MATHELIER Rose

Projet personnel n°2
Sujet 1 : API de géoréférencement vecteur

Objectifs :
- Calculer une transformation géométrique à partir d'un ensemble de points d'appui (plusieurs transformations possibles).
- Transformer une couche de géométries (features).
- Exporter la couche ainsi géoréférencée.
- Produire un rapport sur le géoréférencement effectué : précision (RMS) calculée à partir d'un ensemble de points de contrôle, transformation choisie, vecteurs d'erreur, etc.

*

Ce document présente brièvement la conduite du projet, l'analyse (notamment le diagramme de classe UML), les difficultés rencontrées et les perspectives.
Le code source est hébergé sur GitLab.com/RMathelier.

Plan du rapport :

I) Étude du besoin et de l'existant.

II) Analyse et conception.

III) Résultats.

IV) Perspectives.

V) Bibliographie.

Merci à Serge Botton pour son aide au cours de ce projet, ses explications concernant les transformations de coordonnées et les moindres carrés, et les cours qu'il m'a fourni.

I) Étude du besoin et de l'existant.

On cherche à concevoir une API de géoréférencement vecteur, packagée sous la forme d'une librairie (.jar).

Les spécifications sont les suivantes :
- le programme doit être codé en Java.
- l'utilisation de Maven est attendue.
- une Javadoc doit être générée.
- le programme doit être testé (tests unitaires avec JUnit), et le travail doit être effectué en TDD (Test Driven Development).

La plupart des SIG disposent déjà d'extensions de géoréférencement, mais principalement dédiés aux fichiers rasters. Le fonctionnement sera néanmoins similaire :
- l'utilisateur fournit la couche en entrée.
- il fournit également les points d'appuis (Ground Control Points, ou GCP) et les points de contrôle (Check Points, ou CP).
- le programme calcule les paramètres de transformation (différents selon le type de transformation) à partir des points d'appuis, en utilisant la méthode des moindres carrés. Cette méthode permet également de calculer des résidus pour chaque paramètre.
- enfin, le programme applique la transformation à l'ensemble des géométries de la couche.

Remarque sur les points d'appuis et de contrôle :
Le programme, ainsi que les commentaires et la Javadoc étant en anglais, il est important de préciser la terminologie. En effet, « point d'appui » se dit en anglais « ground control point », souvent raccourci en « control point », à ne pas confondre avec les « points de contrôle » en français, qui correspondent en réalité aux « check points ».

L'extension de géoréférencement raster de QGIS (Georeferencer) implémente les types de transformation suivants : transformation linéaire, transformation de Helmert, transformation polynomiale (ordre 1 ou affine, ordre 2, ordre 3), transformation projective, transformation TPS (thin plate spline).

Une première étape du projet fut donc de déterminer les formules correspondant à ces transformations. Une rapide recherche a permis de trouver différents cours, disponibles dans la bibliographie, fournissant les formules suivantes :

Transformation linéaire :

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \end{pmatrix} = \begin{pmatrix} T_X \\ T_Y \\ T_Z \end{pmatrix} + \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix}$$

Transformation de Helmert, ou similitude à 7 paramètres :

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \end{pmatrix} = \begin{pmatrix} T_X \\ T_Y \\ T_Z \end{pmatrix} + \begin{pmatrix} 1+D & \varepsilon_Z & -\varepsilon_Y \\ -\varepsilon_Z & 1+D & \varepsilon_X \\ \varepsilon_Y & -\varepsilon_X & 1+D \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix}$$

Transformation polynomiale :

$$\begin{cases} E_2 = \sum_{i=1}^{n} A_i \times E_1^{i-1} \\ N_2 = \sum_{i=1}^{n} B_i \times N_1^{i-1} \end{cases}$$

Transformation projective (matrice à appliquer aux coordonnées projectives):

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \end{pmatrix} = \begin{bmatrix} m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 \\ m_7 & m_8 & m_9 \end{bmatrix} \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix}$$

II) Analyse et conception

L'étude de l'existant nous a montré que le processus de géoréférencement s'effectue en trois étapes :
1) Constitution du contexte de la transformation (couche en entrée, points d'appuis, points de contrôles).
2) Calcul des paramètres de transformation et des résidus.
3) Transformation concrète à partir des paramètres calculés.
Ces fonctionnalités doivent donc être correctement isolées, selon le principe de responsabilité unique. On leur attribue donc chacune une classe : Context, Transformation pour le calcul des paramètres, et Georeferencer pour la transformation concrète.

