Guide Technique du programme python de traitement du bilan carbone

C'est en cours de rédaction ! Il est encore très incomplet, mais les choses qui s'y trouvent sont valides (et si elles ne le sont pas, faites-le moi remarquer !)

GUIDE TECHNIQUE DU PROGRAMME PYTHON DE TRAITEMENT DU BILAN CARBONE 1

1. Arborescence du projet 4

2. Vigilances générales 5
Format des données 5
Ecriture des résultats 5
Casse 5
Version 5
Coordonnées GPS 6

3. Structures récurrentes 6
Lecture d'un fichier Excel 6
Recherche d'un élément par nom exact 7
Recherche d'un élément par nom approximatif 7
Ouverture d'un fichier Excel 8

4. Fichiers Données 9

5. Fichiers Sources 10
a. BDDgeoloc.xlsx 11
b. Entrees Manuelles 11
i. Données Entreprises 11
Chiffre d'affaire 12
Nombre de Salarié·es 12
Production en m3 12
Consommation Fuel 12
Consommation Electrique 12
ii. Facteurs Emissions MP 12
FE par famille 12
Famille, et teneur en azote 12
Masses volumiques 13
Engrais et azotes 13
iii. Autres FE 13
iv. Import Etranger Maritime et Ter 13
v. Export Etranger Maritime et Ter 14
vi. Sacherie 14

6. Traitement 15
a. main.py 15
i. Lecture des fichiers généraux 16

ii. Gestion des intrants ... 16
Lecture des achats ... 16
Calcul des distances de fret ... 17
Facteurs d'émission du fret ... 19
Fabrication des intrants ... 21
Compilation fret et fabrication ... 22
Fret par usine ... 24
iii. Gestion des emballages et de la sacherie ... 25
b. GestionIntrants.py et fonctionsFret.py ... 26
i. Fonction lireBDDMP() (gestionIntrants.py) ... 26
ii. Fonction gestionpb_volume_sac() (gestionIntrants.py) ... 27
iii. Fonction calc_fret_par_MP() (fonctionsFret.py) ... 29
iv. calcul_distance_fret_amont() ... 32
v. Fonctions lireImportTerrestre() et lireImportMaritime() (fonctionsFret.py) ... 33
vi. Fonction lire_FEtransport () (fonctionsFret.py) ... 34
vii. Fonction lireFE_matprem() (gestionIntrants.py) ... 34
viii. Fonction associerMPetFE_fab (gestionIntrants.py) ... 34
ix. Fonction associer_fret_FEfab() (gestionIntrants.py) ... 35
x. Fonction corriger_noms_massesvol (gestionIntrants.py) ... 41
xi. Fonction calc_quantite_et_BC (gestionIntrants.py) ... 42
xii. Fonction renverserMP_usine_fret (fonctionsFret.py) ... 42
xiii. Fonction calc_fin_de_vie (gestionIntrants.py) ... 42
c. GestionSacherie.py ... 42
i. Fonction lireConsoSacherie() (gestionSacherie.py) ... 42
ii. Fonction qte_materiaux_sacherie () (gestionSacherie.py) ... 42
iii. Fonction lire_FE_emballage() ... 44
iv. Fonction BC_sacherie () (gestionSacherie.py) ... 44
d. gestionExport.py ... 44
i. lireExport() ... 44
ii. lire_couts_camion_aval() ... 45
iii. lire_ventes() ... 46
iv. regrouper_livraison() ... 46
v. calc_fret_aval() ... 47
vi. lire_conversion_sacherie_GCO() ... 48
vii. optimiser_livraisons() ... 49
optimiser_livraisons() ... 49
parcours() ... 49
solve_tsp_dynamic() ... 50
viii. TracerVentes() ... 51
e. autres.py ... 51
a. Electricité ... 51
b. Fuel ... 52
c. Immobilisations ... 52
d. Déplacements ... 53
f. resultat.py ... 53

7. Résultat ... Erreur ! Signet non défini.

8. Interaction utilisateur·ice ... Erreur ! Signet non défini.

9. Maintenance ... 56
a. Maintenance niveau bas ... 56
i. Changer l'année de calcul ... 56

ii. Changer les emplacements des fichiers 57
iii. Changer les postes calculés/afficher des graphes/afficher les messages de console 57
b. Maintenance niveau moyen 57
i. Calculer une nouvelle année de bilan carbone 57
ii. Ajouter un fournisseur international 58
iii. Ajouter/mettre à jour un catalogue de sacherie 58
iv. Modifier les facteurs d'émission 58
v. Ajouter une usine 58
c. Maintenance haut niveau 58
i. Ajouter des panneaux solaires à un site 58

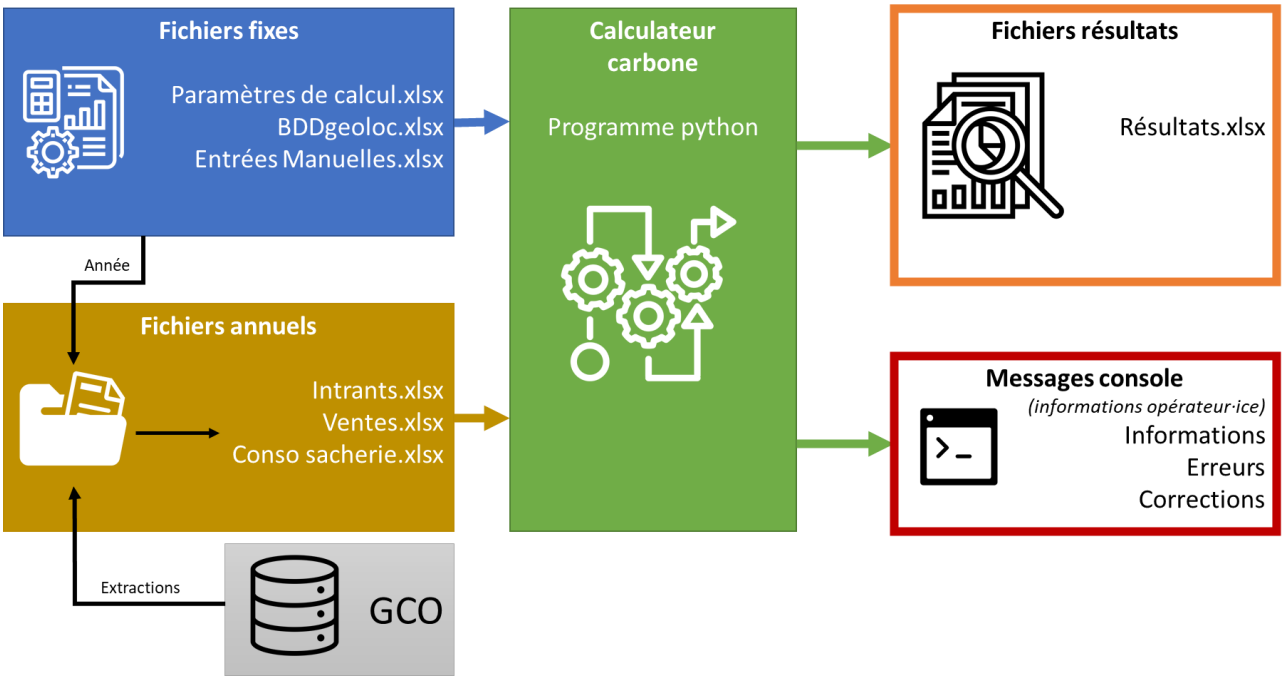
1. Arborescence du projet

Le programme est langage python, découpé en plusieurs fichiers .py. Ils sont regroupés ensembles dans le dossier global nommé « Calculateur Carbone ». L'exécution entraine également la création d'un dossier « _pycache_ » sur lequel il n'est pas souhaitable d'intervenir.

Les fichiers .py sont les suivants :

- main.py : Corps principal du programme qui est exécuté.
- fonctionsMatrices.py : Regroupe des opérations usuelles de traitement des tableaux, comme la recherche d'un élément, le tri, …
- Parametres.py : C'est là où l'utilisateur·ice peut modifier quelques paramètres aisément, pour de la maintenance de moyen niveau.
- gestionIntrants.py : gère la lecture de la base de données des intrants (fichier données/intrants.xlsx), celle des facteurs d'émissions, masses volumiques, etc, le traitement de ces données
- gestionExport.py : gère la lecture du fichier des exports (ventes) sortant des usines, opère leur traitement et calcule leur bilan carbone
- geolocalisation.py : est utile afin de calculer des distances de frets. Permet de lire la base de données des géolocalisations (fichier sources/BDDgeoloc), de calculer des distances à partir de codes postaux, de coordonnées GPS, etc… Permet également de faire remonter des codes postaux invalides et de les inscrire en sortie du programme
- fonctionsFret.py : permet de gérer le fret des intrants, par route ou par bateau.
- affichageResultats.py : gère l'écriture d'un fichier Excel résumant les résultats.
- autres.py : gère les postes secondaires (énergie, immobilisations et déplacements)
-

Le fonctionnement global, en termes d'entrées/sorties, est pour rappel résumé ici :



Pour plus d'informations sur la méthodologie bilan carbone, l'explication de comment sont croisées les données, et la justification des facteurs d'émissions, il faut se référer au document « Méthodologie du calculateur carbone 2020 » (en cours de rédaction).

2. Vigilances générales

Format des données

Les données dans les fichiers Excel que le programme lit, lorsqu'elles sont des nombres à virgules, doivent être inscrits comme tels dans le fichier Excel (en français, les décimales sont séparées par une virgule, le nombre est aligné sur la droite). Autrement, le programme les comprendra comme des chaines de caractères et non comme des nombres.

Ecriture des résultats

Le programme produit des fichiers Excels comme résultats, et a donc besoin de droits d'accès en écriture, qu'il n'a probablement pas dans les dossiers partagés sur un serveur. Il faut donc spécifier une direction locale (disque dur de l'ordinateur ou clé USB).

De plus, si le programme doit écrire sur un dossier existant, il ne pourra pas le faire si ce dossier est ouvert par ailleurs par l'utilisateur·ice.

Casse

Sauf mention explicite, la casse n'est pas importante. Par exemple, lorsque l'on rechercher la correspondance de « ECORCE PIN MARITIME COMPOSTEE 00/10 », on pourra trouver « Ecorce Pin Maritime Compostee 00/10 ». Cependant, les caractères doivent correspondre totalement. On ne trouvera pas par exemple « Ecorce_Pin Maritime Compostée 00-10 » à cause des symboles _, é et – qui ne correspondent pas.

Version

Ce programme a été écrit dans la version 3.8 de Python. Il ne fait appel à rien d'extravagant, mais des incompatibilités peuvent subvenir à l'avenir.

Les modules importés sont :

Time : Horloge, mesure du temps d'exécution des différentes parties

Xlrd : Lecture de fichiers Excel

Colorama : Utilisation de couleurs dans la console pour la lisibilité

Logging : Enregistrement en log des différents niveaux d'alerte lors de l'exécution

Random : Génération de nombres aléatoires

Inspect : Analyse du stack d'exécution, utile pour la localisation des erreurs lors du logging

Numpy : Gestion de matrices dynamiques

Math : Principalement des calculs

Openpyxl : Ecriture et mise en forme de fichiers Excel

Pathlib : Accès aux chemins du système

Matplotlib.pyplot : Tracé de graphes, diagrammes, cartes…

Toutes ces librairies sont (sauf erreur) comprises dans l'installation d'Anaconda[1] (conseillé). Sinon, elles sont installables à l'aide de Pip[2].

[1] Se référer à la partie 8, page 31 pour éditer le code
[2] Fonctionnement de Pip détaillé ici : https://pypi.org/project/pip/

Coordonnées GPS

Par convention, toutes les coordonnées GPS présentes et utilisées dans le code sont en radians (donc $lat_{rad} = lat_{degré} * \frac{\pi}{180}$, idem pour les longitudes.

De plus, lorsqu'un couple longitude latitude est donnée, il est toujours dans cet ordre, longitude puis latitude. Il en va de même dans les colonnes des fichiers *BDDgéoloc* et *Entrées Manuelles*, où la première colonne correspond toujours aux longitudes.

3. Structures récurrentes

Afin d'aider à la compréhension du code, on liste ici les structures utiles que l'on retrouve fréquemment, et qu'il est bon d'avoir en tête pour le comprendre plus naturellement. Elles ne sont souvent pas l'objet d'une fonction dédiée, car la forme des éléments qu'on leur soumet n'est pas toujours exactement la même, mais le fonctionnement est similaire.

Lecture d'un fichier Excel

Exemple : fonction **lireImportMaritime**()

```
def lireImportMaritime(facteur_route:int):
    document = xlrd.open_workbook(p.FICHIER_ENTREES_MANUELLES)
    feuille_bdd_mar = document.sheet_by_index(3)

    #Récupération des données du tableau
    lieux =[]
    for lin in range(2,17):
        ligne = []
        for col in [1,2,3,6,7]:
            ligne.append(feuille_bdd_mar.cell_value(lin,col))
        lieux.append(ligne)
```

Figure 1 Exemple d'une lecture de tableau Excel avec lireImportMaritime()

On lit les fichiers Excel avec le module « *xlrd* » (bien penser à l'importer avant de l'utiliser). On ouvre le document, ici celui des entrées manuelles, avec la première ligne et la fonction **xlrd.open_workbook**(). Puis on ouvre la feuille en choisissant l'indice (commençant à 0) dans le fichier Excel. Ensuite, on initialise un tableau vide qui contiendra les résultats. Puis on itère sur les lignes avec la première boucle for, selon les indices de ligne dans lesquelles se trouvent les données.

On initialise une seconde liste qui contiendra les données de la ligne à lire. Une condition if vérifie que la ligne que l'on s'apprête à lire n'est pas vide, car dans ce cas on ne veut probablement pas l'ajouter au tableau. Si elle n'est pas vide, on itère sur les colonnes, avec l'indice j, en ajoutant l'élément à la liste ligne. Une fois qu'on a lu et ajouté tous les éléments de la ligne, on ajoute la ligne au tableau de résultat final. La structure est donc :

[[$x_{lin=0,col=0}$, $x_{lin=0,col=1}$, $x_{lin=0,col=2}$,...], [$x_{lin=1,col=0}$, $x_{lin=1,col=1}$, $x_{lin=1,col=2}$,...],...]

Recherche d'un élément par nom exact

Exemple : dans la fonction **gestionSacherie.qte_materiaux_sacherie(),** recherche du code postal correspondant au code GCO d'une usine (*num_site*)

```
try:
    cp_site_liv = fonctionsMatrices.recherche_elem(num_site, p.CP_SITES_FLORENTAISE,0,2)
except ValueError:
    cp_site_liv = 44000
    print("problème")
```

Figure 2 Exemple de recherche de nom avec la fonction recherche_elem()

Lorsque l'on recherche un élément dans une colonne d'une matrice pour récupérer la valeur de la même ligne dans une autre colonne, on peut utiliser la fonction **fonctionsMatrices.recherche_elem**(). Son fonctionnement est détaillé dans XXX, mais ici on montre un cas typique de son utilisation.

Lors de son exécution, la fonction utilise la méthode index(), qui renvoie une erreur de type *ValueError* si l'élément demandé n'est pas le bon. Ainsi, si la fonction est appelée avec un élément qu'elle ne trouve pas, l'erreur remontera.

Dans l'exemple, on cherche dans la liste des sites Florentaise, comprenant leur code postal, leur code dans la GCO, et leur nom, le code postal correspondant à un numéro GCO.

La structure en *try/except* est classique : on essaye de faire appel à cette fonction **recherche_elem()**. Cependant, si une erreur en revient, plutôt que de la laisser faire planter le programme en remontant complètement, on « l'attrape » avec « *except ValueError*». Si une erreur de type *ValueError* (qui est le type d'erreur qu'une méthode index() insatisfaite retourne) on attribue un code postal de remplacement, et on signale à l'utilisateur·ice qu'il y a eu un problème.

Recherche d'un élément par nom approximatif

Exemple : la fonction **gestionSacherie.qte_materiaux_sacherie()**, quand on recherche le nom de l'imprimeur (*imprimeur*) dans la liste des *imprimeries*.

```
trouveimp= False
nom_imp_simp= []
for c in imprimeur:
    if c.isalpha():
        nom_imp_simp+=c.lower()
if nom_imp_simp=="" or nom_imp_simp==" " :
    nom_imp_simp="placel"
for imp in imprimeries:
    autre_nom_imp_simp = []
    for c in imp[0]:
        if c.isalpha():
            autre_nom_imp_simp+=c.lower()
    if autre_nom_imp_simp == nom_imp_simp:
        trouveimp = True
        longA = imp[1]
        latA = imp[2]
        break
if not trouveimp:
    fonctionsMatrices.print_log_erreur(
        "Il faut renseigner les coordonnes de l'imprimeur " + imprimeur
                                    +" (réf. "+ref+")", inspect.stack()[0][3])
break       #Une fois trouvée, on arrete de parcourir la liste
```

Figure 3 Exemple de recherche insensible à la casse

Ici, on compare deux noms, mais on est pas sûr·e de la casse. Ainsi, plutôt que de faire appel à **fonctionsMatrices.recherche_elem(),** on parcourt la liste des imprimeries manuellement. On compare alors le nom de l'imprimeur de la liste, représenté par *imp[0]*, à celui de l'imprimeur que l'on recherche.

Mais pour favoriser les chances de trouver, on met les deux en minuscules avec la méthode **lower()**. Également, plutôt que de comparer l'égalité exacte de tous les charactères (ce que l'on ferait avec '=='), on teste la double inclusion. Est-ce que '*imprimeur*' contient '*imp[0]*', ou est-ce que '*imp[0]*' contient '*imprimeur*' ? Dans les deux cas, on considère que la différence est suffisamment minime pour dire que les deux éléments correspondent. On récupère donc les données que l'on voulait dans la matrice imprimeries (ici la latitude et la longitude de l'imprimerie), et on note le marqueur Booleen '*trouvé*' avec True. Il était défini à False avant de parcourir la liste des imprimeries. Enfin, si on a trouvé notre imprimeur, on aura pas besoin de trouver une seconde ligne correspondante. On peut donc arrêter de chercher, avec un '*break*' qui permet de sortir de la boucle for avant la fin. Cela nous permet d'éviter de parcourir inutilement des listes. L'avantage est minime dans ce cas, car la liste est relativement petite, mais il est crucial dans d'autres cas (par exemple la recherche d'une référence de sacherie dans la même fonction).

