Rapport de programmation
Réalisation d'un programme de calcul des effets du trafic sur un ouvrage

Charlotte Wolff

Commanditaire : Franziska Schmidt

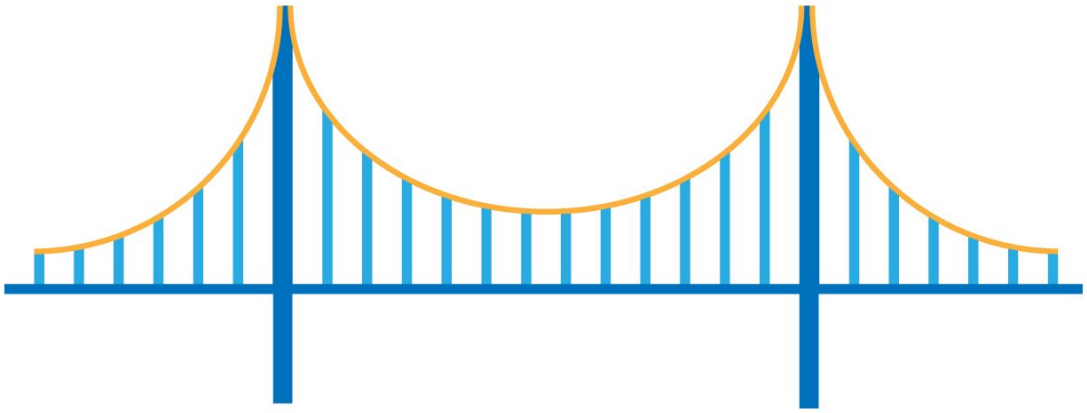

Table des matières

1 INTRODUCTION

Après 15 séances de programmation, le projet demandé par les commanditaires a été codé, en suivant l'analyse qui en avait été faite pendant le mois de décembre. Le présent rapport présente les résultats de la programmation, notamment ce qui a été réalisé comme convenu et les tâches qui n'ont pas été achevées, ainsi que les difficultés rencontrées pendant le développement. Une documentation utilisateur et programmeur viennent compléter cette analyse de fin de programmation.

2 Résultats de la programmation

2.1 Rappel du sujet

Le projet intitulé « Réalisation d'un programme de calcul des effets du trafic sur un ouvrage » a été commandé par Franziska Schmidt de l'IFFSTAR. Le but est de reprendre un ancien code réalisé en FORTRAN permettant le calcul des effets du trafic routier sur un ouvrage, dont on connait la ligne d'influence, au cours du temps.
Les grandes étapes qu'il est demandé de réaliser sont :

- Filtrage du fichier trafic pour enlever les données aberrantes
- Calcul des effets du trafic au cours du temps et enregistrement dans un fichier
- Affichage des histogrammes de fréquence choisis par l'utilisateur, donnant des informations sur le trafic
- Présentation sous forme d'un interface graphique

Le tout a été réalisé en Python et utilise les bibliothèques suivantes :

- MatplotLib
- Numpy
- Datetime
- Csv
- Pyqt4

2.2 Le déroulement de la programmation

L'ordre des étapes de programmation présentées dans l'analyse a été respecté. Les premières séances ont été utilisées pour créer les classes nécessaires au filtrage des données et au calcul des effets du trafic puis les méthodes et fonctions associées. Les différentes classes créées sont écrites dans des fichiers différents, de même que les fonctions permettant l'acquisition des données.

Les mêmes classes que celles présentées dans l'analyse ont été codées, plus une à laquelle je n'avais pas songée, contenant toutes les informations sur les paramètres de filtrage. Cette classe ne possède pas de méthode (à l'exception d'une méthode permettant d'afficher tous les attributs) mais permettra de récupérer facilement les paramètres de filtrage.

Puis j'ai installé les bibliothèques nécessaires à la création de l'interface graphique, à savoir :

- Pyqt4
- Pyuic4

Dans un troisième temps, une interface graphique très simple a été réalisée sous qtCreator et le fichier d'extension .ui a été converti en un fichier python en utilisant Pyuic4. En utilisant ensuite pyqt4, j'ai pu ouvrir l'interface sous python.

L'étape suivante a donc été de créer toutes les fonctions connectant les boutons présents sur l'interface aux fonctions permettant le filtrage des données aberrantes et le calcul des effets en fonction des fichiers trafic et ligne d'influence choisis.