Notre application doit être conçue pour être fonctionnelle, mais aussi maintenable et évolutive. Aussi, le code doit être propre, et factorisé au maximum afin de réduire le nombre de modifications à effectuer dans le code au cours de la vie de l'application. En effet, la première version ne comportera qu'un nombre réduit de transformations implémentées, car la priorité en Agile est d'avoir un outil fonctionnel au plus vite et d'ajouter des fonctionnalités ensuite.
On mettra donc en place un patron de conception fabrique (ou factory) autour de la classe Transformation : c'est la classe TransformationFactory, et uniquement celle-ci, qui construira les classes concrètes de transformation, en fonction du type choisi par l'utilisateur.

Pour plus de lisibilité, on isole les paramètres de chaque transformation dans des classes dédiées implémentant une interface Parameters. C'est une méthode de cette interface, appelée par le Georeferencer, qui réalise le cœur du travail en appliquant les paramètres de transformation (en attribut) aux coordonnées passées en paramètres de la méthode.

Le Georeferencer effectue un travail différent selon la nature des géométries de la couche fournie en entrée : c'est donc un Georeferencer abstrait, qui sera réalisé par trois Georeferencer concret, pour les trois types de géométrie basiques qu'on trouve dans les SIG : point, ligne et polygone. Toutefois, si l'on souhaite spécialiser les géométries possibles (par exemple traiter le cas des polygones avec trous, non traités dans la version actuelle de l'application), le code est écrit de manière à ce que l'ajout soit possible sans modifier le code en profondeur, grâce au polymorphisme.

Le Context est constitué des points et des couches en entrées, il a donc deux liens d'agrégation avec les classes ControlPoint et CheckPoint. Ces classes héritent toutes deux de la classe abstraite PointConnu, elle-même classe fille de la classe Point de Java Topology Suite (JTS). Elles sont très similaires, leur seule différence résidant pour l'instant dans la cardinalité de l'agrégation (il faut au minimum un point d'appui, ce n'est pas le cas pour les points de contrôle). On peut cependant imaginer implémenter des différences entre ces deux classes, par exemple pour l'affichage. Cette possibilité justifie la création de deux classes, même si elles sont en l'état actuel interchangeables avec leur classe mère.

Enfin, on crée également des classes utilitaires dédiées à la lecture et à l'écriture de shapefiles avec des types de géométries différents.

Le diagramme de classe définitif de la première version de l'application est disponible sur dépôt GitLab.

L'application fait appel à un certain nombre de dépendances, gérées grâce au gestionnaire de dépendances Maven.
- Pour la gestion des géométries et des shapefiles, on utilise GeoTools, qui elle-même fait appel à JTS.
- Pour le calcul matriciel, on utilise Efficient Java Matrix Library (EJML).
- Pour les tests unitaires, on utilise JUnit (fourni avec Java) ainsi que PowerMock, qui étend Mockito en ajoutant des fonctionnalités de mock pour les méthodes statiques.

Nous avons fait le choix de ne pas utiliser de librairie pour les moindres carrés, en raison du temps de prise en main que cela représentait compte tenu des compétences limitées de l'équipe de développement dans ce domaine. Nous avons donc implémenté en premier lieu les transformations les plus courantes, modélisables par un système linéaire ne nécessitant pas d'itération des moindres carrés : ce sont la transformation linéaire et la transformation de Helmert. Ces transformations peuvent être résolues par un algorithme très basique, calculant simplement la solution des moindres carrés à partir du vecteur d'observation et de la matrice modèle. Une librairie telle que Least Squares In Java pourra cependant être installée lors des versions suivantes de l'application, si des transformations non linéaires telles que les transformations polynomiales sont implémentées.

III) Résultats

Nous avons implémenté et testé l'ensemble des fonctionnalités demandées, avec deux transformations possibles : la transformation linéaire et la transformation de Helmert. Les résultats obtenus sont les suivants :

Transformation linéaire d'une couche de points :

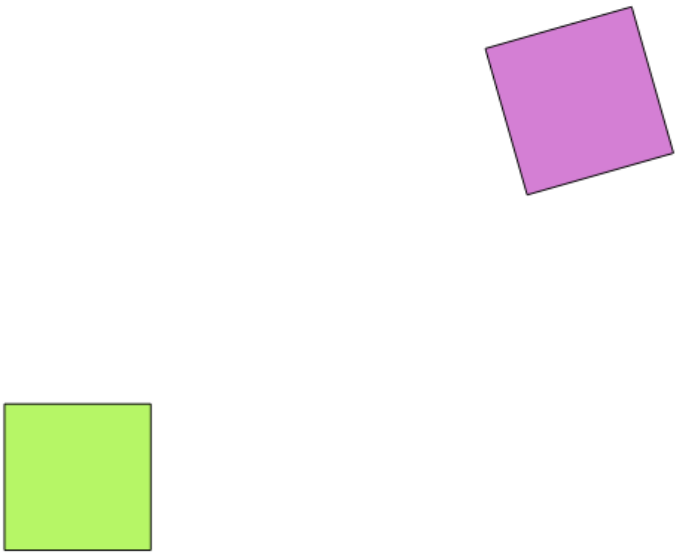
```
--- GROUND CONTROL TO MAJOR TOM... ---

*****

Transformation : class ensg.tsi.majortom.LinearTransfo.

Parameters :
- dx = 0.3333333333333333.
- dy = 3.3333333333333335.
- dz = 0.0.

Residuals :
GCP n° 1 :
- X = 0.0 => Rx = 1.6666666666666667
- Y = 0.0 => Ry = -2.3333333333333335
- Z = 0.0 => Rz = 0.0
GCP n° 2 :
- X = 1.0 => Rx = 2.666666666666667
- Y = 1.0 => Ry = 3.666666666666666
- Z = 0.0 => Rz = 0.0
GCP n° 3 :
- X = 3.0 => Rx = -4.333333333333334
- Y = 5.0 => Ry = -1.333333333333334
- Z = 0.0 => Rz = 0.0

Accuracy :
- RMS = 2.608745973749755
- RMSx = 1.9329022507905338
- RMSy = 1.7519830034690536
- RMSz = 0.0
```

