Projet Génie Logiciel 2

Livrable 4

# Introduction

Lors du livrable 3 nous avions présenté la partie conception de notre projet, c'est-à-dire le diagramme de classe fait en UML, ainsi que la partie réalisation, la maquette de notre interface graphique.

Nous allons, dans ce dernier livrable, présenter la version finale de notre projet, en :

* Montrant les résultats que nous obtenons avec des exemples
* Expliquant comment nous obtenons ces résultats
* Faisant le point sur ce qui a été fait et ce qui n’a pas été fait

# Sommaire

I – Bases p.4

II – Espaces Vectoriels p.6

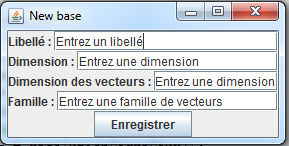
III – Polynômes p.8

IV – Applications Linéaires p.14

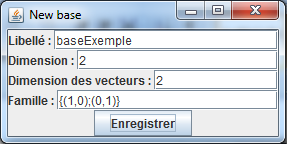
V – Conclusion p.

# I – Bases

Comme il était demandé dans le cahier des charges, notre application permet de créer des bases. Ces bases vont nous permettre de pouvoir construire des espaces vectoriels par la suite. Voici à quoi ressemble la fenêtre qui permet de créer une base :



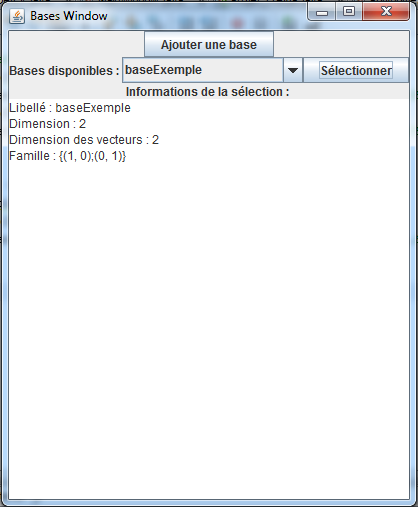
Et voici un exemple de création :



Une fois que l’utilisateur à rempli tous les champs, il peut cliquer sur « Enregistrer » ce qui lance la sérialisation de l’objet. Veuillez noter que la saisie de la famille de vecteurs qui constitue la base s’écrit sous cette forme : {(x1,y1,…) ;(x2,y2,…) ;… ;(xn,yn,…)}. Ceci est très important pour la sérialisation, car le programme reçoit une chaine de caractère, qu’il va devoir décortiquer afin de construire une matrice. C’est pour cette raison que nous avons établi une manière formelle de saisir les matrices.

Si la base à bien été crée, l’utilisateur en est averti via un message pop-up. Sinon le programme gèrera une exception, suivant le problème rencontré.

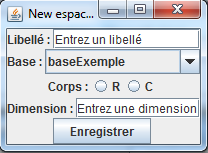
L’utilisateur peut ensuite visualiser les bases qu’il a saisies, en les sélectionnant à l’aide d’un menu déroulant contenant toutes les bases créées.



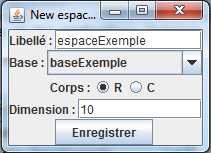
Donc comme on peut le voir, après sélection de la base que nous venons de créer, on retrouve bien les informations qui ont été saisies précédemment.

# II – Espaces vectoriels

Notre application permet aussi de construire des espaces vectoriels, qui serviront d’ensemble pour les polynômes et les applications linéaires. L’utilisateur peut saisir un nouvel espace vectoriel via cette fenêtre :

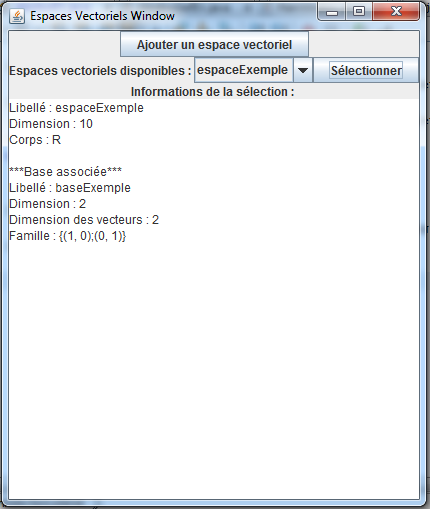


Voici un exemple de création :



Mathématiquement, un espace vectoriel ne peut être construit sans une base. Et comme on peut le voir sur l’image, on retrouve bien les bases que nous avons créées, dans le menu déroulant. Encore une fois l’utilisateur est prévenu via un message pop-up en cas de création de l’espace vectoriel, sinon une exception sera levée.