Enfin, si le marqueur '*trouve*' n'a pas été mis à jour, qu'il est toujours à False, c'est qu'il n'y a pas eu d'élément qui semblait correspondre au nom recherché. Ainsi, on donne des coordonnées par défaut (ici celles de l'usine de St-Mars) pour ne pas faire planter le programme plus tard quand on aura besoin de ces coordonnées, et on le signale à l'utilisateur·ice par une erreur.

Ouverture d'un fichier Excel

Les premières lignes des fonctions qui font appel à de la lecture/écriture de fichier Excel ressemblent généralement à cela :

```
try:
    document = xlrd.open_workbook(p.DIRECTION_FICHIER_SACHERIE)
except FileNotFoundError:
    fonctionsMatrices.print_log_erreur("Le fichier des consos sacherie n'est pas trouvé à l'emplacement "+p.DIRECTION_FICHIER_SACHERIE, inspect.
    sys.exit("Le fichier des consos sacherie n'est pas trouvé à l'emplacement "+p.DIRECTION_FICHIER_SACHERIE)
feuille_bdd = document.sheet_by_index(0)
```

Figure 4 Exemple d'ouverture d'un fichier Excel

Le but est ici de détecter qu'un fichier est manquant, ou alors pas sous le bon nom, avant de l'ouvrir. Essayer d'ouvrir un fichier qui n'est pas disponible fait remonter une erreur de type *FileNotFoundError*. Lorsqu'elle advient, on envoie un message d'erreur, et on quitte le programme en affichant un message d'erreur.

En général, ce ne serait pas bien différent dans l'exécution du programme de ne pas mettre cela. En effet, dans tous les cas, le programme s'arrête, le calcul n'aboutit pas. C'est surtout plus lisible pour l'utilisateur·ice, à qui on donne la raison de l'échec, et l'emplacement qui aurait dû être trouvé.

Dans le cas de la lecture du fichier paramètres, en revanche, cela peut se révéler utile. Ici, si le fichier paramètres n'est pas trouvé, on peut se contenter des paramètres par défaut, et de demander manuellement l'année de calcul à l'utilisateur·ice.

4. Fichiers Données

Les fichiers « données » sont les extractions faites depuis le logiciel de gestion comptable (GCO). Ces fichiers sources sont datés (selon l'année spécifiée lors de l'extraction GCO) afin d'utiliser les bonnes données correspondantes à l'année du calcul du bilan carbone souhaitée, puis classés dans le dossier correspondant à l'année (donc dans Calculateur Carbone/Donnees/20XX/…).

Leur nom et leur contenu est standardisé. Il est donc important de bien les nommer comme suit, et que l'ordre des données qu'ils contiennent soit cohérents avec celui déjà présent[3] :

- **Intrants 20XX.xlsx** : contient l'extraction des achats. En gras sont les colonnes lues par le programme python[4]:
    - **N° Produit** : *entier*.
    - **Dépôt de stock :** *entier*. Numéro de l'usine dans laquelle cet intrant a été livré. Les codes de correspondance sont dans la première feuille des Entrées Manuelles (cf partie XX)
    - **Désignation Produit :** *texte*. Nom de l'intrant. Il faut que les noms des intrants soit cohérent avec celui présenté dans la base de données de leur FE (excepté casse[5])
    - **Référence interne :** *texte*.
    - **N° Client/Fournisseur** : *entier*.
    - Kg : *entier*. Masse livrée, en kilogrammes
    - M3 : *entier*. Volume livré, en mètres cubes
    - Palettes : *entier*. Nombre de palettes livrées (sans unité)
    - **Famille** : *texte*.
    - N°BL : *entier*. Numéro du bon de livraison
    - **Nom Fournisseur** : *texte*.
    - **Code Postal :** *entier*. Code postal du fournisseur
    - **Pays** : *texte*. Pays du fournisseur (format FRA/ALL/BEL/EST/SRI/ITA/ESP/EU)
    - Ville : *texte*. Ville du fournisseur
    - TON : *flottant*. Tonnage

[3] Ce n'est pas l'intitulé des colonnes qui importe, mais leur ordre
[4] Ce qu'il y a dans les colonnes non lues importe donc peu, mais il faut qu'elles soient là
[5] La casse est la présence ou non de majuscule

  - Unité Vente : *texte*. Format KG/T/M3/BB/L/SAC/TON
  - **Unité Achat** : *texte*. Format kg/T/M3/BB/L/SAC/TON/PCE/U/
  - **Quantité Unitaire :** *entier*. Quantité d'achat.
- **Conso sacherie.xlsx** : contient l'extraction des achats de sacherie faite par chaque usine
  - **Dépôt de stock** : *entier*. Numéro de l'usine dans laquelle cet intrant a été livré. Les codes de correspondance sont dans la première feuille des Entrées Manuelles (cf partie XX)
  - Date mouvement : *date*.
  - **Quantité :** *entier*. En nombre de sacs livrés
  - **Référence interne :** *texte*. Référence de sacherie du sac livré
- **Ventes** : contient l'extraction des achats de sacherie faite par chaque usine
  - **N° Bon regroupement :** *entier.*
  - N° Groupement : *entier*.
  - **Sens mouvement** : *texte.*
  - **Montant port revient total:** *flottant.*
  - Montant port acheté total : *flottant* .
  - **Poids total :** *flottant.*
  - Nb palettes **:** *entier.*
  - **Dépôt de stock :** *entier.*
  - **Date liv. client ou arrivée en stock début** : *date*
  - **N° fournisseur :**
  - **N° client :**
  - **Affrètement :** *yes/no*
  - Type Saisie client
  - Type Saisie fournisseur
  - Transporteur
  - **CP liv. :** *entier.* Code postal de la livraison
  - Mode livraison : *texte.*
  - **Type d'élément :** *texte.* CLA/CLP/CTN/INT…
  - Plateforme nationale : *entier*.
  - Plateforme régionale : *entier.*
  - N° B.L client : *entier.*
  - N° B.L fournisseur : *entier*.
  - **Nb palettes :** *entier*.
  - Nb colis : *entier*.
  - N° facture transporteur : *entier*.
  - Année : *entier*.
  - Mois : *entier*.
  - Nom Client : *texte*.
  - Nom Fournisseur : *texte*.
  - N° Client : *entier*.

5. Fichiers Sources

Les fichiers sources sont les entrées indépendantes de l'année de calcul du bilan carbone. Ils sont lus par le programme, et fournissent soit des bases de données pour le programme, soit des paramètres.

Le nom des fichiers est standardisé, ne pas les changer. L'ordre des feuilles dans ces fichiers Excel est important ! Ne pas les mélanger ! Le nom des feuilles, en revanche, peut être changé.

a. BDDgeoloc.xlsx

Ce fichier est lu une seule fois en début du programme pour y récupérer la liste des codes postaux, leurs coordonnées GPS en radians, ainsi que le facteur de conversion des distances entre vol d'oiseau et route. Tout cela se trouve dans le premier onglet (première feuille). Le second n'est jamais ouvert ou lu, il est simplement destiné à se familiariser manuellement avec le système de couplage entre codes postaux et coordonnées GPS.

Cette base de données des codes postaux est récupérée depuis le site *data.gouv.fr*[6]. Les données de la base de données téléchargée fournissent les coordonnées GPS en degré. Cependant, la formule de calcul des distances utilisant les radians, elles doivent être converties en étant multipliées par $\pi/180$. Cette conversion est faite en dure (les données correspondant aux colonnes en radians ne sont pas des formules) afin de ne pas avoir à les recalculer systématiquement, étant donné que cette BDD est inchangeante.

Les noms des communes des départements d'Outre-Mer ont été complétées manuellement, car ne sont pas comprises dans la base de données initialement. Toutes mes excuses si des erreurs s'y cachent.

Les colonnes sont les suivantes :

- Nom_Commune
- **Code Postal**
- Ligne_5
- LibellÃ©_d_acheminement
- Longitude (en degrés)
- Latitude (en degrés)
- **Longitude radians**
- **Latitude radians**

Enfin, la cellule L2 contient le facteur de conversion entre vol d'oiseau et distance de route.

b. Entrees Manuelles

Les entrées manuelles se séparent en plusieurs onglets dont le nom n'est pas important mais l'ordre est à respecter.

La première ligne, épinglée, est un comptage faisant correspondre les colonnes à leurs indices (A=0, B=1, C=2, etc.). Cela sert uniquement à se repérer plus aisément dans la lecture des cellules par le programme.

i. Données Entreprises

Les données entreprises servent à gérer tout ce qui est général, les données ne nécessitant pas de détail (consos électriques/fuel) ou sur la taille de l'entreprise. Les données sont entrées pour chaque site pour chaque année dans plusieurs tableaux

Attention ! Les coordonnées et la taille des tableaux est cruciale dans le code. Si cela change, il faut penser à aller corriger les fonctions de lectures dans le programme.

[6] https://www.data.gouv.fr/fr/datasets/base-officielle-des-codes-postaux/

Chiffre d'affaire

Le tableau du chiffre d'affaire n'est pas décliné par site, cela n'aurait pas de sens, il ne comporte donc qu'une seule ligne et est assez immédiat à comprendre. Renseigner des entiers ou des nombres à virgule (flottants).

Nombre de Salarié·es

Par site et par année, renseigner le nombre de salarié·es par site. Si l'usine n'était pas dans le groupe à une année, renseigner 0. Renseigner des entiers.

Production en m3

Production de chaque site chaque année. Si l'usine n'était pas dans le groupe à une année, renseigner 0. Renseigner des entiers ou flottants.

Consommation Fuel

Consommation par les machines ou engins de chaque site chaque année. Si l'usine n'était pas dans le groupe à une année, renseigner 0. Renseigner des entiers ou flottants. L'unité est en L. type de fuel ?

Consommation Electrique

Consommation électrique externe du site. Si l'usine n'était pas dans le groupe à une année, renseigner 0. Renseigner des entiers ou flottants, en kWh.

Pour le cas des usines produisant une partie de leur électricité (renouvelable, photovoltaïque)

On ne considère que le facteur d'émission de l'énergie renouvelable pour l'énergie consommée par l'usine. Si l'usine a également consommé de l'énergie extérieure, on compte cette énergie avec un mix énergétique conventionnel. Enfin, si par surplus de production, le site revend son énergie renouvelable, ce surplus n'est pas pris en compte. Il ne pourra être considéré comme un évitement, ou une compensation.

ii. Facteurs Emissions MP

Cette feuille sert à considérer le bilan carbone des intrants de l'usine.

Il y a plusieurs tableaux :

FE par famille

Donne les Facteurs d'émissions (FE) en kg équivalent CO_2 par tonne de matière ($kg_{eq}CO_2/t$). Les colonnes sont donc les suivantes :

- **ID** : *entier*.
- **Famille** : *text.* Nom de la famille de bilan carbone. Doit correspondre en noms à la colonne « Famille Bilan Carbone » du tableaux « Familles, et teneurs en azote »
- **FE** : *float*. Facteur d'émission.

Famille, et teneur en azote

Ce tableau permet de faire le lien entre une matière première et sa famille de bilan carbone, afin de lui attribuer dans le programme un facteur d'émission en le mettant en lien (par le nom de la famille) avec le tableau précédent). Pour les engrais, cela permet également de renseigner leur teneur en azote.

- **ID**[7] : *entier*.

[7] Les ID de ces tableaux ne sont pas des clés de correspondance entre les tableaux ! Ce sont simplement des numérotations des lignes, et ne servent à aucun moment à une identification.

- N°MP : *entier*. Numéro de la matière première
- **Désignation MP** : *texte*. Nom de l'intrant. Les intitulés doivent correspondre à ceux présents dans la liste des achats.
- **Réf interne** : *texte*.
- **Familles Florentaise** : *texte.* Ce n'est pas cette colonne qui est utilisée pour mettre en lien l'intrant et son FE ou sa masse volumique
- **Familles Bilan Carbone**: *text.* Nom de la famille de bilan carbone. Doit correspondre en noms à la colonne « Famille Bilan Carbone » du tableaux « FE par famille »
- **N% des engrais** : *float*. Teneur en azote, si c'est un engrais. Si l'intrant n'est pas un engrais ; ne rien renseigner. Ou renseigner 0. Peu importe. La vie n'a que peu de sens finalement.

Masses volumiques

Ce tableau permet de faire le lien entre une matière première et masse volumique. Cela permet de faire les conversion volume/masse dans le programme. Pour certains intrants aucune masse volumique n'est renseignée, c'est alors qu'elle n'est pas nécessaire, et que cet intrant est toujours acheté au poids[8]. Si une masse volumique nécessaire est manquante, le programme utilisera une valeur de 1 t/m3.

- **ID** : *entier*.
- **Désignation MP** : *texte*. Nom de l'intrant. Les intitulés doivent correspondre à ceux présents dans la liste des achats.
- **Masse volumique t/m3**: *flottant*.

Engrais et azotes

Ce tableau donne l'émission de GES en kg équivalent CO2 par tonne d'azote selon le type d'engrais. Le nom du type d'engrais doit correspondre à ceux indiqués dans la colonne « Famille Bilan Carbone » du second tableau de la feuille.

- **ID** : *entier*.
- **Famille** : *texte*. Nom de famille de l'azote. Les intitulés doivent correspondre à ceux présents dans la colonne des familles de bilan carbone des produits.
- **FE (kgCO2/t azote):** *float*. Facteur d'émission.

iii. Autres FE

iv. Import Etranger Maritime et Ter

Pour l'import terrestre, il s'agit seulement d'une base de données de coordonnées GPS des différents fournisseurs. C'est moins immédiat avec le transport maritime, qui fonctionne par « chaine » de livraison :

- Le premier tableau est une liste désordonnée des lieux par lesquels passent le trafic d'intrants :
  - **ID** : *entier*. Le numéro du lieu (propre à cette feuille) : Ce numéro sert à relier les chaines de livraison, il est donc très important
  - **Lieu** : *texte*. Une chaine de texte qui contient les informations par lesquelles on est susceptibles de faire appel à ce lieu (pays, abréviation du pays, ville…). *Ex : EST Estonie Estonia Usine Tourbe*.
  - **Nom fournisseur** : *texte.* Le nom du fournisseur de MP tel qu'il est écrit dans la GCO pour les achats de cette matière première.
  - **Coordonnées GPS** : *flottants*. En degrés puis convertis en radians.

[8] On utilise indifféremment masse et poids

  - **Note** : Le code couleur usine/port départ/port arrivée n'est utile que pour la lisibilité de l'utilisateur·ice, et n'est pas exploité par le programme
- Le second tableau donne les liens de livraison maritime :
  - Lieux : *texte*. N'est utile que pour l'utilisateur·ice. Donne une information rapide sur le produit dont il s'agit généralement, et son pays de provenance.
  - **Usine départ** : *entier*. Numéro de l'usine d'où est produite cet intrant, tel que référencé dans le premier tableau de cette feuille
  - **Port départ** : *entier*. Numéro du port d'expédition de cet intrant à la sortie de l'usine départ, tel que référencé dans le premier tableau de cette feuille
  - **Port arrivée** : *entier*. Numéro du port de réception de cet intrant en France, tel que référencé dans le premier tableau de cette feuille
  - **Distance bateau** : *entier.* Distance en km parcourue par le bateau effectuant ces livraisons. Pour les calculer, il faut passer soit par Google Earth et l'outil de mesure des distances, soit par un site de calcul d'itinéraires maritime[9].
- Le troisième est une base de données des fournisseurs terrestres européens.
  - Le fonctionnement des colonnes est identique à celui du premier tableau de la feuille

v. Export Etranger Maritime et Ter

Le fonctionnement de cette feuille est sensiblement similaire à celui de la feuille précédente pour les imports, notamment pour le maritime. Pour le maritime, l'exception est qu'il n'y a, dans l'autre sens, pas d'usine d'arrivée : on considère que les produits sont livrés au port.

Se référer au paragraphe précédent, Import Etranger Maritime et Ter, pour le détail du fonctionnement.

vi. Sacherie

Cette feuille donne les caractéristiques de la sacherie de chaque produit depuis 2012, ainsi que les coordonnées des imprimeurs qui les fabriquent.

En cellule D6 on renseigne la densité du PEBD, utilisée pour calculer la masse des sachets.

Le fonctionnement de la liste est le suivant :

- A gauche du tableau on a un indice pour l'utilisateur·ice des années concernée par le tableau.
- **Annee** :*entier*. Donne l'année de fin de mise à jour de cette partie de la liste
- Gamme : *texte*. Gamme de commercialisation du sac. Peut être utilisée dans la colonne de recyclage (voir un peu plus bas)
- Référence : *texte*. Référence du produit semi-fini. *Ex : ALDI50E.*
- **Sacherie :** *texte*. Référence de sacherie. *Ex : ALDI50S.*
- Désignation produit : *texte*. Nom du produit.
- Dimensions **du sac :** *entiers.* Laize, Dév et Epais, respectivement en mm, mm, et µm.
- **Type :** *texte*. Composition du sac : PEBD ou Papier
- **Taux de recyclage :** *flottant.* Pour l'instant, seulement s'il s'agit de PEBD (les sacs papier n'étant en pratique pas utilisés). Plutôt que de vérifier les milliers de ligne à la main, le plus simple est de remplir cette colonne en une formule, qui prend en compte la gamme et l'année.

[9] Cette seconde méthode est la plus pratique. Le site suivant permet de faire cela : https://www.searates.com/fr/services/distances-time/

Ex : =SI(ET(C10="Botanic";B10>2017);0,3;_____)[10]. Remplacer ensuite les ___ par une autre formule =SI pour d'autres cas où le sac est en recyclé.

- **Poids plastique :** *flottant.* Calculé en grammes à partir des trois colonnes de dimensions du sac et de la densité de PEBD. Cette cellule est remplie uniquement si le sac est de type PEBD.
- **Poids papier :** *flottant.* Pour l'instant n'est pas remplie, mais se remplirait de la même manière que la colonne précédente, uniquement si le sac est de type PEBD. Note : ce système à double colonne pourrait permettre, à quelques ajustements près dans le programme, de considérer des sacs de composition mixte PEBD/plastique
- **Imprimeur** : *texte.* Nom de l'imprimeur. Doit être écrit de la même façon que dans le tableau d'après (à droite dans la feuille)

Note : Ce tableau est une concaténation chronologique de tous les tableaux de sacherie utilisés par le service marketing. Après quelques années, un nouveau tableau est fait, et l'ancien n'est plus mis à jour. Cependant, ajouter le nouveau tableau à la fin de celui-ci est important. En effet, le programme peut être amené à calculer le bilan carbone d'années antérieures, où certaines références de sacherie ont évolué. Le programme parcourt ce tableau du bas vers le haut, et s'arrête à la première référence si l'année correspond : cela permet de s'assurer d'avoir une liste comportant les références les plus récentes possibles.