En parallèle de cette étape, j'ai également comparé les résultats obtenus avec mon programme avec ceux obtenus avec l'ancien programme : Mon commanditaire m'a fourni un fichier texte contenant les données d'histogramme des effets pour un jeu de donnée test et j'ai comparé avec mon histogramme pour le même jeu de donnée. On obtient des fréquences similaires avec les deux programmes. L'algorithme codé semble donc fonctionner correctement.

Durant la programmation, des copies des fichiers ont été déposées sur un dépôt en ligne Github.

2.3 Bilan sur le code

A la fin du développement du projet informatique, le projet tourne, avec une interface graphique permettant de renseigner le chemin du fichier trafic et du fichier de la ligne d'influence, les paramètres nécessaires au filtrage des données et au calcul des effets. L'utilisateur peut également choisir les histogrammes qu'il veut afficher. S'il souhaite calculer l'effet du trafic sur le pont, il doit renseigner dans quel fichier il veut enregistrer le résultat. Un bouton *Valider* permet de lancer les calculs et d'afficher les histogrammes. Si le programme détecte une erreur, une fenêtre s'ouvre informant l'utilisateur du problème. Il peut s'agir d'un chemin de fichier mal renseigné, aucun histogramme coché, un problème dans le format du fichier trafic, un problème dans les paramètres.

De plus, l'interface possède un bouton *Historique* permettant de remplir automatiquement les champs avec les informations rentrées lors du dernier calcul, et un bouton *Défaut* qui remplit les champs avec des paramètres par défaut.

Les paramètres par défaut sont renseignés dans un fichier *.txt* que l'utilisateur peut donc configurer lui-même en choisissant les paramètres par défaut qu'il veut.

Une liste de checkbox permet de choisir les histogrammes à afficher. Lors que les histogrammes s'affichent, l'utilisateur peut choisir de les enregistrer ou non, cela ne se fait pas de manière automatique.

Enfin, un bouton *Aide* permet de rappeler le format des premières colonnes du fichier trafic à respecter.

Pour gagner du temps de calcul, les effets ne sont pas calculés à toutes les dates t. Le calcul débute à l'arrivée du premier camion sur le pont. Lorsqu'il n'y a plus de camion sur le pont, la prochaine date de calcul est alors celle de l'arrivée du prochain camion sur le pont. Les calculs s'arrête lorsque le dernier camion enregistré dans le

fichier trafic est passé sur le pont.

2.4 Exemple de l'interface

L'interface se présente de la forme suivante (Figure 1) :

