Ce TP a pour but de vous faire implémenter l'algorithme de la **descente du gradient**. Ce TP sera réalisé en python et utilisera des notions du cours d'*introduction à la programmation orientée objet et à la documentation du code* telles que les classes et l'héritage. Attention bien que ce TP soit en python, une partie des opérations mathématiques seront réalisées avec la bibliothèque `Numpy`. Cette bibliothèque demande un peu d'apprentissage et de rigueur. En effet elle ajoute du comportement aux opérateurs mathématiques courants pour gérer du calcul matriciel. Si vous ne faites pas attention aux objets que vous manipulez votre code va rencontrer des erreurs.

1 Architecture du code

Pour vous faciliter le travail et vous donner une base de code, l'architecture suivante vous est fournie :

help : reg_data.py : Un fichier permettant de générer des données pour une régression linéaire.

numerical_methods : Va contenir les méthodes que vous allez implémenter. Pour le moment il contient uniquement un fichier `absctract_gradient_descent.py`. C'est une classe abstraite dont vos descentes de gradient vont hériter. Cette classe à deux méthodes :

descent() : qui contiendra l'algorithme de la descente ;

get_next() : qui contiendra la façon de calculer le prochain point ;

function : Va contenir les fonctions que vous aller vouloir minimiser. Actuellement ne contient que le fichier `abstract_function.py` une classe abstraite dont les classes représentantes les fonctions que l'on va étudier vont devoir hériter. Cela nous permet de savoir les méthodes qui doivent être implémentées dans les classes filles. Pour le moment :

function_definiton() : la définition de la fonction ;

gradient() : la définition du gradient de la fonction.

main.py : Le fichier où vous allez coder le script orchestrant les différentes pièces de votre code.

Vous trouverez page 2 le diagramme de classe du code que vous obtiendrez à la fin du TP.

Le code fourni utilise un package python externe, `numpy`, qu'il vous faut installer. Le code que vous venez de récupérer contient un fichier `requirements.txt` qui centralise les dépendances du code.

Une fois le code récupéré sur votre machine, ouvrez un terminal dans le dossier que vous venez de récupérer (il doit contenir le fichier `requirements.txt`). Si vous êtes