Le second tableau recense simplement les coordonnées GPS des imprimeurs, en radians. Le code postal peut être utilisé pour les retrouver avec une formule dans la BDDgeoloc. Sinon (notamment dans le cas d'un imprimeur à l'étranger), il faut entrer les coordonnées manuellement, depuis un outil comme GoogleMaps ou OpenStreetMaps.

6. Traitement

Le traitement est centré autour de l'exécution du script contenu dans main.py, qui appelle chacune des fonctions. Le détail ligne par ligne des fonctions ne sera pas donné ici, se référer aux commentaires du code pour cela.

Pour le détail des contenus des tableaux :

Souvent, une liste en contient d'autres. Pour expliciter cela, le code typographique est le suivant. En gras sont les variables inscrites dans le programme. Elles peuvent donc être retrouvées (sauf exceptions) dans l'explorateur de variables de l'environnement de travail.En italique et souligné sont des éléments du tableau correspondant à un autre tableau imbriqué dans le premier. Ce sois tableau est donc explicité après. Enfin, en fonte normale sont les intitulés des valeurs comprises dans le tableau.

a. main.py

Au début de ce script se trouvent les imports des autres fonctions du programme, et, si besoin, de modules extérieurs.

[10] Signification : Si la référence correspond à du Botanic ET que l'année est après 2017, alors le taux de recyclage est de 0.3. Sinon, ___.

Ensuite se trouvent les entrées « bas niveau » de l'utilisateur·ices. Ce sont des données qui ne devraient pas changer souvent, comme les chemins d'écriture et de lecture des fichiers, et quelques autres paramètres (moyens de transports par exemple).

Ensuite il s'agit de lire des fichiers utiles généralement dans tout le programme.

i. Lecture des fichiers généraux

La fonction gestionExport.lireExport() est utilisée, pour lire dans Entrees Manuelles le fichier d'export. Les données sont stockées sous la forme de deux variables :

Tableau 1 Variable bdd_exp_terre

Variable	0	1	2	3	4
bdd_exp_terre					

Et :

Tableau 2 Variable bdd_exp_mari

Variable **bdd_exp_mari**	0	1	2
*Chaine_*	*Port_depart*	*Port_arrivee*	Distance de bateau en km

Variable	0	1	2
Port (départ ou arrivée)	Indice	*Ville*	*Pays*

On lit ensuite dans la première feuille des entrées manuelles les caractéristiques des sites florentaise : leur code GCO, code postal, et année d'entrée dans le groupe. Ces informations sont stockées sous la forme suivante

Tableau 3 Variable cp_sites_flor

Variable **cp_sites_flor**	0	1	2	3
Usine	Code GCO	Nom	Code Postal	Année d'entrée

Ce tableau est particulièrement utile pour calculer les distances de fret amont ou aval en faisant correspondre le code GCO au code postal.

ii. Gestion des intrants

Ce qui suit concerne les matières premières, le calcul de leur fret et de leur bilan carbone de fabrication.

Lecture des achats

On lit premièrement la liste des achats grâce à la fonction gestionIntrants.lireBDDMP(). Le tableau **listeAchats** qui en résulte a pour lignes :

Tableau 4 Variable listeAchats

Variable **listeAchats**	0	1	2	3	4	5	6	7	8	9	10
Ligne d'achat	Référence de la MP	Nom	Référence interne	N° client	Famille	Nom fournisse	CP fournisse	CP usine arrivée	Pays du fournisse	Quantité achat	Unité achat
Type	Int	String	String	Int	String	String	Float[11]	Int	String	Float	String

On extrait ensuite une liste unique des noms des intrants, en extrayant la colonne 1 du tableau **listeAchats**, pour ensuite en tirer une liste avec des éléments uniques. La liste en résultant, **listeMP**, est simplement une liste de chaines de caractères comportant les noms des intrants.

Calcul des distances de fret

On calcule ensuite le fret des matières premières en faisant appel à fonctionsFret.calc_fret_par_MP(). Cette fonction renvoie deux tableaux, **fret**[12] et **resultat_usine_MP** :

Tableau 5 Variable fret1 (incomplete)

Variable **fret**	0	1	2	3	4	5
Ligne de MP	Nom MP	Somme du fret router (km)	Somme du fret naval (km)	Moyenne de fret routier (km)	Moyenne de fret naval (km)	*Sous-tableau de quantités*
Type	String	Float	Float	Float	Float	List

Et

Sous-tableau des quantités	0		1	
Ligne unique	Sous-tableau des volumes		Sous-tableau des masses	
Type	List		List	
Element	« M3 »	Volume	« KG »	Masse
Type	String	Float	String	Float

[11] Il arrive que le code postal ne soit pas un code postal Français, ou bien pas ou mal renseigné. Dans ces cas, cet élément sera un *string,* dont l'exception sera gérée plus tard

[12] Attentions, d'autres fonctions modifient ce tableau, et rajoutent des colonnes

Tableau 6 Variable resultat_usine_MP

Variable **resultat_usine_MP**	0	1	2		3	
Ligne de MP	Nom de la MP	*Sous tableau des quantités par site*	Sous-tableau des quantités cumulées		Sous-tableau du fret cumulé	
Type	String	List	List		List	
Sous tableau	-	Cf tableau suivant	Qté volume cumulé (m3)	Qté masse cumulé (kg)	Fret routier cumulé de la MP (km)	Fret bateau cumulé de la MP (km)
Type	-	-	Float	Float	Float	Float

Les lignes du *Sous tableau des quantités par site* sont les suivantes, pour chaque usine. Chaque case contient une liste de deux éléments :

Variable *Sous tableau des quantités par site*	0		1		2	
Ligne par usine	Tableau identification usine		Tableau des parts de quantité		Tableau des parts de fret	
Type	List		List		List	
Sous-sous tableau	Code postal	Nom de l'usine	Part du volume	Part de la masse	Part du trafic routier de la MP	Part du trafic naval de la MP
Type	Int	String	Float	Float	Float	Float

Cela demande quelques éclaircissements :

La fonction calc_fret_par_MP() a été structurée de manière à ressortir d'abord un résultat pour calculer les émissions propres à chaque matières premières, et non par site. Ainsi, pour avoir le bilan carbone de chaque site, comme il était présenté dans les bilans carbones antérieurs à 2019, il faut greffer sur la fonction une manière de compteur quelle quantité de la matière est livrée à quel site, et quel fret cela a engendré. Pour stocker et compter de manière synthétique, on procède par des part. Ce a pour défaut de créer des erreurs d'arrondi, mais qui ne dépassent à priori pas l'ordre du kg de matière première.

Ainsi, comme la quantité (masse et volume) et le fret (route et maritime) cumulé de chaque matière sont comptés dans les tableaux en colonnes 2 et 3 du tableau **resultat_usine_MP**, on pourra ensuite retrouver, pour chaque usine et pour chaque matière première, de combien de quantité de cette matière première cette usine est-elle cliente, et de combien de fret cela a engendré.

Cela peut sembler tarabiscoté de faire ainsi, mais c'est un parti pris, pour ne pas avoir à parcourir une seconde fois le tableau des lignes d'achat. On aurait aussi très bien pu parcourir le tableau des achats selon les usines et non selon les matières premières, mais j'y ai pensé trop tard !

Cette méthode pose cependant une hypothèse sur laquelle on reviendra en explicitant la fonction fonctionsFret. renverserMP_usine_fret().

Facteurs d'émission du fret

Ensuite, pour quelque chose de plus léger, on va récupérer les variables **FE_route** et **FE_bateau** dans le fichier des entrées manuelles. Ce sont des flottants, qui expriment des $kgCO2.t^{-1}.km^{-1}$.

___

On procède ensuite à compléter le tableau de **fret** avec les facteurs d'émission que l'on vient de récupérer. Cela donne le tableau suivant[13] :

[13] Attention ; ce n'est toujours pas la version finale du tableau **fret**

Tableau 7 Variable fret2 (incomplete)

Variable **fret**	0	1	2	3	4	5	6	7	8	9
Ligne de MP	Nom MP	Somme du fret router (km)	Somme du fret naval (km)	Moyenne de fret routier (km)	Moyenne de fret naval (km)	*Sous-tableau de quantités*	FE route cumulé (kgCO2e/t)	FE bateau cumulé(kgCO2e/t)	FE route moyen(kgCO2e/t)	FE bateau moyen(kgCO2e/t)
Type	String	Float	Float	Float	Float	List	Float	Float	Float	Float

Sous-tableau des quantités	0		1	
Ligne unique	Sous-tableau des volumes		Sous-tableau des masses	
Type	List		List	
Element	« M3 »	Volume	« KG »	Masse
Type	String	Float	String	Float

Pourquoi se retrouve-t-on avec quatre facteurs d'émission ?

Cela est la conséquence du fait qu'à partir de maintenant, il faut considérer la divergence d'approche, entre le bilan carbone de Florentaise, et le bilan carbone par produit. Dans la prévision du second, on rajoute dorénavant deux colonnes. Les colonnes des FE route cumulé et FE bateau cumulé (6 et 7)

donneront, lorsqu'on multipliera par la masse de chaque intrant, le fret amont TOTAL que cette matière première a engendré cette année. Ces deux colonnes donneront donc des chiffres intéressant pour le bilan carbone, et les sommer sur chaque matière première donnera le bilan carbone du fret amont total de Florentaise. Le détail par usine de ce fret amont total est déjà pris en charge par le tableau **resultat_usine_MP** (cf page 18, *Calcul* des distances de fret). Les deux colonnes de FE route moyen et FE bateau moyen (8 et 9) donnent quant à elles des résultats qui seront intéressant d'avoir pour un calculateur de bilan carbone des produits. En effet, cela donne en moyenne quelles émissions le fret d'une tonne de matières premières a engendré.

Fabrication des intrants

On procède ensuite à une lecture de la feuille concernant les caractéristiques des MP dans l'Excel des Entrees Manuelles. Les données récupérées sont mises sous des formes assez explicites :

Tableau 8 Variable FE_familles

Variable **FE_familles**	0	1	2
Ligne par famille	Indice de la famille	Nom de la famille Bilan Carbone	Facteur d'émission (kgCO2/t)
Type	Float	String	Float

Tableau 9 Variable MP_familles_N

Variable **MP_familles_N**	0	1	2	3	4	5	5
Ligne par MP	Indice de la MP	N°MP	Nom de la MP	Réf interne	Famille MP	Famille de bilan carbone	Teneur en azote (%N des engrais)
Type	Float	Int	String	String	String	String	Float

Tableau 10 Variable masse_vol_MP

Variable **masse_vol_MP**	0	1	2
Ligne par MP	Indice de la MP	Nom de la MP	Masse volumique (t/m3)
Type	Float	String	Float

Tableau 11 Variable FE_engrais

Variable **FE_engrais**	0	1	2
Ligne par famille d'engrais	Indice de la famille d'engrais	Nom de la famille d'engrais	FE (kgCO2/t azote)
Type	Float	String	Float

___

La

Variable **fret**	0	1	2	3	4	5	6	7	8	9	10	11	12
Ligne de MP	Nom MP	Somme du fret router (km)	Somme du fret naval (km)	Moyenne de fret routier (km)	Moyenne de fret naval (km)	*Sous-tableau de quantités*	FE route cumulé (kgCO2e/t)	FE bateau cumulé(kgCO2e/t)	FE route moyen(kgCO2e/t)	FE bateau moyen(kgCO2e/t)	FE fabrication (kgCO2e/t)	FE total avec fret cumulé(kgCO2e/t)	FE total avec fret moyen(kgCO2e/t)
Type	String	Float	Float	Float	Float	List	Float	Float	Float	Float	Float	Float	Float

fonction gestionIntrants.associerMPetFE_fab() se charge ensuite d'associer les trois tableaux (**FE_familles** et **MP_familles_N** et **FE_engrais**) en faisant correspondre à chaque intrant un facteur d'émission. Cela renvoie le tableau **MP_et_FE** qui est assez immédiat : chaque ligne contient le nom de la matière première et son facteur d'émission en kgCO2/t.

Compilation fret et fabrication

On associe ensuite le tableau **fret** au tableau **MP_et_FE**, pour obtenir le tableau **fret** suivant :

Tableau 12 Variable fret3 (incomplete)

De la même manière qu'expliqué précédemment, la colonne 11 est davantage destinée au calcul du bilan carbone total, et la colonne 12 au bilan carbone par produit

___

Enfin, la fonction gestionIntrants.calc_quantite_et_BC() se charge de calculer le tonnage de chaque matière première en utilisant la liste **masse_vol_MP** pour convertir les m^3 en tonnes. Ensuite elle multiplie ce tonnage par les facteurs d'émission des colonnes 11 et 12, pour aboutir au tableau de **fret** ci-après.

Au passage, elle ajoute la masse volumique au tableau fret en colonne 13 pour faciliter le calcul du bilan carbone des produits.

Ce tableau est ensuite tracé (à l'exception de la colonne 5) dans une feuille Excel de résultats par la fonction affichageResultats.sauveFret().

Tableau 13 Variable fret finale

Fret par usine

Pour entrer dans les catégories, on souhaite également calculer le fret amont par usine, et selon le fret terrestre ou maritime. On reprend donc le tableau **resultat_usine_MP**, et sa structure un peu compliquée, afin de le retourner comme une chaussette ; les éléments du fond devenant les entêtes.

Variable **frat**	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Ligne de MP	Nom MP	Somme du fret router (km)	Somme du fret naval (km)	Moyenne de fret routier (km)	Moyenne de fret naval (km)	Sous-tableau de quantités	FE route cumulé (kgCO2e/t)	FE bateau cumulé(kgCO2e/t)	FE route moyen(kgCO2e/t)	FE bateau moyen(kgCO2e/t)	FE fabrication (kgCO2e/t)	FE total avec fret cumulé(kgCO2e/t)	FE total avec fret moyen(kgCO2e/t)	Masse totale achetée (t)	Masse volumique (t/m3)	Emissions totales de matière cumulées (ktCO2e)	Emissions totales matière moyen (tCO2e)
Type	String	Float	Float	Float	Float	List	Float	Float	Float	Float	Float	Float	Float	Float	Float	Float	Float

C'est la fonction fonctionsFret.renverserMP_usine_fret() qui effectue cela. Son fonctionnement sera détaillé plus loin, mais le tout est qu'elle renvoie le tableau **usines_et_fret**, dont la structure est comme suit :

Tableau 14 Variable usines_et_fret

Variable **usines_et_fret**	0		1	2	3	4	5
Ligne par usine	Identification de l'usine		Masse (t)	Fret route (km)	Fret bateau (km)	BC route (kgCO2e)	BC bateau (kgCO2e)
Type	List		Float	Float	Float	Float	Float
Identification de l'usine	CP usine	Nom usine					
Type	Int	String					

iii. Gestion des emballages et de la sacherie

On récupère dans un premier temps une liste des références de sacherie pour les produits semi-finis, et la consommation de chaque site de cette référence. La fonction gestionSacherie.lireConsoSacherie() retourne le tableau suivant **refs_sacheries** :

Tableau 15 Variable refs_sacheries

Variable **refs_sacheries**	0	1									
Ligne par usine	Nom de la sacherie	Consommation par site									
Type	String	List									
Consommation par site		Site1		Site 2		Site3		Site4		…	
Type		List		List		List		List		List	
Elements		Code site S1	Conso S1 (u)[14]	Code site S2	Conso S2 (u)	Code site S3	Conso S3 (u)	Code site S4	Conso S4 (u)	Code site S_	Conso S_ (u)
Type		Int	int	Int	int	Int	int	Int	int	Int	int

La fonction gestionSacherie.qte_materiaux_sacherie() utilise ensuite cette matrice, ainsi que le tableau des références de sacherie de l'Excel des entrées manuelles, pour produire deux tableau : **conso_materiaux_par_site** et **fret_sacherie**.

Pour chaque site, le premier est structuré comme cela :

Tableau 16 Variable conso_materiaux_par_site

Variable **conso_materiaux_par_site**	0	1						2
Ligne par usine	Numéro CGO du site	Consommation par matériau						Fret (distance totale parcourue par les emballages) en km
Type	Int	List						Float
Consommation par matériau		Conso PEBD		Conso PEBDr		Conso papier		
Type		List		List		List		
Eléments		« PEBD »	Conso PEBD (t)[15]	« PEBDr »	Conso PEBDr (t)	« Papier »	Conso Papier (t)	
Type		String	int	String	int	String	int	

[14] Consommation en nombre de sacs (unité)
[15] Consommation en tonnes de matière

L'utilisation de ce tableau est de pouvoir calculer le bilan carbone des matériaux qui composent la sacherie, à savoir le plastique (recyclé ou non), et le papier, et ce pour chaque usine.

La seconde liste **fret_sacherie** ne compose que deux éléments : le premier est la distance cumulée du transport des sacs en plastiques, en km, et la seconde est la masse cumulée des sacs plastiques. En fait, c'est sensiblement comme si on avait fait la somme de la colonne 2 du tableau précédent, et des secondes sous-colonnes de la colonne 1 de ce même tableau pour la masse.

En faisant cela, une hypothèse implicite est posée, celle que les camions qui transportent la sacherie sont remplis au maximum. Cette hypothèse est certainement assez grosse, et conduit à sous-estimer les émissions réelles. Cependant : la réalité n'est probablement pas que ces camions sont à partiellement vides. Ils sont peut-être remplis avec d'autres éléments de sacherie, auquel cas leur fret sera comptabilisé plus tard[16]. Sinon, si les camions sont remplis d'autres livraisons pour d'autres entreprises clientes du même imprimeur. Et dans ce cas, le périmètre reste restreint à l'activité de Florentaise, et n'empiète pas sur le périmètre de ces autres entreprises.