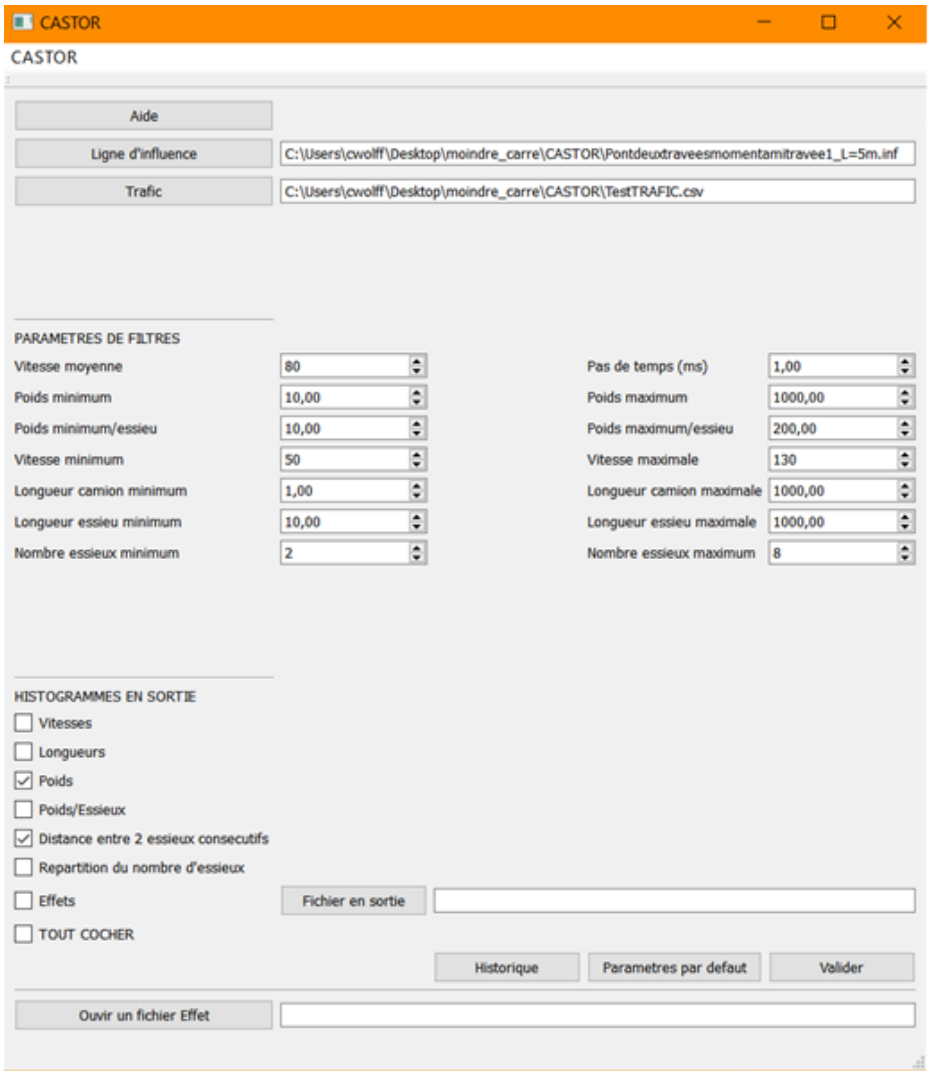

FIGURE 1 – Liste des boutons et leurs fonctionnalités

Lorsqu'un calcul est lancé, en fonction de la longueur du fichier trafic, et du pas de temps choisi, le temps peu être long pour afficher les histogrammes. En fin de calcul, la console affiche les informations présentées dans la Figure 2.

Les différents histogrammes que l'on peut afficher sont présentés dans la Figure3.

2.5 Les poursuites du code

Le code à l'état actuel peut être amélioré de diverses façons. Notamment, il peut être enrichi en proposant d'autres histogrammes de fréquence tel que :

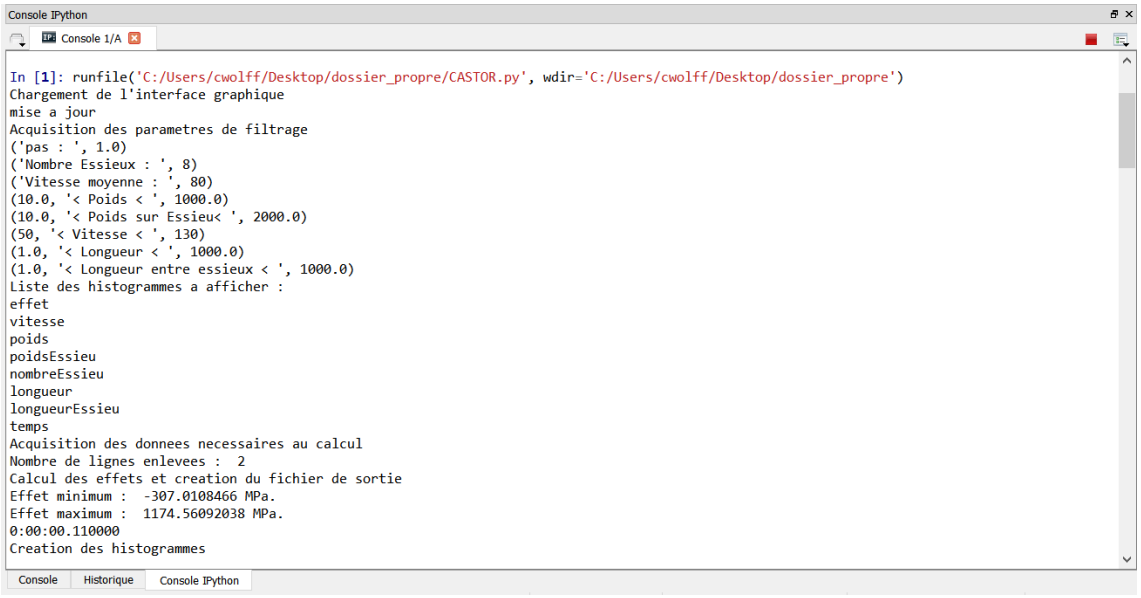

FIGURE 2 – Console python, avec les données affichées lors d'un calcul

- Histogrammes des durées entre l'arrivée entre 2 camions consécutifs
- Le nombre de véhicules par heure en fonction de l'heure de la journée

Cette liste n'est pas exhaustive. Une personne extérieure pourra donc coder ces histogrammes. La documentation programmeur est donc indispensable à la reprise du code.

