, inexistant dans horaires0')
1089|
1090|
1091| ## utilisation dans la db
1092| #on remplit avec notre modèle stop_times_modif
1093| #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
1094| def reecriture_DB():
1095|     conn = sql.connect(r"F:/informatique/TIPE/database/
produit exploitable/GTFS.db")
1096|     c = conn.cursor()
1097|     c.execute('delete from stop_times_modif')
1098|     for train in stop_times_stock:
1099|         trip_id = train[1]
1100|         for passage in train[2]:
1101|             [id_groupe, departure_time, voy, tps]=passage
1102|             departure_time =
conversion_h_m(departure_time)
1103|             c.execute('''
1104|             insert into stop_times_modif(route_id,
trip_id, departure_time, id_groupe, stop_sequence)
1105|             values(?,?,?,?,?)''', (route_id, trip_id,
departure_time, id_groupe, stop_sequence))
1106|     conn.commit()
1107|     conn.close()
1108|
1109| def ecart_db(affichage=False):
1110|     if affichage:
```

```
1111|         ec_fig, ax=pl.subplots(figsize=[23, 16])
1112|     conn = sql.connect(r"F:/informatique/TIPE/database/
produit exploitable/GTFS.db")
1113|     c = conn.cursor()
1114|     c.execute('''
1115|     select s1.departure_time, s2.departure_time,
s1.trip_id
1116|     from stop_times as s1
1117|     join stop_times_modif as s2
1118|         on s1.trip_id = s2.trip_id and s1.stop_sequence =
s2.stop_sequence
1119|     ''')
1120|     carre_ecart = 0; ecart_flat = 0
1121|     for h1, h2, trip_id in c:
1122|         h1 = conversion_minutes(h1); h2 =
conversion_minutes(h2)
1123|         carre_ecart+=(h2-h1)**2 #écart en min^2
1124|         ecart_flat+=h2-h1
1125|         if affichage:
1126|             ax.scatter(h1, 0); pl.scatter(h2, 1)
1127|             ax.annotate(s='', xy=(h1, 0), xytext=(h2, 1),
arrowprops=dict(arrowstyle="->", lw=0.5, mutation_scale=1))
1128|     conn.close()
1129|     print('écart réduit ', np.sqrt(carre_ecart))
1130|     if affichage:
1131|         pl.savefig('régulation', dpi=300,
bbox_inches='tight')
1132|         pl.show()
1133|
1134|
1135|
1136|
1137| ##perturbation
1138| def execute_arret_force(numtrain, tfin):
1139|     arret_force.append([numtrain, tfin])
1140|     if trains[numtrain][4]!=-1: #pas en gare
1141|         execute_arret(numtrain)
1142|     #si en gare, le train ne redémarre tout simplement pas
tant qu'il est en arret_force
1143|
1144| def gestion_arret_force(heure):
1145|     n=len(arret_force)
1146|     for k in range(n-1, -1, -1):
1147|         if arret_force[k][1]<heure:
1148|             arret_force.remove(arret_force[k])
1149|
1150| # teste si numtrain est en arret forcé
1151| def arret_force_de(numtrain):
1152|     for train in arret_force:
1153|         if train[0]==numtrain:
```