C'est finalement une question d'approche, et son contraire se défend aussi. On pourrait considérer qu'à partir du moment où Florentaise participe à l'affrètement d'un camion, les émissions de ce camion peuvent être comptabilisées pour Florentaise. Cela serait plus proche des émissions réelles ; l'impact qu'à eu cette livraison n'est pas une fraction de l'impact de ce camion, fraction dépendant de la charge due à Florentaise. Ici, on reste dans le périmètre de responsabilité de Florentaise.

On ne considèrera pas la même approche par exemple sur l'export. Florentaise a la responsabilité de la logistique, du remplissage, du mode de transport, pour ses ventes. Ici, il en va de la responsabilité de l'imprimeur de gérer autrement (si le bilan carbone s'en trouve amélioré) la logistique de ses exportations.

La question, en outre, serait plus difficile à trancher si l'on fonctionnait avec un facteur d'émission en $kgCO_{2e}$/camion/km, qui est également disponible dans la base ADEME.

b. GestionIntrants.py et fonctionsFret.py

Imports: xlrd, inspect, logging, math, inspect

On regroupe maintenant les deux fichiers car ils traitent tous les deux des matières premières en amont de l'usine. Les deux modules s'échangeant des informations, cela ne faisait pas trop de sens de les analyser séparément.

i. Fonction lireBDDMP() (gestionIntrants.py)

Dans gestionIntrants.py, la première chose qui est faite est la lecture de la base de données des achats, par la fonction **lireBDDMP()**. Elle ne prend pas d'argument en entrée et ressort une liste qui correspond à **listeAchats** (cf Lecture des achats, p.16). Cette fonction fait cependant appel aux variables globales MASSE_PAR_PALETTE et VOLUME_PAR_BIGBAG, définies dans parametres.py. Ces variables ne sont utilisées qu'ici, mais elles sont définies comme des variables globales afin d'être plus facilement trouvables, s'il l'on doit les changer. Les autres variables globales utilisées sont DIRECTION_FICHIER_INTRANTS et CP_SITES_FLORENTAISE.

[16] Finalement, cela vaut mieux ainsi pour ne pas compter en double le fret des plastiques hors sacherie.

Premièrement, la fonction ouvre le fichier Intrants (localisé par la variable globale DIRECTION_FICHIER_INTRANTS) avec le module xlrd. A priori, ce fichier Excel ne contient qu'une feuille, sinon il est seulement important que la feuille qui nous intéresse soit placée en première position, car elle est ouverte à la ligne 2 par *document.sheet_by_index(0)*.

On enregistre le nombre de lignes du fichier pour pouvoir le parcourir avec une boucle for. Après avoir initialisé la matrice result qui contiendra le résultat de la fonction avec les entêtes, on parcourt ligne par ligne le fichier, en commençant à l'indice 1 pour éviter les entêtes du tableau.

A chaque ligne, on lit l'id de la matière première, son nom, la réf, le numéro de dépôt où la livraison arrive, le n° client/fournisseur, la famille (de la matière première, pas du fournisseur), le nom du fournisseur, et le code postal et le pays d'où l'intrant part. On lit ensuite l'unité, puis on lira la quantité.

On distingue déjà l'unité, pour savoir si l'on a un volume ou une masse. Les masses peuvent être représentées par kg, t (ou ton ou tonne(s)), ou une palette (qui sera dans ce cas là de masse définie par MASSE_PAR_PALETTE). Les volumes, eux, sont représentés par m3, l, bigbag (ou bb, et autres variantes) ou par des sacs.

On reconnaît chaque unité en l'identifiant à une chaine de charactères et en couvrant le plus de possibilités possibles. Pour cela, on rend la comparaison insensible à la casse (avec la méthode lower()[17] par exemple) et l'on ne cherche pas à savoir l'égalité stricte mais l'inclusion : `if "m3" in unit_vendue.lower():`. On convertit $1m^3$=1000l et 1t = 1000kg. Ainsi, tout est ramené soit à des kilogrammes, soit à des mètres cubes.

Cependant, dans le cas d'un sac, il peut se poser un problème. Lors de l'achat d'une matière première intitulée par exemple « BOTANIC POUZZOLANE 7/15 15L », on sait combien de sac on achète, mais pas quelle quantité cela représente. Il faut donc aller chercher cette quantité dans le nom du produit (dans l'exemple 15L)

La fonction qui se charge de cela est gestionpb_volume_sac(), elle est détaillée dans le paragraphe suivant.

Enfin, à la fin de la boucle, toujours pour chaque matière première, on essaye de faire appel à la liste des sites de Florentaise sur lesquels on travaille. Avec la fonction recherche_elem(), qui est dans fonctionsMatrices.py, on essaye d'identifier le numéro de dépôt d'arrivée que l'on a identifié dans la liste, et de voir à quel code postal de dépôt il correspond.

Si on ne le trouve pas, on signale une erreur et on le remplace par le code postal de l'usine de St-Mars.

On a donc une liste avec chacun des achats tirée de données brutes, que l'on va traiter dans le iii).

ii. Fonction gestionpb_volume_sac() (gestionIntrants.py)

Cette fonction reçoit uniquement en entrée le nom que l'on souhaite analyser. Elle renvoi un couple d'un float et str, composé de la quantité et de l'unité (soit « m3 » soit « t »).

Il y a d'abord une série d'exception qui peut être complétée à l'avenir. Cette liste sert à gérer les exceptions que la fonction ne saura pas traiter correctement. Par exemple, dans la première exception « TRUFFAUT POUZZOLANE 15L 72 SACS » on risque de ne pas détecter que l'on parle de 72 sacs de 15L et non d'un seul sac de 15L. Les autres exceptions concernent d'autres sacs de matière qui ne contiennent pas d'indices dans leur nom.

[17] Permet de mettre une chaine de charactères en minuscules

```
if nom=="TRUFFAUT POUZZOLANE 15L 72 SACS":
    return 15*72/1000,"m3"
elif nom=="COPODECOR ECORCES DE PIN 10/25 60":
    return 60/1000, "m3"
elif nom =="CERAMIQUE PPC GREEN GRADE":
    return 22.7, "kg"     #Au pif
elif nom =="COPOCAO COQUES DE CACAO":
    return 120/1000,"m3"      #Au pif aussi
elif nom=="Rajouter ici les noms et quantités manuellement":
    return 42,"unite"
```

Figure 5 Exceptions dans la fonction gestionpb_volume_sac

Après ces exceptions, si le nom à rechercher n'en faisait pas partie, on analyse la chaine de caractères.

On parcourt la chaine de caractères un à un (première boucle while). Si un caractère est un chiffre, on entre dans une seconde boucle while qui examine les caractères qui suivront : tant que ce sont des chiffres, on les ajoute à une chaine de caractères « chiffres ».

Quand on sort de cette seconde boucle while, on est donc soit à la fin du nom de l'intrant (auquel cas on n'a rien trouvé), soit sur un caractère qui n'est pas un chiffre. On s'intéresse donc à ce caractère :

- Si c'est un « L » ou « l », et si la liste des chiffres détectés n'est pas vide, on a à faire à un volume en litres. Ainsi on renvoie la quantité divisée par mille et l'indication « m3 ». Pour avoir la quantité, on reprend la liste de chiffres qui précédait le « L », que l'on joint en une chaine de chiffres avec la méthode join() puis que l'on transforme en entier avec int()
- Sinon, si la liste des chiffres détectés n'est toujours pas vide, mais que la lettre suivante est un k, et que celle d'après est un g, on a bien des kilogrammes. Attention cependant à la structure du elif. Il faut bien s'assurer que l'on teste d'abord si la chaine de charactères a un indice i+1 avant de savoir s'il vaut « g ». D'où les parenthèse de priorisation, qui permettent d'invalider la condition elif si (i+1)<len(nom) n'est pas vérifiée avant de tester nom[i+1]. Dans tous les cas, cette fois là pas besoin de plus de conversion, on renvoie directement la quantité (après join() et int()) et l'unité « kg ».
- Sinon, si le caractère est un espace, on regarde si celui d'après est accessible (comme dans le point précédent en testant (i+1)<len(nom)). Si celui d'après est une lettre, il s'agit probablement de l'unité. On incrémente une nouvelle fois i en conservant la liste de chiffres. Ainsi, au prochain passage de la boucle while, on rentrera probablement dans une des deux premières conditions

- Enfin, si l'on n'a pas identifié de lettre correspondant à une unité, et que le caractère après un éventuel espace n'y correspond pas, on vide la liste de chiffres et on recommence. On incrémente i jusqu'à retomber sur un chiffre

```
if nom!="":
    chiffres = ""
    unit = ""
    res = 0
    identifie = False
    i=0
    while i<len(nom):
        while i<len(nom) and nom[i].isdigit(): #important de tester dans cet ordre
            chiffres+= str(nom[i])
            i+=1
        if i>=len(nom):
            break
        if len(chiffres) != 0 and (nom[i] =="L" or nom[i]=="l"):
            identifie = True
            unit = "m3"
            res = int("".join(chiffres))
            res /= 1000 #Penser à convertir les L en m3 !
            break
        elif len(chiffres) != 0 and (nom[i].lower()=="k" and (i+1)<len(nom)) and nom[i+1].lower()=="g":
            identifie = True
            unit = "kg"
            res = int("".join(chiffres))
            break
        #S'il y a un espace entre la qté et l'unité
        elif (nom[i].lower()==" "and (i+1)<len(nom)) and nom[i+1].isalpha():
        #On passe au caractère suivant, ça passe pas par le "else" ni par le second "while" et tout rentre dans l'ordre !
            i+=1
        else:
            chiffres = ""
            i+=1
    if identifie:
        return res,unit
    else:
        fonctionsMatrices.print_log_erreur("Detection de la quantité par sac indéchiffrable pour: "+nom+". A rajouter dan
        return 1,"kg"  #Par défaut, si on a vraiment rien pu trouver, on donne 1KG
return 0,'kg'
```

Figure 6 Identification de la quantité dans le nom d'un sac

iii. Fonction calc_fret_par_MP() (fonctionsFret.py)

Cette fonction traite les données semi-brutes de la liste d'achats. Le but est de calculer à la fois la matrice fret, explicitée page 16, qui donne les informations du fret pour chaque matière première, mais aussi la matrice resultat_usine_MP qui contient les informations du fret par usine. Pour optimiser le calcul et éviter de parcourir deux fois la longue listeAchats, on remplit les deux matrices simultanément.

Premièrement, on ne va travailler que sur la slice listeAchats[1 :], car l'élément d'indice zéro contient les entêtes de listeAchats. On les intègrera cependant à la fin à la liste résultat pour en faire les entêtes de *fretMP*.

On initialise comme listes vides la liste de résultat fretMP et celle qui contiendra les possibles problèmes de codes postaux rencontrés, erreursCP. On crée également une liste des intitulés des matières premières (apparaissant une seule fois) à partir de listeAchats, en la passant par fonctionsMatrices.extraire_colonne_n pour ne prendre que la colonne d'indice 1, puis par fonctionsMatrices.listeunique() pour supprimer les doublons.

On récupère ensuite les deux tableaux qui correspondent à l'import terrestre et maritime, qui se trouvent dans la feuille d'indice 3 des Entrées Manuelles. Le fonctionnement de ces tableaux est expliqué en page 13. Les fonctions qui permettent cela sont lireImportMaritime() et lireImportTerrestre(), détaillées dans la sous-partie suivante.

On prépare après cela la matrice resultat_usine_MP, qui se remplira ainsi pour rappel :

Tableau 17 Variable resultat_usine_MP

Variable **resultat_usine_MP**	0	1
Ligne de MP	Nom de la MP	*Sous tableau des quantités par site*
Type	String	List
Sous tableau	-	Cf tableau suivant

Les lignes du *Sous tableau des quantités par site* sont les suivantes, pour chaque usine. Chaque case contient une liste de deux éléments :

Variable *Sous tableau des quantités par site*	0		1	2	3	4	5
Ligne par usine	Tableau identification usine		Tonnage	Km route	Km bateau	BC route	BC bateau
Type	List		Float	Float	Float	Float	Float
Sous-sous tableau	Code postal	Nom de l'usine					
Type	Int	String					

Ainsi on a pour chaque intrant la quantité, le kilométrage, et le bilan carbone dans chaque usine.

La boucle principale commence ensuite, et tout ce qui suit est exécuté pour chaque matière contenue dans la liste unique créée précédemment.

❶

On a donc une matière X donnée par son nom.

On initie premièrement le compte du nombre de livraisons qui se font par bateau et par camion pour chaque matière (nombre_livraisons_route_tot et nombre_livraisons_bateau_tot), ainsi que le compte du nombre de kilomètres faits en camion ou en bateau[18] (sommeRoute_tot et sommeBateau_tot). Enfin, on initialise un comptage du tonnage total sous tonnage_tot. On note que le suffixe '_tot' aux variable témoigne de fait qu'elles se rapporte à des données spécifiques à l'intrant mais sommées sur toutes les usines.

On récupère ensuite dans la liste masse_vol_MP la masse volumique de l'intrant qui nous intéresse. Si on ne le trouve pas, ce qui n'arrive que si il n'a pas été identifié ou assimilé précédemment, on lui attribue une masse volumique de 1t/m3.

[18] On note ici qu'une livraison qui emprunte un bateau compte comme une livraison bateau ET une livraison camion, due aux transferts de charge au départ et à l'arrivée.

On parcourt après cela la liste des usines concernées par cette matière première, qui se trouve en case d'indice 1 du premier tableau. On initialise encore des compteurs du nombre de livraison route et bateau, cette fois avec le suffixe '_usine', car on veut comptabiliser le nombre de livraison par ce mode de transport de cette matière première et dans cette usine.

Ensuite, on rentre dans une troisième boucle où l'on va cette fois parcourir la listeAchats. Il va s'agir de recenser tous les achats de la matière X par l'usine en question, qui ont été enregistrés. On identifie cela par la condition if listeAchats[j][1]==listeunique[i] et une identification du numéro de dépôt, le tout immédiatement après l'entrée dans cette troisième liste.

On note ici l'avantage de mettre l'identification du numéro de dépôt, moins gourmande en mémoire, avant la comparaison de chaque nom de la liste. S'il y a dix dépôts équitablement livrés, cela évite un 90% de comparaisons de chaines de caractères, ce qui peut être précieux lorsque l'on a de grandes listes à parcourir.

On travaille alors pour une livraison donnée, qui correspond à notre matière X recherchée.

Le compte des livraisons en camion est incrémenté.

On enregistre d'abord le dépôt de livraison dont il s'agit, le pays d'origine, ainsi que la quantité. Selon si l'unité est en m3 ou en kg, on enregistre correctement les variables « m3 » ou « kg ». Cependant, si l'unité n'est pas reconnue, on le signale par un message d'erreur, et aucune valeur ne sera incrémentée. Cette exception ne devrait arriver que si elle avait été détectée dans la fonction **lireBDDMP()**.

Grâce à la masse volumique précédemment lue, on passe le tout en tonnes. On incrémente la variable tonnage_tot avec le tonnage de cette livraison, et on enregistre dans la matrice résultat.

On calcule ensuite la distance de fret liée à cette livraison, grâce à la fonction calcul_distance_fret_amont() présentée après. Elle renvoie une distance de route et une distance de bateau, ainsi

Une fois que la boucle for qui parcourt la listeAchats, on a calculé le fret pour chaque livraison qui correspondait à la matière première X. On calcule les moyennes des trajets en route et en bateau, et on enregistre dans la matrice resultat_usine_MP les quantités totales de m3 et de kg, ainsi que les distances totales de route et de bateau. On attribue ensuite une part de responsabilité dans ces quantités et distances pour chaque usine, comprise entre 0 et 1, et étant le ratio entre la quantité livrée à ce dépôt par rapport à la quantité totale. Ainsi, si une usine s'est fait livrer 100% du stock d'une matière, c'est à elle que l'on attribuera les émissions de ces intrants.

Enfin, on ajoute toute la ligne [nomMP, sommeRoute,sommeBateau, moyenneRoute, moyenneBateau, qtte]

On recommence à ❶ encore pour toutes les autres matières premières de listeUnique.

Juste avant de finalement retourner les deux matrices fretMP et resultat_usine_MP, on insère à l'indice zéro de fretMP les entêtes indicatives, et on enregistre avec geolocalisation.corriger_les_erreurs_CP() la liste des erreurs de codes postaux.

Pour aller plus loin, et calculer la valeur finale du fret amont, il faut convertir toutes les quantités de qtte en tonnes. Ainsi, on va avoir besoin des masses volumiques, et donc d'un peu de traitement de gestionIntrants.py à partir de la sous-partie vii, page vii34.

iv. calcul_distance_fret_amont()

On a d'abord une divergence selon si le produit vient de France (donc par camion), d'Europe (hors France) par camion, ou de n'importe où mais par bateau.

Si la matière première vient de France, cela va être relativement aisé d'estimer le fret, grâce aux codes postaux.

Déjà, on peut dire que la distance faite en bateau est nulle. Ensuite, on utilise la fonction geolocalisation.distance_CP() en lui donnant comme arguments la liste des codes postaux BDD_geoloc, le code postal du fournisseur français, inscrit dans la case d'indice 6 de la ligne de listeAchats, le code postal du dépôt d'arrivée, et le facteur de conversion VO/route 1.3.

La fonction distance_CP() renvoie la distance calculée en indice 0, et indique False en indice 1 si les deux codes postaux étaient bien référencés et qu'aucune erreur n'est à signaler, et le code postal problématique sinon. Dans le cas où il n'y a pas de problème, on incrémente la sommeRoute par la distance calculée entre les deux codes postaux. S'il y a eu une erreur, on enregistre le code postal problématique dans la liste erreursCP.

Si la matière première vient en camion mais pas de France, on utilise la liste d'imports terrestres récupérée au début de la fonction. Les pays que cela concerne sont listés ici : Allemagne, Belgique, Espagne, Pays-Bas, et Italie. Les livraisons de tourbe d'Estonie dans certains dépôts est aussi concernée par le fret en camion. Mais pour laisser le fret maritime de la tourbe être calculé pour les autres dépôts, on n'ajoute pas l'Estonie à cette liste. A la place, on rajoute la condition or ("tourbe"in listeAchats[j][1].lower() and depot_arrive in [50500,7170,1370,44850,29530,40210]). On peut donc ajuster ici les dépôts qui se font livrer par camion la tourbe estonienne.