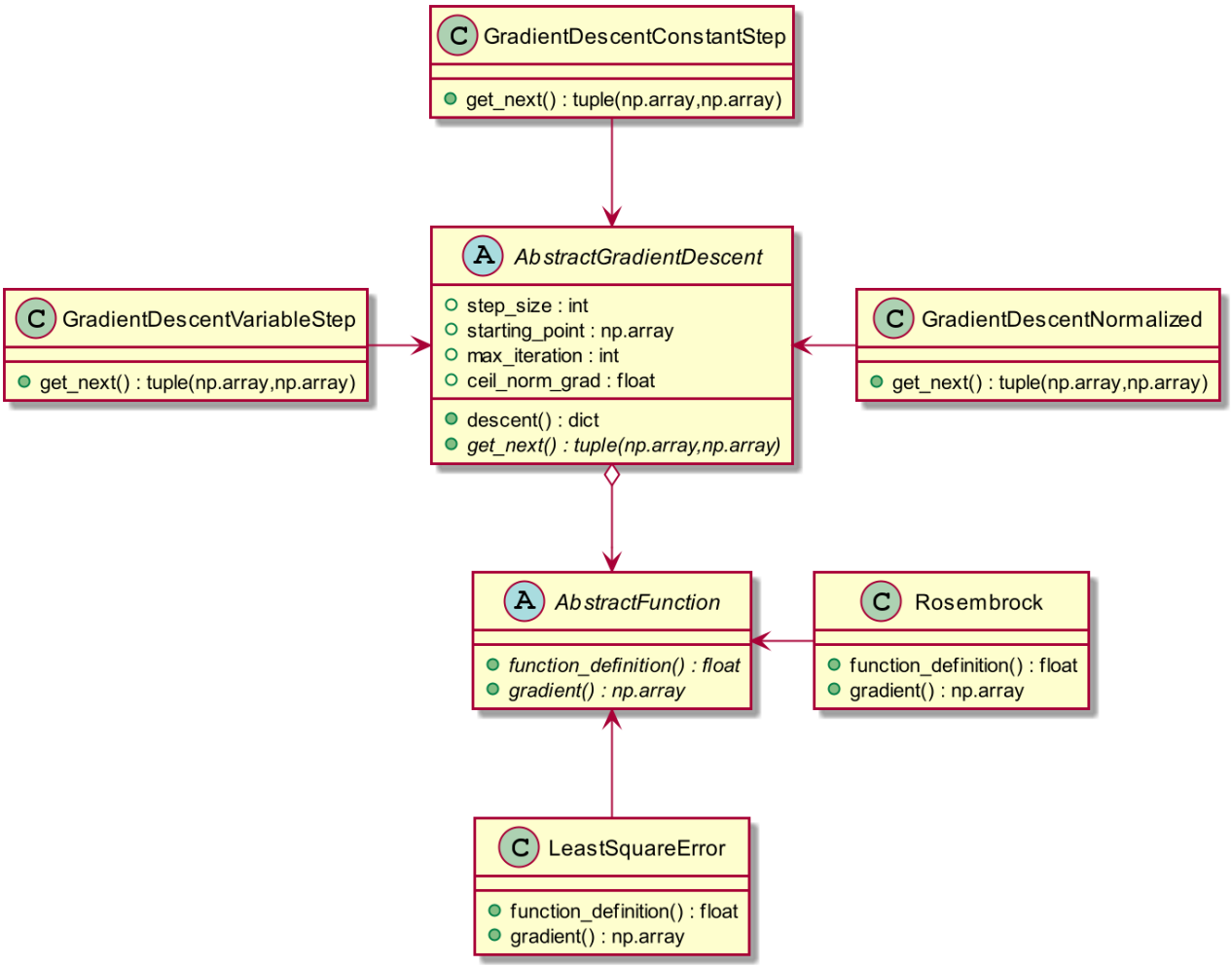

Figure 1 – Diagramme de classe du TP

sous windows, avec l'explorateur windows placez vous dans le bon dossier puis dans la barre d'adresse tapez `cmd` puis appuyer sur `entrée`. Cela ouvrira un terminal au bon endroit. Puis faites

— si vous êtes sur les VM :

```
pip install -proxy=http://pxcache-02.ensai.fr:3128 -user -r requirements.txt
```

— si vous êtes chez vous :

```
pip install -r requirements.txt
```

ou

```
pip3 install -r requirements.txt
```

si la première commande n'a pas fonctionné

Si vous êtes sous IOs vous le trouver dans l'onglet Finder, et pour les linuxien·ne·s vous devez savoir comment faire (sinon allez dans le dossier, clic droit, ouvrir terminal).