```
1154|                 return True
1155|         return False
1156|
1157|
1158| ##régulation
1159| #on détermine l'horizon de régulation en supposant
connaitre la durée de la régulation
1160|
1161| #heure est en heures
1162| # 0.2 h = 12 min
1163| def variables_de_pert(duree, tm, g1, g2):
1164|     global duree_pert_est, tmin, gare_pert1, gare_pert2,
horizon_retarde
1165|     duree_pert_est=duree
1166|     tmin=tm
1167|     facteur_pert=4
1168|     gare_pert1=g1; gare_pert2=g2
1169|     horizon_retarde = duree_pert_est*facteur_pert
1170|
1171| #on veut que le dernier train à partir ne soit pas retardé
1172| def retarder_departs(heure):
1173|     global trains_retardes
1174|     [name1, name2] =
correspondance_traffic_DB([gare_pert1, gare_pert2], liste_id)
1175|     conv_tmin = conversion_h_m(tmin)
1176|     conv_tmax =  conversion_h_m(tmin+duree_pert_est)
1177|     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)
1178|     print(destinations[numtrain_course])
1179|     trains_retardes = []
1180|     N=len(trains)
1181|     # print(numtrain_course, trains[numtrain_course],
destinations[numtrain_course], gare_pert1, gare_pert2)
1182|
1183| #on arrete les trains
1184|     for numtrain in range(N):
1185|         ret = retard(heure)
1186|         destin = destinations[numtrain]
1187|         if test_destination(destin):
1188|             trains_retardes.append(numtrain)
1189|             execute_arret_force(numtrain, heure+ret)
1190|     if consigne_pert:
1191|         dessin_regulation()
1192|         pl.plot([heure, heure+ret], [1, 2], marker='o',
color='black')
1193|     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])
1194| #on retarde directement le départ des trains considérés
1195|     num_h=0; dep_time=0 #initialisation
1196|     print('horizon retardé : ',
conversion_h_m(horizon_retarde), ' min \n')
1197|     while dep_time<horizon_retarde + tmin:
1198|         destin = dest[num_h]
1199|         dep_time=horaires[num_h][1]
1200|         # print(correspondance_traffic_DB(destin,
liste_id), test_destination(destin), '\n')
1201|         if test_destination(destin):
1202|             ret = retard(dep_time)
1203|             hor_dep = horaires[num_h][1]
1204|             new_hor = hor_dep + ret
1205|             horaires[num_h][1] = new_hor
1206|             if consigne_pert:
1207|                 pl.plot([hor_dep, new_hor], [1, 2],
marker='o', color='black')
1208|
1209|         num_h+=1
1210|     if consigne_pert:
1211|         pl.savefig('retards en consigne des trains',
dpi=300,  bbox_inches='tight')
1212|         pl.show()
1213|
1214|
1215| def retard(heure):
1216|     return (tmin+horizon_retarde-heure)*duree_pert_est/
horizon_retarde
1217|
1218| #on regarde si la direction définie par gare_pert1 et
gare_pert2 est dans destination
1219| def test_destination(destin):
1220|     n = len(destin)
1221|     for k in range(n-1):
1222|         if gare_pert1==destin[k] and
gare_pert2==destin[k+1]:
1223|             return True
1224|     return False
1225|
1226| def mise_en_place_pert(heure):
1227|     if heure<tmin and heure+dt/60>=tmin:
1228|         retarder_departs(heure)
1229|
1230| def dessin_regulation():
1231|     pl.figure(figsize=[23, 16])
1232|     pl.xlabel('temps (heures)')
1233|     pl.title('retards en consigne des trains')
1234|     pl.plot([tmin, horizon_retarde + tmin], [1, 1])
1235|     pl.plot([tmin, horizon_retarde + tmin], [2, 2])
```

```
1236|
1237|
1238|
1239|
1240| ##paramètres du modèle et calage
1241| '''Tout est mis en m/min
1242| vit commerciale : 50 km.h-1
1243| vit_nominale : 65 kmh
1244| capacité totale 1700
1245| on calcule accel nominale=3912
1246| '''
1247|
1248| vitesse_nominale = round(65000/60)  #65km/h en m/min
1249| vitesse_arrivee=round(vitesse_nominale/3) #le train doit
arriver en gare à allure raisonnable
1250| contenance = 1700
1251| tps_arret = 0.5 #30 sec
1252| tps_population = 1/10000 #pour 10000 voyageurs en gare, le
train attendra une minute de plus (à ajuster en fonction des
paramètres de flux)
1253| distance_freinage = 300
1254| distance_demarrage = 150 #longueur d'une gare
1255| distance_securite = 800
1256| distance_securite_mini = 500 #deux trains ne doivent
jamais être à moins de 500m
1257| 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
1258| freinage_nominal=round(-accel_nominale/1.5)
1259|
1260| var=[vitesse_nominale, vitesse_arrivee, contenance,
tps_arret, tps_population, distance_freinage,
distance_demarrage, accel_nominale, freinage_nominal]
1261| delt0=300 #m/min
1262| delt1=300 #m/min
1263| delt2=300 #passagers
1264| delt3=0.3 #min
1265| delt4=tps_population/2
1266| delt=[delt0, delt1, delt2, delt3, delt4]
1267|
1268| bornes=[[var[k]-delt[k], var[k]+delt[k]] for k in
range(5)]
1269| variabs=[var[k] for k in range(5)]
1270| infinitesimaux=[50, 0.05, 0.1, 30, 30]
1271| '''' minimum(retour_ecart, var, bornes, infinitesimaux, 1,
0.01, False)
1272| la derive_partielle de tps_population est nul ...
1273|
1274| Lorsque dt est petit (9 sec), l'exécution du programme est
plus lente et les trains sont en avance sur la théorie
```

```
1275| pour dt grand (20 sec), les trains sont en retard sur la
théorie
1276| Les paramètres dépendent du dt choisi...
1277|
1278| monteCarlo(retour_ecart, bornes, 10)
1279|
1280| '''
1281|
1282| 'résultat du monte carlo :'
1283| # vitesse_nominale = 1150 #contre 1083 m/min dans le
modèle original.
1284| #cela corrrespond à 69 km/h
1285| # vitesse_arrivee = 300 #contre 361 m/min --> faible
différence
1286| #surtout il y a une forte dispertsion de vitesse arrivée
aux différents minima --> peu d'impact dans la modélisation
1287|
1288| #trop de fluctuation pour tps_arret
1289|
1290| #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
1291|
1292|
1293| ## initialisation
1294| try:
1295|     test = horaires0[0]
1296| except NameError: #si c'est la première fois
1297|     donnees_GTFS("810:B")
1298|
```