Là encore on peut mettre à zéro la distance de bateau.

On va rechercher les coordonnées GPS du fournisseur. Pour cela, on pose latA = -180 comme un marqueur. On parcourt ensuite la liste des fournisseurs, et on compare les noms. Si une entrée du tableau de l'import terrestre contient le nom du fournisseur, on enregistre les coordonnées GPS (en écrasant la valeur 180 mise précédemment), et on casse la boucle for.

Si après cette recherche, la latitude latA est toujours à -180, c'est que rien n'a été trouvé dans l'identification du nom du fournisseur (puisqu'il n'y aura pas normalement dans le tableau la valeur -180).

On essaye donc de trouver des coordonnées GPS qui correspondent au pays. On reparcourt la même liste, comme on vient de le faire, mais avec la première colonne, celle des localités, et en en comparant les éléments avec le pays du fournisseur. S'il y a correspondance, on écrit les coordonnées GPS de la ligne en écrasant latA.

Si après cette seconde recherche, la latitude latA est toujours à -180, c'est que rien n'a été trouvé dans l'identification du pays du fournisseur non plus. On renvoie dans ce cas un message d'erreur et on attribue des coordonnées quelconques[19].

Une fois les coordonnées obtenues, on utilise geolocalisation.distance_GPS_CP() pour estimer la distance entre deux points si l'un est caractérisé par son code postal (ici le dépôt d'arrivée) et l'autre par ses coordonnée GPS (ici l'usine du fournisseur). De la même manière que précédemment, si le résultat d'indice 1 de cette fonction est False, il n'y a pas d'erreur à signaler, et sinon on enregistre le code postal problématique.

Enfin, si la matière première vient par bateau, on incrémente nombre_livraisons_bateau, et on utilise la liste import_maritime. De la même façon que juste avant, on essaye de trouver le nom du fournisseur dans import_maritime, puis, si on ne l'a pas trouvé, par le nom du pays. Si on trouve, on lit la distance de bateau, et la distance de route entre l'usine du fournisseur et le port d'expédition, qui a été calculée dans lireImportMaritime().

Il ne reste plus qu'à compléter la première distance de route avec la seconde, celle entre le port d'arrivée et le dépôt Florentaise de la livraison. Les coordonnées du port sont dans import_maritime, et on utilise encore geolocalisation.distance_GPS_CP() pour estimer la distance entre les coordonnées GPS et le code postal du dépôt.

On complète alors la distanceRoute avec cette seconde distance, puis on incrémente sommeRoute et sommeBateau.

v. Fonctions lireImportTerrestre() et lireImportMaritime() (fonctionsFret.py)

LireImportTerrestre() est très simple : on ouvre le fichier Excel et la feuille adéquate, puis on itère sur les lignes et les colonnes pour remplir le tableau des fournisseurs :

Tableau 18 Variable import_terrestre

0	1	2	3
Lieu (pays, ville…)	Nom du fournisseur	Latitude (radians)	Longitude (radians)
Str	Str	Float	Float

La fonction **lireImportMaritime()** lit les tableaux uns à un. D'abord celui des lieux, à gauche, où sont mélangés les usines de fournisseurs, les ports d'expédition et les ports de réception en France. Ensuite, le tableau des chaines de livraison est lu, avec à chaque ligne l'indice de l'usine de départ, l'indice du port d'expédition, l'indice du port de réception et la distance à parcourir en bateau entre les deux.

On cherche ensuite à connaitre la distance de bateau et de route pour acheminer la matière première qui fait appel à un fournisseur. Ce calcul se fera dans calc_fret_par_MP() mais on va préparer le calcul en explicitant les points du second tableau avec les informations du premier.

Ainsi, après avoir préparé une matrice résultat vide, on procède ligne par ligne (chaine de livraison par chaine de livraison. On lit dans le premier tableau latA et longA les coordonnées de l'usine de production à l'origine de la matière première, et latB et longB les coordonnées du port d'expédition. Avec l'aide la formule de calcul des estimations des distances à partir des coordonnées GPS, on a donc

[19] latA = 25.317° et longA = 54.9° correspondont au centre de l'Europe (à peu près…)

la première distance de route. Pour la distance de bateau c'est facile, elle est déjà renseignée dans le second tableau.

On renvoie donc un tableau comprenant, pour chaque chaine de livraison :

Tableau 19 Variable import_maritime

0	1	2	3	4	
Lieu de l'usine (pays, ville…)	Nom du fournisseur	Première distance de route (usine->port) (km)	Distance de bateau (km)	Coordonnées du port d'arrivée	
Str	Str	float	int	List	
				latPA	longPA
				Float	Float

Ainsi il ne restera qu'à calculer la seconde distance de route avec les coordonnées en indice 4 de la ligne selon le dépôt dans lequel la matière première est livrée.

vi. Fonction lire_FEtransport () (fonctionsFret.py)

La fonction lire_FEtransport() se trouve dans fonctionsFret.py, mais elle est aussi utile dans les calculs de fret aval et de fret de sacherie. Elle est donc appelée au tout début du fichier main.py principal.

La fonction lit de manière très classique le fichier des entrées manuelles à la feuille 2, pour renvoyer deux flottants, correspondant au facteur d'émission du camion et celui du bateau.

On lit toute la liste qui est présente dans le fichier Excel, mais on ne retient au final que celui dont l'intitulé est sélectionné dans les cellules E8 et E9 de la feuille 2 de Entrées Manuelles.

vii. Fonction lireFE_matprem() (gestionIntrants.py)

C'est encore une fonction assez classique de lecture du fichier Entrées Manuelles à la feuille 1. Les tableaux lus sont celui des facteurs d'émission (en kgCO2e/t) par famille de bilan carbone, des familles de bilan carbone par nom de matière première (et le taux d'azote dans le cas des engrais azotés, le tableau des masses volumiques (en t/m^3) par nom de matière première, et enfin le tableau des engrais avec le facteur d'émission (en $kgCO2e/t_{azote}$).

On renseigne simplement les numéros des lignes et colonnes des cellules dans lesquelles se trouvent ces tableaux, puis on itère sur les lignes (et éventuellement les colonnes) en remplissant au fur et à mesure les tableaux retournés à la fin de la fonction.

viii. Fonction associerMPetFE_fab (gestionIntrants.py)

On veut ensuite assembler les tableaux lus par la fonction précédente. Le principe est qu'on se base sur la liste MP_familles_N (celle avec les noms des matières premières, leur famille et, pour les engrais, leur taux d'azote). On utilise la famille pour lier cette table à celles des facteurs d'émission, les tables FE_familles et FE_engrais. Le but est de finir avec un tableau à deux colonnes, avec dans la première le nom des matières premières, et dans la seconde leur facteur d'émission en kgCO2e/t.

Pour cela on parcourt la liste MP_familles_N : ❶

Si le nom de la famille de l'intrant fait référence à un engrais (en commençant par « Engrais » ou par « Ammonitrate »), on doit chercher le facteur d'émission dans la liste FE_engrais. On utilise la fonction

fonctionsMatrices.recherche_elem() pour rechercher le nom de la famille dans cette liste et en retourner le facteur d'émission. La fonction renverra une valueError si elle ne trouve pas l'élément. On détecte cette erreur avec un « try : except : » et dans ce cas on écrit un message d'erreur et on donne un facteur d'émission nul.

Si le nom de la famille n'est pas un engrais, on recherche exactement pareil dans l'autre table, FE_familles.

On ajoute ensuite la ligne de la matière première au tableau de résultat et on recommence à ❶ pour les autres éléments de MP_familles_N.

Note : On pourrait s'interroger à ce moment sur le point suivant : comment être sûr·e que la liste des familles de bilan carbone par nom d'intrant écrite dans le fichier excel des Entrées Manuelles, est suffisamment complète ? N'y aura-t-il pas sûrement des éléments inscrits dans les achats qui n'y figurent pas, ou alors avec un nom légèrement différent ? L'interrogation est légitime car non, beaucoup des intrants effectivement achetés ne figurent pas dans la liste du fichier. Ainsi, on ne pourra pas faire d'association exacte entre les noms des matières premières achetées et inscrites dans listeAchats/fret, et ceux inscrit dans le tableau listant les matières premières et leur facteur d'émission. Et, catastrophe, le même problème se pose pour leur associer une masse volumique si c'est nécessaire ! Pas d'inquiétude, on résoudra cela du mieux qu'on peut dans la sous partie suivante…

ix. Fonction associer_fret_FEfab() (gestionIntrants.py)

Cette fonction un peu longue à pour but de faire communiquer la liste des matières première achetées, et dont on a calculé le fret dans fonctionsFret.calcul_fret_par_MP(), avec la liste des facteurs d'émission issus de gestionIntrants.associerMPetFE_fab(). Comme exprimé dans la note juste avant, cela n'est pas facile, car les noms ne vont pas forcément correspondre exactement, et une « Recherche d'un élément par nom exact » ou une « Recherche d'un élément par nom approximatif » ont de grandes chances d'échouer. Bref, prenons la fonction dans l'ordre, depuis le début.

D'abord, on fait une copie de la liste des matières premières et FE en en supprimant les lignes vides, avec la construction d'une liste par compréhension : MP_et_FEok = [x for x in MP_et_FE if x[0]!=""].

On note également la longueur de cette liste, et on initialise la liste des résultats introuvés.

On parcourt ensuite la liste fret, seulement à partir de l'indice 1 (car l'indice 0 contient les entêtes du tableau fret). Pour chaque matière première inscrite dans fret (une matière par ligne), on va chercher à lui attribuer un facteur d'émission de MP_et_FE.

On note le booleen 'trouve' à False, qui recensera si on a trouvé ce que l'on cherche, et on essaye ensuite quatre méthodes consécutivement. Elles sont classées de la plus exacte à la moins exacte. Une méthode de recherche n'est exécutée que si la précédente n'a pas réussi à trouver un facteur d'émission, et donc à inverser la valeur de 'trouve'.

Pour chaque méthode, on parcourt la liste MP_et_FE. Si on trouve, on change la valeur de 'trouvé', on incrémente le nombre d'éléments trouvés, et on ajoute dans la liste MP_et_FE une ligne à la fin qui comprend [élément recherché, FE, élément auquel il a été assimilé].

Puisqu'on modifie la liste MP_et_FE en ajoutant des éléments en même temps qu'on y recherche des éléments, on veille à ne la parcourir que jusqu'à l'indice n, la longueur de la liste initiale. Sinon, on risque d'avoir une boucle infinie.

On détaille donc ci-après les quatre méthodes d'identification utilisées :

La première est une comparaison exacte (sauf majuscules). Durant le premier parcours de la liste des FE, on compare les noms (mis en minuscule par la fonction .lower()) avec une comparaison exacte '==’.

```
#On parcourt la liste des facteurs d'émission
for k in MP_et_FEok[1:]:
    try:
        float(k[1])
    except ValueError:#L'exception arrive souvent quand on a un caractère vide -> on suppose que c'est 0
        k[1] = 0
        logging.info("Le FE de "+fret[i][0]+" est vide. On lui attribue un FE de 0.", inspect.stack()[0][3])

    if fret[i][0].lower()==k[0].lower():
        trouve = True
        compte +=1
        FE_MPfab = k[1]
        MP_et_FEok.append([fret[i][0], k[1], k[0]])
        break
```

Au passage, juste avant, on en profite pour « corriger » les valeurs numériques de la liste MP_et_FE. Si l'on détecte que le nombre renseigné n'est pas un float (ou une chaine de caractères interprétable comme telle), on envoie un message d'information, qu'il est remplacé par un FE nul. En effet, on ne met pas un message d'erreur, car ce cas de figure advient majoritairement quand la case est vide, c'est-à-dire que le champs « famille » n'est pas renseigné dans le fichier Excel, ou alors qu'il fait référence à une famille de manière incorrecte (faute d'orthographe), ou que la famille n'existe pas.

L'exception est attrapée en utilisant le fait que la méthode float() renvoie une ValueError quand on lui demande de transformer en flottant une variable qui n'en est manifestement pas un.

Si on a trouvé l'élément qui correspond (par le nom exact en minuscule), on met le marqueur à True, qui nous permettra de ne pas passer par les méthodes d'association restantes, on incrémente le compte du nombre d'éléments trouvés, on attribue le facteur d'émission (qui sera utile à la fin, après la recherche), et on ajoute une nouvelle ligne.

La seconde méthode est une comparaison inclusive des lettres uniquement. Cela fonctionne de manière similaire à la première, mais avant de comparer, on épure chaque nom de tous les caractères qui ne sont pas des lettres. Ainsi, on se passe notamment des chiffres des référence qu'il y a souvent à la fin des noms.

Cette méthode présente donc un peu plus d'approximation que la première méthode. Mais, étant donné que des facteurs sont exprimés en émission par masse de matière, le fait que l'on associe celui d'un sac de 10L à celui d'un sac de 15L ne donne pas d'erreur à priori. L'impact est davantage problématique pour la même correction qui adviendra dans la partie suivante, concernant cette fois la liste des masses volumiques.

```
if not trouve:        #Mais on va essayer de le trouver quand même
    #D'abord on vire les caractères qui ne sont pas des lettres
    nom_recherche = ""
    for c in fret[i][0]:
        if c.isalpha():
            nom_recherche+=c.lower()
    for k in MP_et_FEok[1:]:
        nom_k = ""
        for c in k[0]:#On met aussi en minuscule et simplifié cette liste
            if c.isalpha():
                nom_k+=c.lower()

        #Là on trouve si la seule différence était un chiffre ou autre symbole
        if nom_k!="" and (nom_k ==nom_recherche or nom_k in nom_recherche or nom_recherche in nom_k):
            trouve = True
            compte +=1
            FE_MPfab = k[1]
            MP_et_FEok.append([fret[i][0], k[1], k[0]])
            break
```

Pour ce qui est du détail de cette partie, le voici. On s'assure déjà que l'élément n'a pas été trouvé dans la première méthode. Si ce n'est effectivement pas le cas, on « épure » le nom que l'on recherche. On initialise une chaine de caractères vide, que l'on complète lors d'un parcours caractère par charactère du nom initial et dans le cas où le caractère vérifie la méthode isalpha(). Au moment de l'ajout, le charactère est également mis en minuscule.

Ensuite, on parcourt la liste des matières premières, et on fait de même avec chaque nom.

Enfin, on compare les deux chaines obtenues. On regarde si le nom que l'on compare n'est pas vide (car si c'est le cas il sera forcément compris dans n'importe quelle autre chaine de charactères) puis on regarde si l'un est compris dans l'autre, ou inversement. Si c'est le cas, on procède aux mêmes étapes qu'à la fin de la méthode précédente.

La troisième se base sur la distance de Levenshtein. Elle n'est cependant pas très fonctionnelle, et produit peu d'identifications par rapport aux autres (voire aucune). On la laisse cependant car elle n'est pas trop lourde en calcul, et peut servir dans certains cas (fautes de frappe par exemple).

Cette partie repose sur la fonction fonctionsMatrices.levenshtein(). Le code est directement récupéré de https://stackabuse.com/levenshtein-distance-and-text-similarity-in-python/ où elle est aussi très clairement expliquée. L'aboutissement est un score, un entier positif appelé *distance de Levenshtein*, qui traduit le nombre de différences entre les deux. Par exemple :

Tableau 20 Exemples de distances de Levenshtein

Str1	Str2	Levenshtein(str1, str2)
« test »	« text »	1
« test »	« tet »	1
« Cesi et un teste ? »	« Ceci est un test. »	4
« 5 »	« 6 »	1

Ainsi cette méthode calcule un score de ressemblance entre les deux chaines de caractères à comparer. On veut associer des noms pour lesquels la ressemblance est « suffisamment grande », c'est-à-dire pour lesquels le score est « suffisamment bas ». Afin de quantifier cela, on initialise la distance_min à une valeur infinie (ou plus grand que le nombre maximal de différences qu'il peut y avoir entre deux chaines de charactères), un seuil d'acceptabilité (par exemple 4), un taux de tolérance (par exemple 0.4), et une liste vide appelée temp, qui contiendra la meilleure assimilation trouvée.

```
if not trouve: #Si on a toujours pas trouvé, on récupère la distance de Levenshtein entre les deuxchaines de caractères
               #et on prend celle qui ressemble le plus
    distance_min = np.inf
    temp = []
    tolerance = 8 # nombre de caractères d'erreurs acceptés
    taux = 0.4
    for k in MP_et_FEok[1:]:
        distanceL = fonctionsMatrices.levenshtein(k[0].lower(), fret[i][0].lower())
        if distanceL<tolerance:
            trouve = True
            c_m3 +=1
            FE_MPfab = k[1]
            MP_et_FEok.append([fret[i][0], k[1],k[0]])
            compte +=1
            break
        elif distanceL<distance_min:
            distance_min = distanceL
            temp = [fret[i][0], k[1], k[0]]
            break
    if not trouve and len(temp)>0 and distance_min< taux*min(len(k[0]), len(fret[i][0])):
        trouve = True
        c_m3 +=1
        compte +=1
        FE_MPfab = temp[1]
        MP_et_FEok.append(temp)
```

Une fois cela initialisé, on parcourt à nouveau MP_et_FE. On note la distance de Levenshtein entre les deux chaines de charactères (mises en minuscule). Là, si la différence entre les deux est inférieure à la barre de tolérance (si moins de 4 « fautes »), on assimile immédiatement. On ne va pas cherche plus, on enregistre et on casse la boucle for pour passer à l'assimilation suivante.

Sinon, si la distance est supérieure ou égale à la tolérance, mais tout de même meilleure (donc inférieure) à la meilleure distance trouvée jusque-là, on enregistre l'amélioration. On dit qu'à présent la meilleure distance vaut celle que l'on vient de calculer, et l'élément qui a pour l'instant cette meilleure distance est enregistré dans la liste temp.

A la fin, si on a pas trouvé de nom qui ressemblait à moins de 4 différences au nom recherché, on s'intéresse au « moins pire », qui est enregistré dans temp. A priori, temps contient forcément quelque chose, car il y a forcément un nom qui ressemble à celui recherché plus que « pas du tout » (ce qui est sous-entendu par l'initialisation à une valeur infinie de distance_min). Si la distance minimale qu'on a trouvée est tout de même acceptable, c'est-à-dire inférieure au taux de tolérance multiplié par la longueur de la plus petite des deux chaines comparées, on se contente de cela. Et puis si on a rien qui ne s'en rapproche par cette méthode, on essaye une dernière fois.