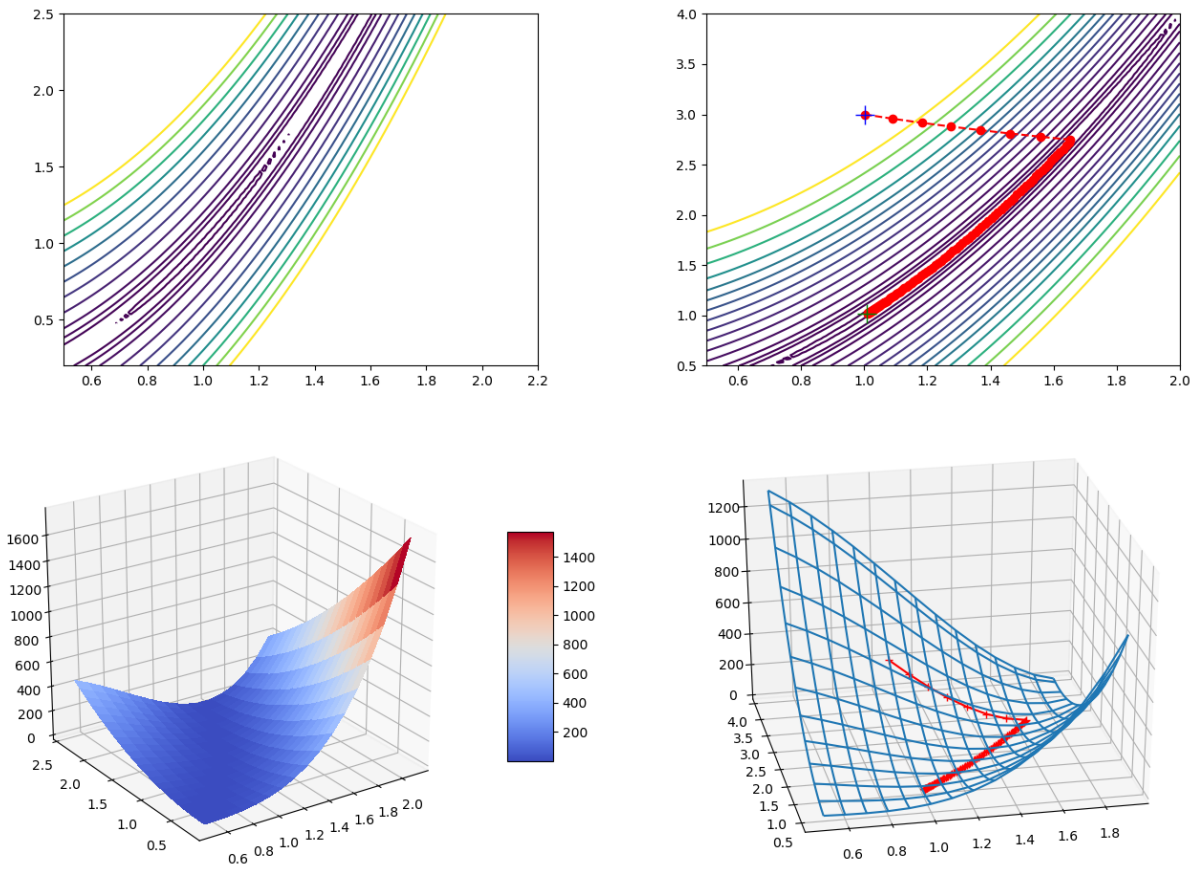

Figure 2 – Fonction de Rosenbrock et descente de gradient

2 Descente de gradient sur la fonction de Rosenbrock

2.1 Fonction de Rosenbrock

Dans cette partie vous allez chercher le minimum global de la fonction de Rosenbrock définie par :

$$f(x, y) = (x - 1)^2 + 100(x^2 - y)^2$$

C'est une fonction très souvent utilisée pour tester des algorithmes de recherche d'optimum du fait de sa forme particulière.

Question 1 : Créez un fichier `rosenbrock.py` contenant la classe `Rosenbrock`. Implémentez la méthode `function_definiton()` qui représente la fonction de Rosembrock. Elle prend un paramètre `point` qui est une liste python ou un array numpy et elle va retourner l'image de x par la fonction de Rosenbrock. Créez une fonction `gradient()` qui contient uniquement l'instruction `pass`.

Question 2 : Déterminez le gradient de la fonction de Rosenbrock. Implémentez la fonction `gradientd()` qui retourne le gradient en un point x sous forme d'un array numpy.

2.2 Descente de gradient à pas fixe

Vous allez maintenant implémenter la méthode de la descente du gradient à pas fixe.

Question 1 : Rappelez la formule de descente du gradient. Appliquez cette formule à la recherche de minimum de la fonction de Rosenbrock.

Question 2 : En déduire le pseudo-code de l'optimisation de cette fonction par la méthode de la descente du gradient. Quelles sont les conditions d'arrêt de notre algorithme ?

Question 3 : Dans le fichier `abstract_gradient_descent.py` implémentez la méthode `descent()` qui réalise la descente du gradient. Dans le fichier `gradient_descent_constant_step.py` implémentez la méthode `get_next()` qui retourne le point suivant de la descente du gradient à pas fixe.

Astuce 1 : La classe `gradient_descent_constant_step.py` dispose d'un attribut de type `abstract_function`. Cet attribut à deux méthodes, `function_definition()` et `gradient()`.

Astuce 2 : Votre méthode `descent()` doit retourner :
- La liste des points que vous avez visité
- Leur valeur par la fonction de étudiée
- La norme du gradient en ces points

Vous pouvez retourner toutes ses informations avec un dictionnaire python ou en créant un objet python avec ces 3 attributs.

Astuce 3 : Vous pouvez utiliser les arrays de numpy pour vous aider. Vous trouverez sur Moodle une cheat sheet numpy.

Question 4 Dans le fichier *main.py* utilisez votre implémentation de la descente du gradient pour trouver le minimum de la fonction de Rosenbrock. Testez le comportement de votre algorithme avec plusieurs paramètres.

Astuce 1 : Pensez à instancier un objet pour la descente de gradient et pour la fonction et passer les attributs nécessaires pour réaliser la descente de gradient.

Astuce 2 : La descente du gradient est une méthode de la classe, alors vous allez devoir appeler la méthode `descent()` sur l'objet instancié précédemment

2.3 Descente de gradient à pas variable

Vous allez proposer une nouvelle implémentation de votre descente du gradient, en remplaçant le pas fixe par un pas variable de la forme :

$$t_k = \frac{a}{1 + b * k}$$

Question 1 : Cette modification change-t-elle fortement notre algorithme ?

Question 2 : Créez un fichier `gradient_descent_variable_step.py` contenant une classe `GradienDescentVariableStep` qui hérite de la classe `AbstractGradientDescent`. Modifiez le constructeur de votre classe pour prendre en compte ces deux nouveaux paramètres. Modifiez la méthode `get_next()` pour prendre en compte le nouveau pas.