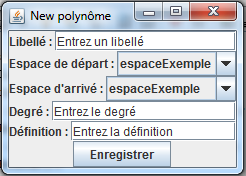
Une fois que l’utilisateur a saisi au moins un espace vectoriel, il peut le sélectionner via cette fenêtre et obtenir ses informations, comme ci-dessous :



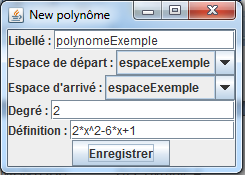
Comme vous pouvez le voir, la fenêtre indique bien les informations que nous avons saisies à la création, mais en plus il affiche les informations de la base sur laquelle il est construit.

# III – Polynômes

L’utilisateur peut créer des polynômes, pour lesquels il pourra calculer la primitive, la dérivé, l’image et les racines. Il peut saisir un polynôme via cette fenêtre :

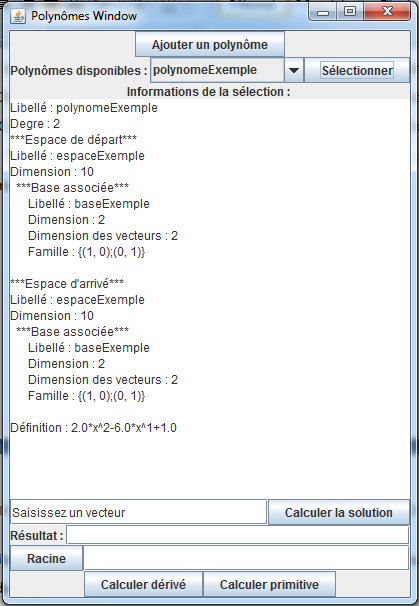


Et ci-dessous un exemple de création :

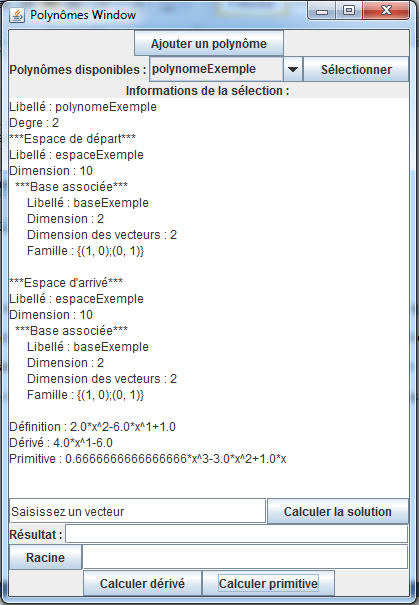


En ce qui concerne la saisie de la définition il est essentielle quelle soit saisie de cette manière : cn\*x^n+…+c0. Chaque coefficient doit être saisi, même si celui-ci est nul, sinon une exception sera levée. Comme pour les formulaires de création précédents, si la création s’est bien déroulée alors l’utilisateur en est averti par un message pop-up, dans le cas contraire une exception sera levée suivant l’erreur rencontrée.

Une fois que l’utilisateur possède un polynôme il peut, via la fenêtre ci-dessous, accéder aux informations du polynôme.