La dernière méthode identifie par les mots. On tente de « comprendre » ce que contiennent les noms, en les décomposant en des listes de mots. Cela évacue le problème des charactères autres que les lettres, des espaces etc… Le traitement pour chaque nom est un peu plus long, mais relativement simple, et il est strictement identique pour les noms recherchés que pour les noms de MP_et_FE avec lesquels ils sont comparés.

On initialise une liste qui contiendra les chaines de charactères de chaque mot, et une chaine de charactères vide que l'on remplira un mot à la fois. On parcourt le nom à analyser, et on ajoute les charactères qui sont des lettres au mot en cours, jusqu'à ce qu'on arrive à un espace. Si on rencontre un chiffre, on casse avec un *break* la recherche à ce moment, et on ne prend pas en compte ce qui vient après[20].

Si on a atteint un espace, c'est logiquement que l'on vient de finir un mot. On s'intéresse donc à ce mot. S'il n'est pas très intéressant, c'est-à-dire qu'il est contenu dans une liste d'exception, on le réinitialise directement sans l'enregistrer ; ça ne nous aidera pas vraiment à savoir quelque chose de

[20] A vue d'œil, il n'y a pas dans la liste des produits achetés de cas où quelque chose de déterminant advient après un chiffre. Arrêter l'analyse ici est donc un bon moyen de ne pas récupérer des unités (kg, L ou mm par exemple)

signifiant sur le produit. On exclus donc par cette liste les noms de marque, les articles indéfinis, lettres isolés… Ensuite, il arrive que les mots soient de abbréviation. On retrouve de manière assez commune « TER » pour « Terreau », « PAI » pour « Paillage » ou « Pallis ». Ainsi, on a ensuite une liste de comparaison du mot en cours pour ajouter l'abréviation ansi que le mot complet.

Enfin, dans tous les cas, on ajoute le mot à la liste des mots du nom analysé.

A la fin de la boucle, il faut cependant faire attention à ajouter le dernier mot, car il n'y a probablement pas d'espace à la fin.

On procède ainsi pour le nom de la matière première recherchée, ainsi que, une fois dans la boucle for sur MP_et_FE, pour chaque nom à comparer.

Cette fois-ci, on cherche un score de ressemblance, qui doit donc être le plus grand possile.

Ainsi, une fois que l'on a les deux listes de mots « rech » et « mots_k », on parcourt l'une, en regardant pour chaque mot, si ce mot est dans l'autre. A chaque fois que c'est le cas, on augmente le score de 1.

```
if not trouve: #Si là on a toujours pas trouvé, on tente l'identification des mots
    rech = []
    mot = ""
    for c in fret[i][0]:
        if c.isdigit():
            break
        elif c != " " and c.isalpha():
            mot +=c
        elif c ==" " and mot!= "":
            if mot not in ["DE", "TaN", "TN", "TRUFFAUT", "BOTANIC", "SYSTEME", "U", "DOR", "D", "N", "K"]:
                if mot == "TER":
                    rech.append("TERREAU")
                elif mot =="INT":
                    rech.append("INTERIEUR")
                elif mot == "PTES" or mot == "PLTES":
                    rech.append("PLANTES")
                elif mot =="PAI":
                    rech.append("PAILLIS")
                elif mot in "AROM":
                    rech.append("AROMATIQUES")
                elif mot in "AQUAT":
                    rech.append("AQUATIQUES")
                rech.append(mot)
            mot = ""
    if len(mot)!= 0 and mot.lower() not in ["l", "kg", "mm", ]:
        rech.append(mot)
    stemp = 0
    temp = []
    for k in MP_et_FEok[1:n]:
```

```
for k in MP_et_FEok[1:n]:
    mots_k = []
    mot = ""
    for c in k[0]:
        if c.isdigit():
            break
        elif c != " " and c.isalpha():
            mot +=c
        elif c ==" " and mot!= "":
            if mot not in ["DE", "TaN", "TN", "TRUFFAUT", "BOTANIC", "SYSTEME", "U", "DOR", "D", "N", "K"]:
                if mot == "TER":
                    mots_k.append("TERREAU")
                elif mot =="INT":
                    mots_k.append("INTERIEUR")
                elif mot == "PTES" or mot == "PLTES":
                    mots_k.append("PLANTES")
                elif mot =="PAI":
                    mots_k.append("PAILLIS")
                elif mot =="AROM":
                    mots_k.append("AROMATIQUES")
                elif mot in "AQUAT":
                    rech.append("AQUATIQUES")
                mots_k.append(mot)
            mot = ""
    if len(mot)!= 0 and mot.lower() not in ["l", "kg", "mm", ]:
        mots_k.append(mot)
    score = 0
    comm = []
    for v in rech:
        for w in mots_k:
            if v.lower()==w.lower():
                score += 1
                comm.append(v)
    if score >= 3:
        trouve = True
        FE_MPfab = k[1]
        MP_et_FEok.append([fret[i][0], k[1],k[0]])
        compte +=1
        break
    elif score >=2 and score>stemp:
        temp, stemp, elco = [fret[i][0], k[1], k[0]], score, comm

    elif score==len(mots_k):
        trouve = True
        FE_MPfab = k[1]
        MP_et_FEok.append([fret[i][0], k[1],k[0]])
        compte +=1
if temp != [] and stemp>=1:
    trouve = True
    compte +=1
    FE_MPfab = temp[1]
    MP_et_FEok.append(temp)
```

Si le score est de trois ou plus, donc s'il a trois mots en commun ou plus, c'est suffisant pour assimiler les deux noms, et on arrête la recherche là. Sinon, si le score est tout de même meilleur que le meilleur trouvé jusque-là, on enregistre le nouveau meilleur, en mettant à jour le score maximal stemp. Sinon, si le score ne pourra pas s'améliorer, c'est-à-dire si tous les mots d'un nom sont compris dans l'autre, on a trouvé le meilleur, et on s'arrête aussi.

Enfin, si à la fin on a rien trouvé de parfait, mais qu'on a un enregistrement avec un mot en commun dans temp, on se contente de cela.

Si après ces trois méthodes, rien n'a été trouvé, on ajoute le mot à une liste de mots introuvés, afin de rendre plus facile à l'utilisateur·ice la tâche de compléter la base de données. S'il n'y a qu'un élément, on l'affiche directement dans le log des erreurs.

Enfin, on ajoute à la matrice fret les colonnes correspondant au FE de fabrication, à la somme $FE_{fabrication}$ + $FE_{fret\ amont\ cumulé}$, et à la somme $FE_{fabrication}$ + $FE_{fret\ amont\ moyen}$. Elles correspondront respectivement aux colonnes 10, 11 et 12 de fret.

Pour dire un petit mot sur l'utilité des méthodes : sur une liste d'achats donnée, qui contient initialement 259 éléments, à comparer à la liste MP_et_FE qui en compte 351, la première méthode ne permet d'en identifier que 42%. La seconde en assimile, elle, 82% de ce qui reste, celle de Levenshtein aucun, et la dernière, par comparaison des mots, permet d'identifier 96% des restants, ce qui laisse encore un élément introuvé à ajouter manuellement.

La méthode de Levenshtein peut effectivement sembler inutile ici, mais elle permet de se prémunir d'éventuelles fautes de frappes, que ce soit dans l'entrée des commandes ou dans l'ajout d'un élément à la base de données. On pourrait éventuellement la passer après, en quatrième, afin qu'elle ne soit pas sollicitée sauf dernier recours. Mais d'un autre côté, elle est moins permissive que la dernière, et donc, si faute de frappe il y a, on sera plus à même de l'associer au bon élément qu'à n'importe quoi avec la comparaison des mots. C'est pour cette raison que l'on conserve cette méthode, et qu'on la conserve en troisième position. De toute façon, dans l'exemple cité précédemment, elle n'est appelée que pour 27 éléments (soit 10% des cas).

x. Fonction corriger_noms_massesvol (gestionIntrants.py)

Cette fonction a un but similaire à celui de la fonction associer_fret_FE_fab() précedente. Elle ajoute avec les masses volumiques les éléments qui n'en n'ont à priori pas une, mais qui peuvent fonctionner par assimilation. Cependant, on évite de refaire tout le travail que l'on vient de faire, et de devoir repasser à nouveau toutes les listes à l'analyse plusieurs fois. En effet, on a pris soin, dans la partie ix, d'ajouter à MP_et_FE, dans la troisième colonne, le nom des éléments nouveaux auxquels on pouvait les assimiler. Ainsi, cela nous permet de procéder comme suit sur chaque élément de MP_et_FE.

```python
def corriger_noms_massesvol(masses_vol:list,MP_et_FE:list, fret:list):
    n = len(masses_vol)
    introuves = []
    compteur = 0
    for elem in MP_et_FE[1:]:
        if len(elem)==3:#C'est qu'on lui a trouvé un substitut dans "associer_fret_FEfab
            nom_recherche = elem[2]
        else:
            nom_recherche  = elem[0]

        trouve = False
        for masvol in masses_vol[:n]:
            if nom_recherche.lower() in masvol[1].lower() or masvol[1].lower() in nom_recherche.lower():
                if nom_recherche != elem[0]: #Si pour le trouver on a utilisé un substitu, on rajoute le vrai élément au tableau
                    compteur +=1
                    masses_vol.append([n+compteur, elem[0], masvol[2]])
                trouve = True
                break
        if not trouve:
            introuves.append(elem[0])
    for el in fret:
        trouve = False
        for masvol in masses_vol:
            if el[0].lower() in masvol[1].lower() or masvol[1].lower() in el[0].lower():
                el.append(masvol[2])
                trouve = True
                break
        if not trouve:
            el.append(1)
    if len(introuves)>0:
        fonctionsMatrices.print_log_erreur(str(len(introuves))+" masses volumiques sont à ajouter manuellement", inspect.stack(
        # print(introuves)
    fret[0].append("Masse volumique (t/m3)")
    fonctionsMatrices.print_log_info("Ajout de "+str(len(masses_vol)-n)+" entrées dans les masses volumiques par assimilation",
    return masses_vol, fret
```

Premièrement, on regarde la longueur. S'il contient deux éléments ([nom, FE]) c'est qu'il était dans la liste d'origine (c'est-à-dire dans « Entrées Manuelles.xlsx »), sinon, s'il en contient trois, ([nom, FE, nom assimilé]), c'est qu'il a été rajouté par gestionIntrants.associer_fret_FEfab(). Dans le premier cas, on recherche normalement ce nom dans la liste masses_volumiques, et dans le second on recherche à la place le nom assimilé.

xi. Fonction calc_quantite_et_BC (gestionIntrants.py)

Cette fonction est la dernière qui va intervenir sur la matrice de fret. Elle va utiliser les masses volumiques que l'on vient d'identifier et la colonne des quantités (en kg et m3) pour calculer le tonnage. Enfin, à l'aide de ce tonnage et des FE, elle calcule les bilans carbones complets.

L'avantage, c'est que l'ajout de la masse volumique à la matrice fret a été fait dans la fonction précédente, il ne reste donc qu'à la récupérer en colonne 13 et convertir les volumes en tonnes. On somme cela à la masse qui existait déjà (en pensant à la convertir de kilogrammes en tonnes

xii. Fonction renverserMP_usine_fret (fonctionsFret.py)

Le seul but de cette fonction est de mettre en forme les informations contenues dans la matrice resultat_usine_MP. A cette fin, après avoir initialisé la matrice res avec [[id usine],masse, kmroute, kmbateau, BCroute, BCBateau], où id_usine est l'ensemble avec nom, code GCO et code postal, on parcours la liste resultat_usine_MP, matière première par matière première.

On récupère sa masse et son volume total, toutes usines confondues, sa masse volumique, puis la route et le bateau de fret amont. On parcourt la liste des usines où cette matière a été livrée. Là, on lit les parts (comme expliqué en 6.b). On calcule donc pour chaque usine la masse, et les distances, ainsi que le bilan carbone de ces distances, puis on les ajoutes à la bonne case de la matrice résultat

xiii. Fonction calc_fin_de_vie (gestionIntrants.py)

c. GestionSacherie.py

Imports:

Le but ici est de lire le fichier de consommation de sacherie par référence et par usine, de le croiser avec la liste des références lue dans Entrées Manuelles, puis de calculer les matériaux consommés par les emballages, ainsi que le fret amont associé.

i. Fonction lireConsoSacherie() (gestionSacherie.py)

C'est une fonction de lecture du fichier Excel 'Conso sacherie.xlsx' très classique, se référer au Structures récurrentes, page 6, pour des indications sur le fonctionnement.

ii. Fonction qte_materiaux_sacherie () (gestionSacherie.py)

C'est cette fonction qui se charge de croiser la liste de la consommation, stockée dans refs_sacherie, et de la croiser avec la base de données des références contenue dans le fichier des entrées manuelles.

D'abord, elle lit les références stockées dans les entrées manuelles, de haut en bas, donc de la plus ancienne à la plus récente. Cette partie est très classique, pour plus de détails, voir au chapitre Structures récurrentes, page 6. On obtient une liste **reference_materiaux** qui contient pour chaque réf :

Tableau 21 Variable reference_materiaux

0	1	2	3	4	5
Nom de la référence	Matériau (PEBD ou papier)	Masse (g)	Taux de recyclage [0,1]	Imprimeur	Année de la référence
Str	Str	Float	Float	Str	Int

Ensuite, on initialise les deux matrices résultat : une qui contient la consommation en trois matériaux (PEBD, PEBD recyclé, ou papier) de chaque site ainsi que le fret de sacherie total (tous matériaux confondus) à destination de cette usine.

L'autre matrice résultat contient pour chaque usine son numéro, la distance cumulée de fret, et le produit distance x masse.

On s'occupe aussi de lire la liste des imprimeurs qui est à côté de la liste des références de sacherie dans le fichier des entrées manuelles. Celle-ci ne contient que leur nom et leurs coordonnées GPS. Elle servira bien entendu à calculer le fret en amont de la sacherie.

Enfin, avant d'attaquer la lecture de la liste des consommations, on inverse la liste des références que l'on vient de lire. Ainsi, les références les plus récentes se trouvent en premier. En la parcourant, du haut vers le bas, et en vérifiant que leur date de mise en fonctionnement est antérieure à la date de calcul du bilan carbone, on pourra s'arrêter à la première référence trouvée.

On parcourt donc la liste des consommations, puis pour chaque référence consommée, on parcourt la liste des références reference_materiaux. Dès que les deux se croisent et que l'année de mise en utilisation de cette référence est cohérente avec l'année du bilan carbone, on enregistre le matériau, masse, imprimeur, taux de recyclé. Également, on récupère les coordonnées GPS de l'imprimeur en parcourant le tableau imprimeries que l'on a lu juste avant.

Une fois que l'on a les informations concernant ce sac, on calcule la masse de PEBD vierge, de PEBD recyclé et de papier qu'il contient, à partir du matériau, du taux de recyclage et de sa masse. Potentiellement, on pourrait faire des petits ajustements ici pour prendre en compte du papier à différents taux de recyclé, voire même, avec un peu plus d'efforts, des sacs avec plusieurs matériaux.

Enfin, maintenant qu'on a la quantité de matériaux, il ne reste qu'à calculer le fret.

Pour chaque usine dans laquelle ce matériau est livré, on récupère le code postal. Ensuite, avec les coordonnées GPS, ce code postal et la fonction geolocalisation.distance_GPS_CP() on estime la distance entre l'imprimerie et l'usine.

A la fin, on a un tableau dont chaque ligne comporte :

Tableau 22 Variable conso_materiaux_par_site

0	1		2		3		4
Usine	Matériau	Masse	Matériau	Masse	Matériau	Masse	Distance totale
Int	Str	Float	Str	Float	Str	Float	Float

On a aussi tableau similaire pour le fret :

Tableau 23 Variable fret_sacherie_par_usine

0	1	2
Usine	Distance cumulée (km)	Dist. Cumulée*masse unitaire*quantité (t.km)
Int	Float	Float

iii. Fonction lire_FE_emballage()

C'est une autre fonction classique de lecture du fichier Entrées Manuelles. Elle retourne sous forme de tableau les données des FE des matériaux classiques de sacherie. Se référer à Structures récurrentes, page 6, pour plus d'indications concernant son fonctionnement.

iv. Fonction BC_sacherie () (gestionSacherie.py)

Une fois les données des quantités de chaque matériau et le fret associé récupéré, cette fonction est assez simple.

Une première passe dans fret_sacherie_par_usine en multipliant par le facteur d'émission du fret routier en $kgCO_{2eq}$/t.km donne le bilan carbone en fret. Une seconde permet de remplir un tableau de la forme

Tableau 24 Variable conso_materiaux_par_sites

0	1	2	3	4	5
Site	PEBD (t)	PEBDr (t)	Papier (t)	PVC (t)	Carton (t)
Liste (nom, n° et CP)	Float	Float	Float	Float	Float

Comme le tableau conso_materiaux_par_sites contenait les quantités en tonnes, et comme le facteur d'émission contenu dans les Entrées Manuelles est en $kgCO_{2e}/t_{materiau}$, on pense à convertir en tonnes de CO2 équivalent.

d. gestionExport.py

Le module gestionExport a pour but d'estimer les émissions de GES dues au fret aval des produits vendus par les sites étudiés. La partie délicate de cette étape est d'une part le nombre de données à traiter (plus de 18000 livraisons), ainsi que la variété de celles-ci, tantôt en France, tantôt à l'étranger, parfois regroupées en une seule livraison, parfois de l'interdépot.

i. lireExport()

La première étape est de lire les adresse des destinataires à l'étrangers, adresses pour lesquelles la méthode de calcul par base de données des codes postaux ne fonctionnera pas.C'est le rôle de la fonction lireExport().

Celle-ci ouvre le fichier des entrées manuelles à l'onglet d'indice 4, comprenant les adresses d'export maritime et terrestre. Dans la suite, seule le tableau des adresses terrestre nous aidera, mais le principe de lecture est le même que dans v.Fonctions lireImportTerrestre() et lireImportMaritime() (fonctionsFret.py).

Les deux tableaux renvoyés par lireExport sont bdd_exp_terre, et bdd_exp_mari, dont la structure est donnée ci-après.