De plus, pour chaque histogramme, il serait intéressant de calculer et d'afficher les valeurs minimale et maximale et des statistiques sur les valeurs. Cela pourrait être affiché dans une nouvelle fenêtre qui s'ouvre lorsque l'on a appuyé sur le bouton *Valider*.

L'interface peut également être complétée avec d'autres paramètres tels que des heures de la journée où l'on ne fait pas les calculs et des paramètres pour l'affichage des histogrammes. Le fichier *defaut.txt* contient les paramètres de filtrage mais est difficilement lisible pour le modifier. Il serait intéressant de rajouter les lignes permettant de comprendre à quoi correspond chaque valeur.

De plus, le programme pourra être complété en intégrant le trafic dans les deux sens de circulation du pont.

Enfin, pour simplifier l'utilisation du code par le plus grand nombre, il serait intéressant de mettre tous les codes dans un dossier pour ne garder que le fichier *CASTOR.py* hors de ce dossier et d'y faire appel. Cela n'a pas été fait car j'ai eu des difficultés à importer le fichier contenant le squelette de l'interface. D'autre part, de créer un exécutable pour le programme.

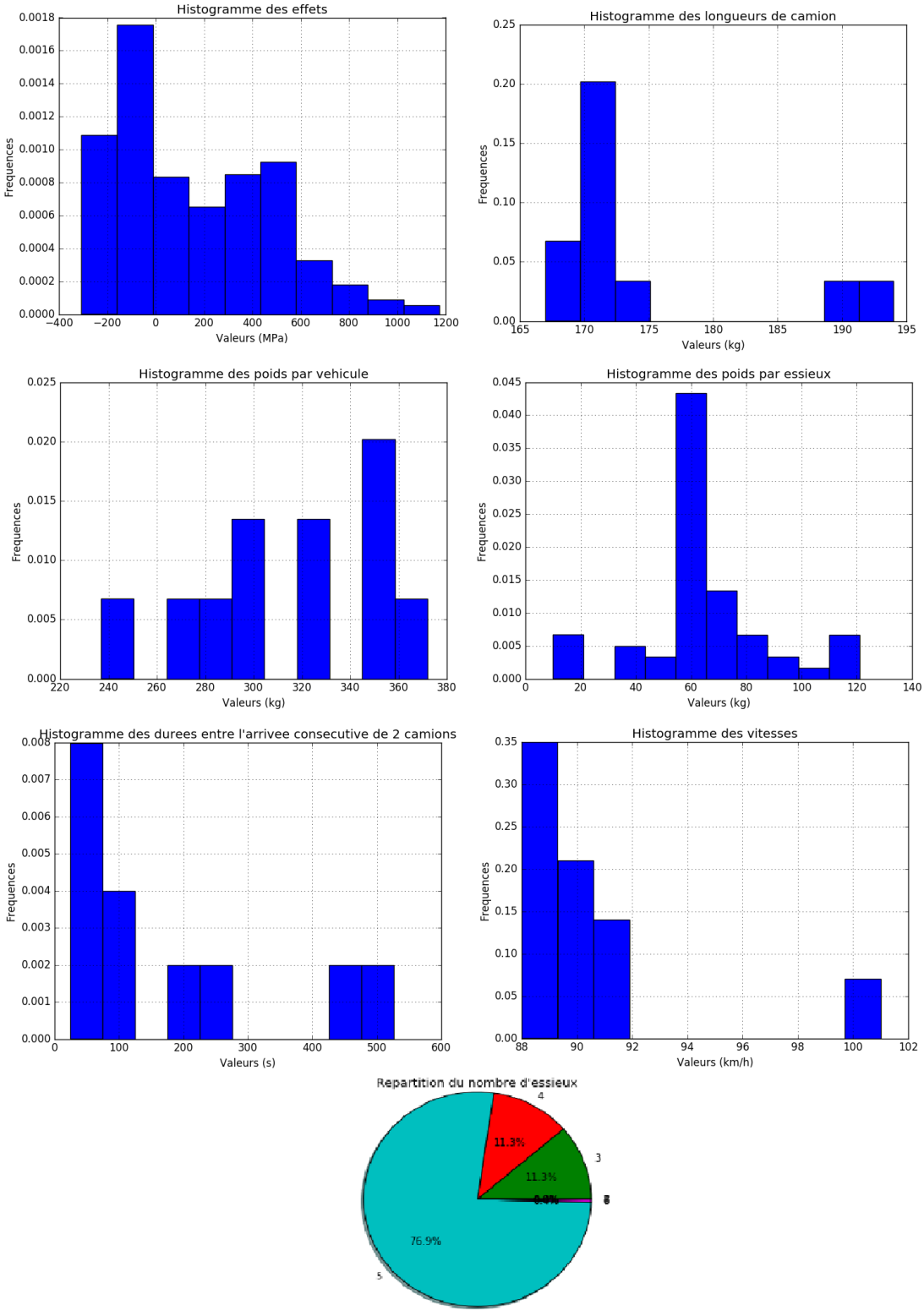

FIGURE 3 – Exemple d'histogrammes que l'on obtient après un calcul

3 Les difficultés rencontrées

Lors du développement, différents problèmes ont été rencontrés qui ont été surmontés.

La plus grosse difficulté rencontrée a été de gérer les sauts de temps lorsqu'il n'y a plus de camion sur le pont afin d'alléger les temps de calcul. En effet, avant les sauts de temps, le programme calculait l'effet du trafic sur le pont toutes les ms, sur une période pouvant atteindre un an. Les temps de calcul pouvaient donc être extrêmement longs. Or, les ponts utilisés faisant 5m, la grande majorité du temps, il n'y a pas de camion sur le pont donc l'effet du trafic est nul.