Gauche : en violet, les points à leur position initiale ; en jaune, les points géoréférencés.
Droite : le rapport généré après cette transformation.

En observant ces résultats, on constate que la précision est mauvaise et les résidus importants car on a rentré au hasard les coordonnées des points de contrôle pour le test.

On peut également travailler avec des couches de lignes et de polygones, en prenant garde de garder de travailler avec des groupes de coordonnées correspondant aux entités, en conservant leur ordre et, dans le cas des polygones, en gardant des coordonnées identiques au début ou à la fin pour refermer le polygone.

Le résultat d'une transformation de Helmert sur une couche de polygone est le suivant :



En vert, le polygone à sa position initiale, en violet, le polygone géoréférencé.

```
--- GROUND CONTROL TO MAJOR TOM... ---

*****

Transformation : class ensg.tsi.majortom.HelmertTransfo.

Parameters :
- T1 = 3.2857142857142865.
- T2 = 3.4285714285714275.
- T3 = 0.0.
- R1 = 0.0.
- R2 = 0.0.
- R3 = -1.0.
- S = -0.7142857142857139.

Residuals :
GCP n° 1 :
- X = 0.0 => Rx = -1.2857142857142865
- Y = 0.0 => Ry = -2.4285714285714275
- Z = 0.0 => Rz = 0.0
GCP n° 2 :
- X = 1.0 => Rx = 1.4285714285714275
- Y = 1.0 => Ry = 3.2857142857142865
- Z = 0.0 => Rz = 0.0
GCP n° 3 :
- X = 3.0 => Rx = -0.1428571428571449
- Y = 5.0 => Ry = -0.8571428571428577
- Z = 0.0 => Rz = 0.0
Accuracy :
- RMS = 1.4374722712498653
- RMSx = 0.8586296628693805
- RMSy = 1.152858027969212
- RMSz = 0.0
```

IV) Perspectives

Ce projet constitue une première version fonctionnelle de notre API de géoréférencement vecteur. Il a été réalisé en tentant de respecter au maximum les bons principes de conception, de manière à pouvoir étendre facilement ses fonctionnalités, pour l'instant limitées à deux types de transformations pour les trois types basiques de géométries.

Les versions suivantes de l'application devraient élargir à la fois les géométries traitées, notamment le cas relativement fréquent des polygones avec trous, qui n'est pas pris en compte ici. On pourra ensuite optimiser le programme en se calquant sur l'arbre des géométries de GeoTools pour appliquer à chacune un traitement adapté.

Une seconde piste importante d'amélioration est de diversifier les types de transformation possibles. Il suffira pour cela de créer une classe supplémentaire héritant de Transformation ainsi que la classe de paramètres associée et d'implémenter le nouvel algorithme de calcul des paramètres, celui d'application des paramètres aux coordonnées et de définir le nombre minimal de points d'appuis pour la réaliser, le reste de l'application n'a pas à être touché.

Il serait également souhaitable de contrôler les entrées fournies par les utilisateurs afin de lever des exceptions lorsque c'est nécessaire, par exemple lorsqu'un chemin vers un fichier est invalide, ou quand le fichier n'est pas un shapefile.

Enfin, on peut réfléchir à d'autres fonctionnalités, par exemple générer un rapport plus propre, en PDF, avec une illustration des résidus, et un aspect global plus agréable à l'œil.

V) Bibliographie

Beilin, Jacques (2012), *Notions de moindres carrés*, École Nationale des Sciences Géographiques.

Botton, Serge (2017), *Systèmes de référence et transformations de coordonnées*, École Nationale des Sciences Géographiques.

Boyer, Edmond, *Les transformations géométriques du plan*, Institut d'Informatique et Mathématiques Appliqués de Grenoble.