Tableau 25 Variable bdd_exp_terre

bdd_exp_terre	**Indice**	**Pays/localisation**	**Client**	**Longitude (rad)**	**Latitude (rad)**
Indice	0	1	2	3	4
Type	Int	Str	Str	Float	Float

Tableau 26 Variable bdd_exp_mari

bdd_exp_mari	**Indice**	**Port de départ**	**Port d'arrivée**	**Distance bateau (km)**
Indice	0	1	2	3
Type	Int	List	List	Float

Où les indices 1 et 2 sont des sous-listes de ports de structure :

Tableau 27 Sous-liste des ports

Sous-liste port	**Indice**	**Ville**	**Pays**	**Longitude (rad)**	**Latitude (rad)**
Indice	0	1	2	3	4
Type	Int	Str	Str	Float	Float

ii. lire_couts_camion_aval()

Cette fonction n'est pas utilisée dans la suite du programme, mais elle est écrite dans l'éventualité d'une utilité future, concernant les livraisons aval. Sa structure est celle, classique, de la lecture d'un tableau Excel. Se référer donc aux Structures récurrentes pour plus d'explications. Les tableaux renvoyés sont ceux du fichier des Entrées Manuelles, onglet 2 :

Tableau 28 Varibale bdd_coutkm_dist

Dépôt	d< 250 km	d >250 km
St Mars	4.02	1.14
St Escobille	4.24	1.08
…	…	…
Moy globale	3.45	1.31

Tableau 29 Variable bdd_coutkm_E

'Dépôt'	'€ >= 100 €'	'100 =<€ =<500 '	'€ > 500 €'

'St Mars'	2.93	3.16	1.42
'St Escobille'	3.49	3.68	2.02
…	…	…	….
'Moy globale'	2.68	2.91	1.66

L'idée est de pouvoir éventuellement déduire du coût du transport la distance exacte parcourue par le transporteur. Mais l'approche des coordonnées GPS, jugée plus simple à implémenter a été préférée.

iii. lire_ventes()

La fonction de lecture des ventes est encore une fois une fonction de lecture simple d'un fichier de données, ici celui des ventes de l'année de calcul, se référer aux Structures récurrentes pour plus de détails.

La particularité ici est que l'on inscrit clairement avant d'itérer sur chaque ligne du tableau :

```
nbrows = feuille.nrows
col_groupement = 0
col_sens = 1
col_montant_port = 0
col_poids = 5
col_nbpalettes = 22 #Important: prendre la dernière colonne du nombre de palettes,
col_depot_depart = 7
col_date=8
col_num_client = 10
col_cp_arrivee = 15
col_type_client = 17
```

Cela permet d'éviter de lire les mauvaises colonnes et de les ranger correctement par la suite. Le tableau renvoyé est listeVentes, qui a la structure suivante :

Tableau 30 Variable listeVentes

ListeVentes	Numéro de groupement	Sens de livraison	Montant port	Masse (t)	Nb de palettes	Num dépôt départ	Date	Num client	CP arrivée	Type client
Indice	0	1	2	3	4	5	6	7	8	9
Type	Float	Str	Float	Float	Float	Float	Float	Float	Float	S

iv. regrouper_livraison()

Cette fonction a pour but de regrouper les livraisons du tableau listeVentes, lu précédemment, par numéro de regroupement. Le but est d'obtenir pour chaque numéro de groupement la ligne suivante :

Tableau 31 Variable livraisons

Livraisons	Numéro de regroupement	Numéro de dépôt	Type client	*Sous-livraison*
Indice	0	1	2	3
Type	Float	FLoat	Str	List

Où sous-livraison est un tableau de la forme suivant, regroupant chaque livraison ayant un même numéro de regroupement :

Tableau 32 Variable de sous-livraison

Sous -liste de livraisons	Numéro de groupement	Sens de livraison	Montant port	Masse (t)	Nb de palettes	Num dépôt départ	Date	Num client	CP arrivée	Type client
Indice	0	1	2	3	4	5	6	7	8	9
Type	Float	Str	Float	Float	Float	Float	Float	Float	Float	S

La structure de la fonction est celle-ci :

```
def regrouper_livraison(listeVentes:list):
    annuler_regroupements= False
    stat = [0,0,0]
    liste_groupes = []
    livraisons = []
    for vente in listeVentes[1:]:
        if [vente[0], vente[5]] not in liste_groupes or annuler_regroupements:#Si on a pas déjà cette livraison
            livraisons.append([vente[0], vente[5], vente[9], [vente]]) #On l'ajoute  (num grpement, num depot, type client,[ligne entière] )
            liste_groupes.append([vente[0], vente[5]]) #Et on signale qu'on y passe désormais (num grpement, num depot)
        else:
            i = liste_groupes.index([vente[0], vente[5]])
            livraisons[i][3].append(vente) #Important que ce listes soient remplies en même temps
    livraisons.insert(0, ["num grpement", "num depot", "type client", "sous-livraisons"])
```

D'abord, il y a un booléen permettant par la suite de ne pas regrouper les livraisons. Cela permet de simuler quel serait le bilan carbone du fret aval si aucune livraison n'est regroupée, mais il est bien sûr défini à False par défaut.

On initialise ensuite deux listes vides. Liste_groupes contiendra des identifiants uniques, enregistrant ainsi les numéros

Le principe est de parcourir la liste des ventes par vente. Si l'on rencontre un numéro de regroupement et un numéro de dépôt qui n'ont pas encore été enregistrés, on les ajoute à la liste livraisons et à une liste temporaire qui se construit progressivement avec les numéros de regroupement et de dépôt des livraisons déjà enregistrée. Sinon, si on l'a déjà enregistrée (présente dans liste_groupes), on récupère l'indice i qui indique quelle position il a lorsqu'il a été ajouté à la liste livraison et liste_groupes. Enfin, on insère la livraison comme sous-livraison à la liste livraisons.

Cela est rendu possible par le fait que les listes livraisons et liste_groupes sont construites en même temps avec les mêmes éléments. Cette structure un peu spéciale est utilisée pour éviter de rentrer dans une boucle infinie.

In fine, on obtient bien la structure des deux tableaux ci-dessus.

Note : on aurait pu faire cette étape de regroupement au même moment que la lecture du fichier des ventes pour éviter de rajouter un algorithme qui la parcourt encore une fois. Cependant, cela aurait été minime comme bénéfice (le regroupement des 18000 livraisons ne prenant que quelques secondes), pour une compréhension bien plus difficile et un implémentation moins intuitive.

v. calc_fret_aval()

Une fois la liste des achats regroupées, on peut passer au calcul du fret aval à proprement parler.

La structure du résultat principal, BC_fret_aval_par_usine, est la suivante, pour chaque usine :

Tableau 33 Variable BC_fret_aval_par_usine

BC_fret_aval_par_usine	Sous-liste de l'usine	Sous-liste des bilans carbones
Index	0	1
Type	List	List

Où

Sous-liste de l'usine	Code GCO de l'usine	CP de l'usine	Nom de l'usine
Index	0	1	2
Type	Float	Float	Str

Et

Sous-liste des bilans carbones	$BC_{Grand\ public}$(tCO2e)	$BC_{Interdepot}$(tCO2e)	BC_{PRO}(tCO2e)
Index	0	1	2
List	Float	Float	Float

On itère sur chaque élément de la liste livraisons, qui comporte une ou plus sous-livraison.

Premièrement, on initialise le bilan carbone de ce voyage à zéro. On identifie ensuite de quelle usine part la livraison à partir du code « sacherie » de l'usine. Pour cela, on utilise la liste renvoyée par lire_conversion_sacherie_GCO() et enregistrée comme variable globale dans CONVERSION_SACHERIE_GCO. Si on ne trouve pas, on le signale à l'utilisateur·ice et on part du principe que l'on attribue cette livraison à l'usine de St-Mars.

On calcule ensuite la distance de fret aval.

S'il y a plusieurs sous livraisons, c'est-à-dire que la longueur de l'élément d'indice 3 (les sous livraisons) est strictement supérieure à 1, on passe par la procédure d'optimisation décrite plus bas dans optimiser_livraisons(). On en récupère la distance parcourue, la distance multipliée par le tonnage, ainsi que le bilan carbone (donc masse*tonnage*FE route).

Sinon, s'il n'y a qu'une livraison à calculer, on cherche les coordonnées GPS du point d'arrivée avec la fonction geolocalisation.getGPSfromCP(). Une fois cela acquis, on calcule la distance entre le code GPS de l'usine et ces coordonnées. On multiplie cela par la masse transportée pendant la livraison. Si le bilan carbone est négatif, c'est que la masse était négative, par conséquent il s'agit d'un retour, ce qui correspond tout autant à une émission. On passe alors le bilan carbone en valeur absolue.

Maintenant que l'on a le bilan carbone, il reste à le ranger correctement dans le tableau BC_fret_aval_par_usine. On parcourt les lignes de ce tableau, en comparant le code de l'usine à celui de la vente, et, selon s'il s'agit de GP (mention CLA, CLB, ou ANJ), d'inter-dépôt (mention INT) ou de professionnel (autres mentions) on incrémente la case adéquate.

On trace enfin le graphe de proximité et la carte de proximité si l'utilisateur·ice l'a demandé, en appelant TracerVentes().

vi. lire_conversion_sacherie_GCO()

Cette fonction a pour utilité de fournir une liste permettant de passer des codes usines donnés dans le fichier d'extraction des ventes, aux codes GCO utilisés pour chaque usine individuellement. Cette liste est écrite dans le premier onglet des Entrées Manuelles. Il s'agit donc ici d'une fonction de lecture de fichier Excel assez immédiate, voir les Structures récurrentes pour plus de détails.

vii. optimiser_livraisons()

Dans le cas d'une livraison groupée, on travaille avec une liste de sous-livraisons, et le but est d'obtenir l'ordre dans lequel parcourir les points afin de minimiser la distance à parcourir. Cela se fait en trois étapes.

optimiser_livraisons()

La fonction optimiser_livraisons a pour but de transformer la liste des sous livraisons en une liste exploitable par les fonctions qui suivent.

D'abord, le point de départ est les coordonnées GPS de l'usine de dépôt. Si on ne le trouve pas, on le signale à l'utilisateur·ice et on attribue le code postal de St-Mars.

Le but est ensuite de remplir un tableau des coordonnées et de la masse à livrer à chaque point. Le premier point est le dépôt, avec une masse à livrer nulle, temporairement.

Ensuite, on parcourt la liste des sous-livraisons, et, pour chacune où la masse à livrer n'est pas nulle, on ajoute à la liste. Cependant, si on passe déjà par un point (c'est-à-dire deux livraisons de même code postal), on fusionne les deux points identiques. En effet, avant d'ajouter le point et sa masse à la liste, on vérifie d'abord que son code postal n'est pas dans la liste des codes postaux déjà visités. Si ce n'est pas le cas on ajoute le point à la liste des points, qui prend la forme suivante :

Tableau 34 Variable coordonnees_et_masse

coordonnees_et_masse	Longitude (rad)	Latitude (rad)	Masse à livrer
Indice	0	1	2
Type	Float	Float	Float

On incrémente également de la masse à livrer à chaque point la cellule d'indice [0][2] qui correspond à la masse au point de départ, l'usine. Ce sera la masse avec laquelle le camion partira et commencera le trajet.

Pour fusionner deux points de code postal identique, on incrémente simplement la masse (et la masse de départ en cellule [0][2]) sans ajouter une nouvelle ligne.

Ensuite, si la liste des points est de longueur strictement supérieure à 1, c'est-à-dire s'il y a plus de point que seulement le point usine, alors on passe à la phase de parcours.

Il peut arriver que la liste coordonnees_et_masse ne contienne que le pont de départ à l'usine, dans le cas où tous les autres points y ont été fusionnés. Cela ne peut arriver que si la livraison groupée faisait l'objet uniquement de client·es localisé·es au même code postal de l'usine.

Dans ce cas, le bilan carbone est virtuellement nul, car la distance est nulle, mais ce cas de figure n'arrive en pratique jamais jusqu'alors.

parcours()

La fonction parcours() a le double but de calculer le bilan carbone trajet par trajet, et, éventuellement, de tracer les livraisons optimisées.

On trace d'abord les points de livraisons, puis on fait appel à la fonction solve_tsp_dynamic() qui renvoie l'ordre dans lequel on doit parcourir les points pour optimiser la livraison en distance.

On ajoute à ce trajet un retour au point de départ, même si le bilan carbone de cette étape est virtuellement nul. On initialise ensuite les compteurs de distances, de distance*tonnage, et de charge, respectivement à zéro, zéro, et somme de toute les masses à livrer, contenue dans la cellule [0][2] du tableau coordonnees _et_masses.

La variable prec, pour précédent, contient l'indice du point précédent le prochain où l'on va, c'est-à-dire celui où l'on est. Cette variable est donc initialisée à 0, donc le point de l'usine de départ. Ensuite, on parcourt la liste du chemin optimisé, en avançant d'un point à chaque fois.

On calcule la distance, par hypothèse des petits angles, par une formule de pythagore pour estimer la distance qu'il y a entre deux coordonnées GPS. Cela se fait par l'intermédiaire de la fonction distance() qui intègre au passage le facteur de conversion route[21].

On a donc la distance, et pour connaitre la masse de chargement du camion sur cette portion de trajet, on garde en compte dans la variable *charge*, que, à chaque étape, on livre, donc on se libère, d'une masse connue.

Pour plus de détails sur ce cheminement, se référer au Guide Méthodologique.

solve_tsp_dynamic()

Le problème que l'on cherche à résoudre, celui d'un voyageur qui doit parcourir un certain nombre de points en en minimisant la distance, est ancien, bien connu, et pourtant il n'existe pas d'algorithme universel pour le résoudre. Il existe donc une variété d'approche qui permette de donner une solution suffisamment bonne, et ici on utilisera une approche de programmation dynamique. Le procédé suivi consiste à réduire le problème en plusieurs sous-problèmes, jusqu'à atteindre des sous-problèmes immédiats à réduire. La démarche est expliquée ici[22]. L'implémentation en code est reprise de l'utilisateur·ice Mlalevic sur GitHub.

Le résultat est visuellement satisfaisant, et permet d'avoir, en un temps raisonnable, une bonne approche de la solution, dont on donne deux exemples tirés de l'exercice 2019 dans la Figure 7 Exemples du résultat d'optimisation en distance des livraisons groupéesFigure 6 Identification de la quantité dans le nom d'un sac.

[21] Pour passer d'une distance en ligne droite à une distance de route effective

[22]https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_travelling_salesman_problem.htm

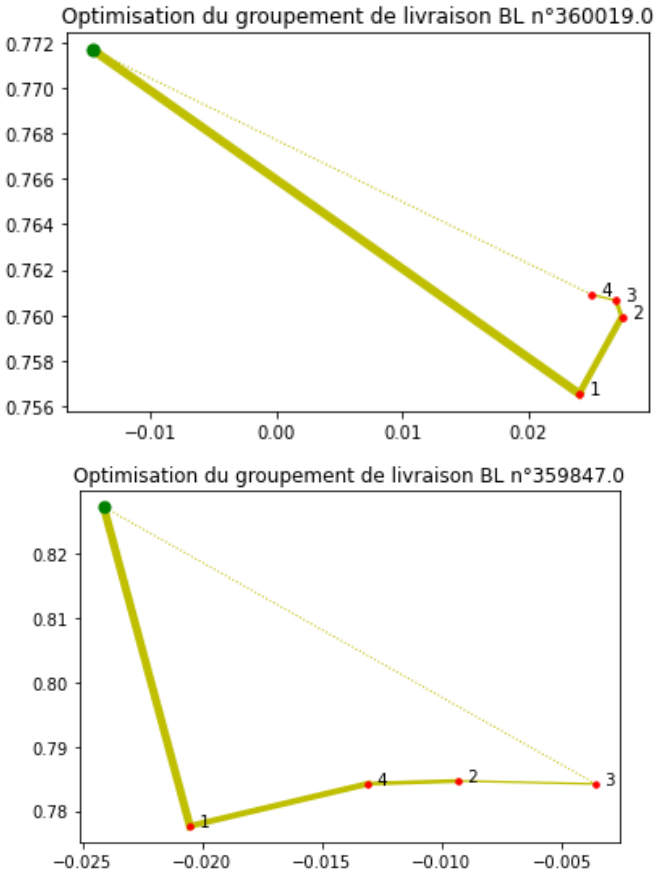


Figure 7 Exemples du résultat d'optimisation en distance des livraisons groupées

Cependant, cette approche ne saurait être appliquée pour minimiser le bilan carbone, elle ne peut l'être que pour minimiser la distance. Pour plus de détails encore une fois, se référer au Guide Méthodologique qui explicite les raisons de cette limite.

viii. TracerVentes()

e. autres.py

Comme son nom l'indique, le module « autres » prend en charge les calculs des postes n'ayant pas de module dédié. Il s'agit donc de l'énergie (Electricité et Fuel), des immobilisation et des déplacements d'individus.

a. Electricité

L'électricité se calcule en trois étapes. D'abord, la fonction lire_FE_elec() lit le facteur d'émission de l'électricité dans l'onglet « Autres FE » du fichier des entrées manuelles. La fonction lire_conso_elec() fait de même avec les consommations par site à l'année voulue. Le fonctionnement de ces deux fonction est immédiat, se référer aux Structures récurrentes pour plus de détails.

Le résultat de lire_conso_elec() est un tableau comme celui-ci-après.

La fonction calc_BC_elec prend en entrée le FE de l'électricité lu précédemment et le tableau des consommation. Elle parcourt ce dernier en ajoutant à chaque ligne le produit de la consommation avec le FE, en convertissant en tCO2e, le FE étant en $kgCO_{2e}$/kWh et la consommation étant en kWh.

Une exception est faite pour les sites équipés de panneaux photovoltaïque, ce qui ne concerne que Lavilledieu à l'heure actuelle. Si l'année de calcul est supérieure à 2017 et que l'on est au poste de Lavilledieu, alors on utilise le FE de la production photovoltaïque enregistré comme variable globale.

```
def calc_BC_elec(FE_elec:float,conso:list):
    for x in conso:
        if x[0][2].lower()=="lavilledieu" and p.ANNEE >2017:
            x.append(p.FE_photovoltaique*x[1]/1000)
        else:
            x.append(FE_elec*x[1]/1000) #Conversion en tonnes
    return conso
```

Le résultat est ensuite enregistré par resultat.py.

b. *Fuel*

Le fonctionnement est tout à fait similaire à celui des fonctions dédiées à l'électricité, ci

Le fuel se calcule en trois étapes. D'abord, la fonction lire_FE_fuel() lit le facteur d'émission du carburant dans l'onglet « Autres FE » du fichier des entrées manuelles. La fonction lire_conso_fuel() fait de même avec les consommations par site à l'année voulue. Le fonctionnement de ces deux fonction est immédiat, se référer aux Structures récurrentes pour plus de détails.