Question 3 : Testez votre code dans le fichier `main.py` avec différents paramètres. Que constatez-vous ?

2.4 Descente de gradient à pas normalisé

Vous allez proposer une nouvelle implémentation de votre descente du gradient, en remplaçant le pas fixe par un pas normalisé :

$$t(x) = \frac{t}{\|\nabla f(x)\|}$$

avec un t une constante fixe.

Question 1 : Cette modification change-t-elle fortement notre algorithme ?

Question 2 : Créez un fichier `gradient_descent_normalized.py` contenant une classe `GradienDescentNormalized` qui hérite de la classe `AbstractGradientDescent`. Modifiez la méthode `get_next()` pour prendre en compte le nouveau pas.

Question 3 : Testez votre code dans le fichier `main.py` avec différents paramètres. Que constatez-vous ?

3 Descente de gradient appliqué à la régression linéaire (bonus)

3.1 Problème initial : la régression des moindres carrés

On a observé m individus et une valeur d'intérêt qui leur est associée (l'appétence pour un type musical donné, le besoin d'une formation complémentaire sur un thème donné, la probabilité de rembourser un crédit, ...). L'objectif est de comprendre le lien entre un individu et la valeur d'intérêt afin par exemple de prédire cette valeur d'intérêt pour un nouvel individu.

En pratique, chaque individu $i \in [\![1, m]\!]$ est associé à un vecteur $\mathbf{x}^{(i)} \in \mathbb{R}^n$ et à sa valeur d'intérêt $y^{(i)}$. Le vecteur $\mathbf{x}^{(i)}$ et la valeur $y^{(i)}$ sont connus pour tous les individus observés jusqu'à présent. Pour un nouvel individu, on connaîtra, $\mathbf{x}$, mais pas y. Par la suite, on notera $\mathbf{X}$ la matrice de dimension $m \times n$ dont chaque ligne i correspond au vecteur $\mathbf{x}^{(i)}$, et on notera $\mathbf{y}$ le vecteur de taille m dont la i-ème composante vaut $y^{(i)}$:

$$\mathbf{X} = \begin{bmatrix} — \, {\mathbf{x}^{(1)}}^T \, — \\ — \, {\mathbf{x}^{(2)}}^T \, — \\ \vdots \\ — \, {\mathbf{x}^{(m)}}^T \, — \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}.$$

On utilise le modèle le plus simple pour relier $\mathbf{x}$ et y : on suppose qu'il existe $\beta \in \mathbb{R}^n$ tel que y soit égal au produit scalaire entre $\mathbf{x}$ et β : $y = \langle \mathbf{x}, \beta \rangle = \sum_{j=1}^{n} \beta_j x_j$. Il reste alors à choisir un estimateur $\hat{\beta}$ de β calculé à partir des valeurs $(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)})$.

Nous choisissons l'estimateur des moindres carrés :

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} \left(\langle \mathbf{x}^{(i)}, \beta \rangle - y^{(i)} \right)^2, \tag{1}$$

qui se réécrit sous les formes

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \frac{1}{m} \left\| \mathbf{X}.\beta - \mathbf{y} \right\|_2^2 \quad \text{et} \quad \hat{\beta} = \underset{\beta}{\operatorname{argmin}} \frac{1}{m} \left(\mathbf{X}.\beta - \mathbf{y} \right)^T . \left(\mathbf{X}.\beta - \mathbf{y} \right). \tag{2}$$

Question 1 : Faire un schéma en 2 dimension du problème posé et visualiser ce que représente $\hat{\beta}$

Question 2 : Quel est le gradient par rapport à β de la fonction minimisée ?[1]

3.2 Implémentation

Question 1 : Créez une fichier `least_square_error.py`. Ce fichier contiendra la classe `LeastSquareError` qui héritera de `AbstractFunction` . Codez la méthode

1. Vous pouvez vous aider de The Matrix Cookbook, à condition de trouver la formule qui convient...

`function_definition()` et `gradient()`.

Les valeurs de $\mathbf{X}$ et $\mathbf{y}$ s'obtiennent en appelant la fonction `get_data()` du module `reg_data.py`. La fonction retourne un couple de valeur, la première représentant $\mathbf{X}$ et la seconde $\mathbf{y}$. Vous pouvez faire `X, Y = reg_data.py` pour obtenir les deux valeurs.

Question 2 : Utilisez le code fait dans la partie 1 pour réaliser une descente du gradient dans le cas de l'erreur des moindres carrées.

Question 3 : À quoi correspond le point trouvé ?