Des problèmes auxquels je n'avais pas pensés pendant l'analyse du projet ont été soulevés. Notamment pour gérer les dates du fichier, pour les convertir en temps et les comparer les unes les autres. Pour les gérer, j'ai finalement utilisé la bibliothèque Datetime.

La création de l'interface m'a demandé également beaucoup de temps, notamment pour trouver comment convertir le fichier .ui en fichier .py. De plus, les slots permettant de connecter les boutons de l'interface aux fonctions de calculs, ne peuvent pas prendre d'attributs en entrée. J'ai donc perdu du temps à coder toutes les fonctions pour chaque bouton, alors que certains boutons font la même chose (notamment les boutons *Defaut* et *Historique*). Pour alléger le code de l'interface, j'ai donc créer un nouveau fichier python contenant toutes les fonctions auxquelles font appel les boutons.

D'autres parties du code sur lesquelles je pensais que j'allais perdre du temps ont finalement été rapidement réalisées, notamment le filtrage des données ou la fonction d'historique.

4 Documentation utilisateur

4.1 Contenu du dossier

Le dossier du projet contient :

- Un fichier **CASTOR.py** qui permet de faire tourner le programme
- Un fichier **historique.txt** qui contient les informations sur les paramètres du dernier calcul réalisé
- Un fichier **defaut.txt** qui contient les paramètres de filtrage par défaut. Ils peuvent être changés par l'utilisateur
- Des fichiers de code permettant de faire tourner l'interface, au nombre de 3
- Des fichiers permettant de faire tourner le calcul, il s'agit de la définition de différentes classes et d'un fichier avec les fonctions nécessaires aux calculs.