Le résultat de lire_conso_fuel() est un tableau comme celui-ci-après.

La fonction calc_BC_fuel prend en entrée le FE du fuel lu précédemment et le tableau des consommation. Elle parcourt ce dernier en ajoutant à chaque ligne le produit de la consommation avec le FE, en convertissant en tCO2e, le FE étant en $kgCO_{2e}$/m3 et la consommation étant en m3.

Le résultat est ensuite enregistré par resultat.py.

c. *Immobilisations*

Les données concernant les immobilisations, c'est-à-dire les émissions dues aux bâtiments étalées sur une certaine durée d'amortissement, sont suffisamment fixes pour ne pas demander de calcul trop intensif. Les données sont regroupées dans l'onglet d'indice 6 du fichier des Entrées Manuelles.

Le code ne lit pas les tableaux, qui sont assez explicites par ailleurs, mais uniquement les cellules J14, J22, J30, J38 etc., ce qui correspond au total amorti pour chaque usine. Le tout est enregistré dans un tableau simple de la forme suivante.

Tableau 35 Variable immobilisations

Immobilisations	Site	Total amorti (tCO2e)
Indice	0	1
Type	Str	Float

d. *Déplacements*

f. resultat.py

Le module résultat a pour but d'enregistrer les données calculées dans une matrice générale enregistrée comme une variable globale avec le nom MATRICE_RESULTAT. Il permet aussi le regroupement des résultats selon des postes plus globaux.

a. *Création de la matrice résultat*

La fonction creerMatriceResultat() permet de préparer la variable liste MATRICE_RESULTAT à y accueillir le total de chaque poste et de chaque site. Elle est organisée de la manière suivante.

Tableau 36 MATRICE_RESULTAT à sa création

Poste 1	Poste 1	Poste 1	…	Poste 2	…
Site1	Site 2	Site 3	…	Site 1	…
0	0	0	0	0	0
tCO2e	tCO2e	tCO2e	tCO2e	tCO2e	tCO2e
0	0	0	0	0	0
tCO2e/m3	tCO2e/m3	tCO2e/m3	tCO2e/m3	tCO2e/m3	tCO2e/m3

Il s'agit ensuite simplement de deux boucles imbriquées pour remplir les cases, en laissant à 0 celles qui seront complétées ultérieurement.

b. *Enregistrement d'un poste*

Toutes les fonctions enregistrant les résultats dans la matrice résultat se ressemblent beaucoup, et pourraient presque faire l'objet d'une et unique fonction. Prenons alors un cas quelconque : on souhaite enregistrer les résultats du calcul d'un poste, et ces résultats se présentent sous la forme suivante :

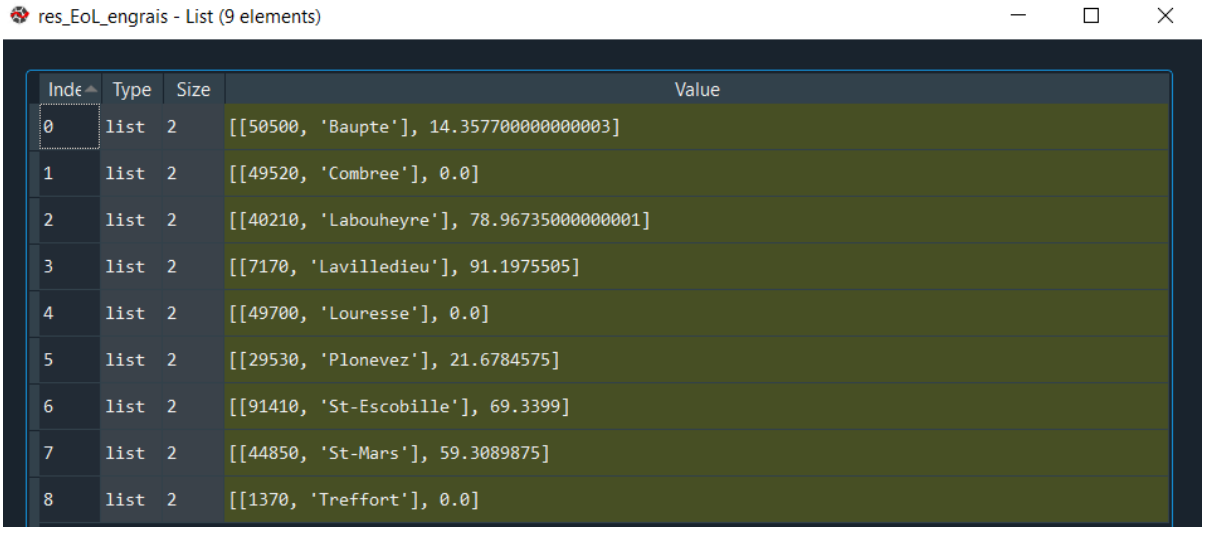


Figure 8 Résultat sur la fin de vie des engrais (2019)

Le nombre contenu dans la case 2 de chaque ligne représente les émissions du poste annuelles par usine. La forme de la fonction qui l'enregistre est la suivante :

```
def enregistre_eol(prod_par_site:list,MP_par_usine:list,massesvols:list, res_EoL_engrais
    i = 0
    while i<len(p.MATRICE_RESULTAT):
        poste = p.MATRICE_RESULTAT[i][0]
        if poste=="Protoxyde d'azote" and p.MATRICE_RESULTAT[i][1][0]!=-1:
            for usine in res_EoL_engrais:
                if usine[0][1]==p.MATRICE_RESULTAT[i][1][1]:
                    p.MATRICE_RESULTAT[i][2][0]=usine[1]
                    p.MATRICE_RESULTAT[i][2][1]='teCO2'
                    for u in prod_par_site:
                        if u[0][2]==usine[0][1]and u[1]!=0:
                            p.MATRICE_RESULTAT[i][2][2] = (usine[1])/u[1]
                            break
                    p.MATRICE_RESULTAT[i][2][3] = "teCO2/m3"
                    break
        i+=1
```

Figure 9 Fonction d'enregistrement du résultat sur la fin de vie des engrais

On parcourt toute la matrice résultat poste par poste du début à la fin. Dès que l'on trouve un poste qui correspond à ce que l'on veut enregistrer (ici « Protoxyde d'azote »), on itère sur chaque usine du résultat à enregistrer. Si l'on trouve l'usine correspondante à la ligne de la matrice résultat, on enregistre la valeur (usine[1]). Ensuite, on va rechercher la production de cette usine sur cette année dans la matrice prod_par_site. Là encore, s'il y a correspondance, on récupère la production de l'usine en m3 dans la case u[1] et on l'enregistre.

On note la présence de *break* à chaque fin de boucle *for*. Comme expliqué dans les Structures récurrentes, cela permet d'arrêter de parcourir une liste dès lors que l'on a trouvé l'élément qui nous intéressait.

c. *Regroupement des postes*

La fonction regrouper_postes_resultat permet d'obtenir en résultat une matrice plus lisible que la MATRICE_RESULTAT, en regroupant, toutes usines confondues, certains postes.

Les postes regroupés sont les suivants :

Tableau 37 Regroupement Des Postes

Fret amont	"MP bateau" ; "MP route"
Fret aval	"Ventes pro" ; "Ventes Jardineries" ; "Inter dépôt"
Sacherie	"PE neuf" ;"PE recyclé" ;"PVC" ;"Carton"; "Autres Emballages" ;"Fret amont des emballages"
Mat. Prem.	"Matières premières"
Energie	"Electricité" ; "Fuel"
Utilisation/EoL	"CO2 issu de la tourbe" ;"Protoxyde d'azote"
Autre	"Déplacements", "Immobilisations"
Total	Total

Chacun de ces postes est décliné en deux chiffres : le total pour toutes les usines, et le total par production. Le résultat est le suivant :

Tableau 38 Matrice des résultats groupés

« »	"Emissions totales (tCO2e)"	"Emissions ramenées à la production (tCO2e/m3)"
Fret amont	0	0
Fret aval	0	0
Sacherie	0	0
Mat. Prem.	0	0
Energie	0	0
Utilisation/EoL	0	0
Autre	0	0
Total	0	0

Reste ensuite simplement à parcourir la MATRICE_RESULTAT pour remplir les cases vides.

7. Sorties du programme

a. Résultat

La production principale de ce programme est un fichier Excel à plusieurs onglets, généré automatiquement.

Note : certains graphes voire onglets peuvent manquer dans le cas d'une exécution partielle du bilan carbone, ce qui est tout à fait normal.

i. Résultats généraux

Cet onglet contient principalement la matrice MATRICE_RESULTAT telle quelle, ainsi que le total du bilan carbone.

Le tableau principal est directement illustré en diagramme à barre, et, pour chaque poste, on donne le total annuel, et le total ramené à la production du site.

ii. Résultats détaillés

iii. Erreurs et informations

iv. Codes postaux à corriger

v. Assimilations de MP

vi. Log Fret par Matière Première

b. Graphes
c. Console
d. Résoudre les messages d'erreur
e. Log

8. Maintenance

On peut séparer trois niveaux de maintenance, selon la complexité de la modification à effectuer :

- Niveau bas : Modification des paramètres de calcul
- Niveau moyen : Modification du fichier des entrées manuelles
- Niveau haut : Modification du code python

a. Maintenance niveau bas

La maintenance de bas niveau est accessible en modifiant le fichier « Paramètres de calcul ». Ce fichier regroupe des entrées utilisateur·ice qui peuvent être modifiées pour obtenir divers calculs. L'ordre des lignes dans ce fichier n'est pas importante, mais il faut conserver les mêmes colonnes. Les seules cases à modifier sont celles de la seconde colonne « Valeur ». Ces valeurs sont lues par le programme pour fixer certaines variables.

La colonne 1 est une indication de ce que fait la variable.

La colonne 2 est la valeur de la variable.

La colonne 3 est le nom de la variable, et ne doit pas être changé.

La quatrième et dernière colonne est le type d'information à rentrer dans la seconde colonne :

- Int : Nombre entier
- Float : Nombre à virgule
- String : Chaine de caractères (lettres et symboles)
- Bool : booléen (True ou False)

Si une variable n'est pas trouvée ici alors qu'elle devrait y être, elle sera remplacée par une valeur par défaut inscrite dans le programme python.

Les changements faits dans le fichier ne sont pas pris en compte par le programme tant que ceux-ci ne sont pas sauvegardés. Veiller donc à sauvegarder le fichier avant de le lancer. Il n'est en revanche pas nécessaire de le fermer.

i. Changer l'année de calcul

Pour effectuer le calcul sur une autre année, changer la seconde cellule de la ligne « année de calcul ». En notant 20XX l'année de calcul considérée l'année prise comme référence est la période :

Du 1er juillet $20XX_{-1}$ (année précédente) au 30 juin 20XX

Il est important, en choisissant l'année de calcul, de s'assurer que le reste des fichiers est bien présent pour cette année de calcul. Par défaut, il faut aller voir dans « Données/20XX/ » si l'on a bien les trois fichiers : Intrants 20XX.xlsx ; Conso sacherie 20XX.xlsx et Ventes 20XX.xlsx.

ii. Changer les emplacements des fichiers

L'emplacement des fichiers est crucial pour le bon fonctionnement du programme, qui ne tournera pas normalement si tous ne sont pas trouvés au bon endroit, ou si on lui demande d'éditer le fichier de résultat dans un emplacement inexistant.

Les chemins spécifiés pour la lecture (Direction du fichier des entrées manuelles, Direction du fichier de géolocalisation, Direction du fichier des intrants (achats de matières premières), Direction du fichier des extrants (ventes de produits finis) et Direction du fichier des achats de sacherie sont relatif au dossier d'exécution du programme, c'est-à-dire à où se trouve le .exe ou le main.py. Il est recommandé de ne pas y toucher et de laisser l'arborescence telle quelle.

Dans les chaines de caractères qui s'y trouvent, ANNEE sera remplacé automatiquement par l'année spécifiée plus haut.

Le chemin qui est la valeur de CHEMIN_ECRITURE_RESULTATS, en revanche, est un chemin absolu. C'est là où le programme publiera le fichier Excel de résultat. Il faut donc qu'il s'agisse d'un emplacement où le programme a le droit d'écrire, ce qui n'est pas le cas, par sécurité, des emplacements réseau de Florentaise. Il faut donc sauvegarder un emplacement local.

Pour le faire écrire par exemple dans un dossier nommé « Résultat » qui se trouverait sur le bureau, la marche à suivre est la suivante :

1. Créer le dossier sur le bureau
2. L'ouvrir
3. Cliquer droit sur l'adresse et la copier en tant que texte
4. Coller le contenu dans la cellule et **important : y ajouter un anti-slash (alt gr+8) à la fin**.

On procèdera de même pour tout emplacement local.

iii. Changer les postes calculés/afficher des graphes/afficher les messages de console

Il s'agit ici simplement de changer les booleens de la ligne correspondante, ce qui est assez immédiat.

Les booleens avec la valeur True se colorent en vert, et ceux à False sont en orange. Il est important d'y mettre une majuscule au début, et que la case soit d'une couleur ou de l'autre, mais pas blanche, ce qui signifie que la valeur ne sera pas correctement lue par le programme.

b. Maintenance niveau moyen

i. Calculer une nouvelle année de bilan carbone

Pour calculer une nouvelle année de bilan carbone, il est nécessaire de s'assurer de plusieurs choses :

1. Que les fichiers pour le calcul de l'année sont bien présents au bon endroit et avec les bonnes colonnes

Il faut trois fichiers annuels pour effectuer un calcul. Ils sont rangés dans « Données/20XX/ » et sont nommés Intrants 20XX.xlsx, Conso sacherie 20XX.xlsx et Ventes 20XX.xlsx.

Se référer à la partie Fichiers Données pour s'assurer que les colonnes issues de l'extraction GCO sont bien les bonnes.

Les lignes d'extraction GCO pour les obtenir sont les suivantes :

2. Que les données annuelles sont bien remplies

Il s'agit là des tableaux dans le premier onglet du fichier « entrées manuelles » :

Chiffre d'affaire (unique pour tous les sites):

Nb salarié·es :

Production, en m^3 :

Consommation de fuel, en m^3 :

Consommation électrique

3. Que le catalogue de sacherie est suffisamment récent

4. Que l'année spécifiée dans les Paramètres de calcul est bien celle désirée

ii. Ajouter un fournisseur international

iii. Ajouter/mettre à jour un catalogue de sacherie

Lorsqu'un nouveau catalogue de sacherie est achevé, il faut l'ajouter à la liste des sacheries avant de faire le calcul d'une année récente. Attention : il ne faut pas supprimer le catalogue préexistant, utile pour les bilans carbones des années antérieures ! Il faut l'ajouter à la liste, à la suite de ce qui existe déjà.

iv. Modifier les facteurs d'émission

Les facteurs d'émission sont répartis sur deux onglets du fichier Entrées Manuelles :

v. Ajouter une usine

Pour ajouter une usine, il faut ajouter une ligne à tous le tableaux de l'onglet 0 du fichier Entrées Manuelles.

c. Maintenance haut niveau

Pour de la maintenance de haut niveau, il faut connaître ou se former un minimum au code python, au risque de créer davantage d'erreurs.

i. Ajouter des panneaux solaires à un site

Liste des tableaux

Tableau 1 Variable bdd_exp_terre 16
Tableau 2 Variable bdd_exp_mari 16
Tableau 3 Variable cp_sites_flor 16
Tableau 4 Variable listeAchats 17
Tableau 5 Variable fret1 (incomplete) 17
Tableau 6 Variable resultat_usine_MP 18
Tableau 7 Variable fret2 (incomplete) 20
Tableau 8 Variable FE_familles 21
Tableau 9 Variable MP_familles_N 21
Tableau 10 Variable masse_vol_MP 21
Tableau 11 Variable FE_engrais 21
Tableau 12 Variable fret3 (incomplete) 22
Tableau 13 Variable fret finale 23
Tableau 14 Variable usines_et_fret 24
Tableau 15 Variable refs_sacheries 25
Tableau 16 Variable conso_materiaux_par_site 25
Tableau 17 Variable resultat_usine_MP 30
Tableau 18 Variable import_terrestre 33
Tableau 19 Variable import_maritime 34
Tableau 20 Exemples de distances de Levenshtein 37
Tableau 21 Variable reference_materiaux 43
Tableau 22 Variable conso_materiaux_par_site 43
Tableau 23 Variable fret_sacherie_par_usine 44
Tableau 24 Variable conso_materiaux_par_sites 44
Tableau 25 Variable bdd_exp_terre 45
Tableau 26 Variable bdd_exp_mari 45
Tableau 27 Sous-liste des ports 45
Tableau 28 Varibale bdd_coutkm_dist 45
Tableau 29 Variable bdd_coutkm_E 45
Tableau 30 Variable listeVentes 46
Tableau 31 Variable livraisons 46
Tableau 32 Variable de sous-livraison 47
Tableau 33 Variable BC_fret_aval_par_usine 48
Tableau 34 Variable coordonnees_et_masse 49
Tableau 35 Variable immobilisations 52
Tableau 36 MATRICE_RESULTAT à sa création 53
Tableau 37 Regroupement Des Postes 54
Tableau 38 Matrice des résultats groupés 55

Liste des illustrations

Figure 1 Exemple d'une lecture de tableau Excel avec lireImportMaritime() 6
Figure 2 Exemple de recherche de nom avec la fonction recherche_elem() 7
Figure 3 Exemple de recherche insensible à la casse 8
Figure 4 Exemple d'ouverture d'un fichier Excel 9
Figure 5 Exceptions dans la fonction gestionpb_volume_sac 28
Figure 6 Identification de la quantité dans le nom d'un sac 29

Figure 7 Exemples du résultat d'optimisation en distance des livraisons groupées..... 51
Figure 8 Résultat sur la fin de vie des engrais (2019)..... 53
Figure 9 Fonction d'enregistrement du résultat sur la fin de vie des engrais..... 54