4.2 Installation

Afin de faire tourner le programme, l'utilisateur devra posséder python.2.7, ainsi que les différentes bibliothèques citées plus haut pour faire tourner le programme

4.3 Utilisation

Pour faire tourner le programme, il suffit ensuite d'ouvrir un terminal et de se placer dans le dossier du projet. La ligne de commande suivante (Figure 4) permet de lancer le calcul :

FIGURE 4 – Ligne de commande pour lancer le programme

Cela va ouvrir l'interface graphique permettant une utilisation facile du programme. Les différentes fonctionnalités sont présentées dans la Figure 5 :

Bouton	Fonctionnalité
Format fichier Trafic	*Ouvre une fenêtre d'information indiquant le format du fichier trafic*
Ligne d'influence	*Permet de choisir le fichier contenant la ligne d'influence*
Trafic	*Permet de choisir le fichier contenant l'enregistrement du trafic*
Fichier en sortie	*Si l'histogramme des effet est coché, il permet de choisir où stocker le calcul des effets*
Historique	*Renseigne les champs avec les informations du dernier calcul réalisé*
Defaut	*Renseigne les champs avec des paramètres par défaut*
Valider	*Lance le calcul, si tous les champs sont correctement renseignés*

FIGURE 5 – Liste des boutons et leurs fonctionnalités

5 Documentation programmeur

De même que pour l'utilisation, les bibliothèques citées plus hauts sont indispensables pour la programmation.

Le code est ensuite séparé en deux grandes parties :

- Le code pour gérer l'interface graphique
- Le code pour gérer les fichiers et le calcul

5.1 Le code pour gérer les fichiers et le calcul

Cette partie de code est séparée dans différents fichiers .py, à savoir :

- Un fichier *run Calculs.py* qui permet de faire tourner le programme sans interface graphique. Il suffit d'y rentrer les chemins des fichiers trafic, ligne d'influence et le chemin où enregistrer le fichier des effets et de lancer le programme. Les paramètres de filtrage que l'on souhaite appliquer sont à renseigner dans le fichier *classe Parametres.py*, ou directement dans le fichier *run calculs*, dans la partie du code créant les paramètres.

- Des fichiers différents pour chaque classe utilisée. Pour chaque classe sont définis des méthodes "set" permettant de mettre à jour facilement les valeurs, et une méthode affiche permettant d'afficher dans la console python les attributs de la classe. Puis pour chaque méthode de chaque classe, une première ligne indique ce que fait la méthode et une seconde indique lorsque c'est le cas, ce que renvoie la méthode. Les classes créées sont présentées dans la Figure 6.

5.2 Le code pour gérer l'interface graphique

Cette partie du code comprend les quatre fichiers suivant :

- Un fichier *interface1.ui* qui contient la structure de l'interface, créé grâce à QtCreator.

Nom du fichier	Nom de la classe	Présentation
classe_parametres	Parametres	Contient les parametres de filtrage et la liste des histogrammes à afficher
classe_Essieu	Essieu	Crée les essieux de tous les camions avec ses caractéristiques
classe_Camion	Camion	Crée les camions avec ses caractéristiques dont la liste de ses essieux
classe_Acquisition	Acquisition	Crée l'acquisition avec les colonnes du fichier trafic nécessaire au calcul et les informations de la ligne d'influence

FIGURE 6 – Liste des boutons et leurs fonctionnalités

- Un fichier *squelette interface.py* qui contient la structure de l'interface, fichier converti grâce au fichier *interface1.ui* et *pyuic4*.

- Un fichier *fonction interface.py* qui contient les fonctions permettant d'alléger les fonctions de connectivité des boutons.
- Un fichier *run interface.py* qui lie les boutons de l'interface et contient la fonction main() lançant le programme.

6 Conclusion

Pour conclure, ce projet a été extrêmement enrichissant sur beaucoup d'aspects.

- Coder seule un projet du début à la fin m'a permis d'apprendre à gérer au mieux le code en créant des classes et des fonctions et de mieux comprendre également l'intérêt de créer des classes avec des méthodes et des attributs.

- L'utilisation de bibliothèque que je ne maitrisais pas m'a obligée à me documenter seule et à faire preuve donc d'autonomie.

- Le programme étant destiné à être repris et amélioré, toutes les fonctions et méthodes doivent être rigoureusement écrites, avec notamment une méthode permettant d'afficher les attributs de la classe, les setters et les getters et de nombreux commentaires pour permettre une relecture facile du code. Ce projet m'a donc appris à être rigoureuse et précise dans l'écriture d'un code, deux qualités nécessaires pour les travaux de développement dans une équipe.

- J'ai également appris à gérer au mieux les différentes versions de code et à prendre des systématismes dans la sauvegarde des différentes versions, en utilisant github notamment et en mettant en ligne chaque nouvelle version du code.

Ainsi, ce projet a été un projet très professionnalisant.