Comme on peut le voir sur l’image ci-dessus, les informations indiquent tout ce qui concerne le polynôme sélectionné ainsi que toutes les informations des espaces de départ et d’arrivé. Ensuite si l’utilisateur clique sur « Calculer dérivé » ou « Calculer primitive » le résultat du calcul demandé vient s’insérer dans la fenêtre d’information, comme ci-dessous :

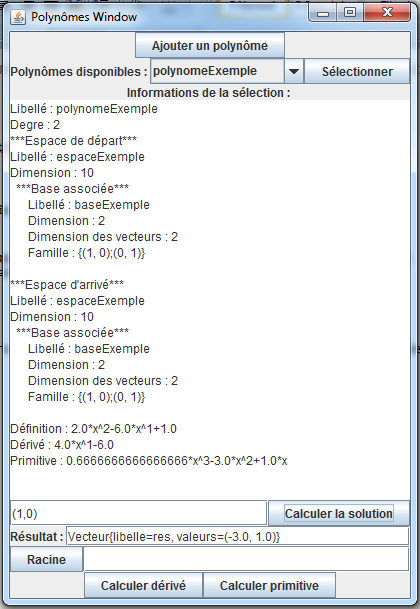


On voit bien sur l’image ci-dessus, les résultats de la dérivé et de la primitive. Voici comment ces calculs sont réalisés :

Procédure getDerive()  
Variables locales :   
 double[degre] coeffDerives (avec degre, le degré du polynôme)  
 Polynome derive  
Début  
 Pour i<-0 à i < degre  
 coeffDerives[i] = coefficients[i] \* (degre-i) //Avec coefficients le tableau de coefficients du polynôme  
 i<-i+1   
 Fin Pour  
 derive = Nouveau Polynome(Dérivé, coeffDerive, degre-1, espace de départ, espace d'arrivé)  
 p.derive = derive (p étant le polynôme sur lequel on appelle la procédure)  
Fin

Procédure getPrimitive()  
Variables locales :   
 double[degre+1] coeffPrimitive (avec degre, le degré du polynôme)  
 Polynome primitive  
Début  
 Pour i<-0 à i < degre+1  
 coeffPrimitive[i] = coefficients[i] /((degre-i)+1) //Avec coefficients le tableau de coefficients du polynôme  
 i<-i+1  
 Fin Pour  
 primitive = Nouveau Polynome(Primitive, coeffPrimitive, degre-1, espace de départ, espace d'arrivé)  
 p.primitive = primitive (p étant le polynôme sur lequel on appelle la procédure)  
Fin

L’utilisateur peut aussi demander à calculer l’image d’un polynôme pour un vecteur donnée, voir ci-dessous :

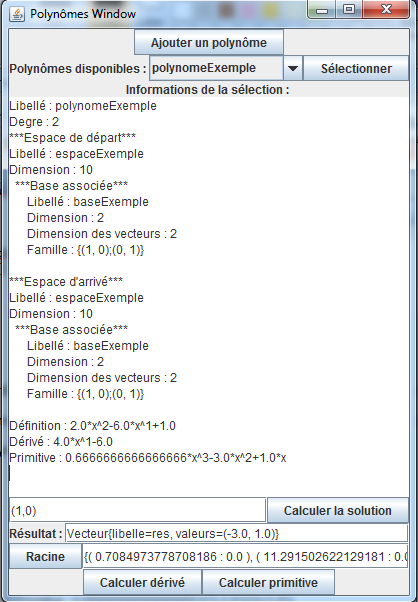


Dans l’exemple ci-dessus, nous avons demandé au programme de calculer la solution du polynôme, pour le vecteur et nous obtenons le vecteur résultat .

Pour calculer la solution, nous utilisons l’algorithme ci-dessous :

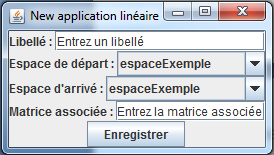
Fonction calculerImage(Vecteur v : entré) : Vecteur  
Variables locales :  
 Vecteur[degre+1] retenue  
 String[v.dimension] valRes  
 Vecteur res(res, valRes)  
 Entier deg = p.degre  
Début  
 Pour i<-0 à i < valRes.length  
 valRes[i] = "" //Initialisation du tableau  
 i<-i+1  
 Fin Pour  
 Pour i<-0 à i < degre+1  
 retenue[i] = Vecteur(v) //Constructeur par recopie  
 retenue[i] = retenue[i].puissance(deg) //Méthode de la classe vecteur qui calcul la puissance  
 retenue[i] = retenue[i].multiplierScalaire(coefficients[i])  
 deg<-deg-1  
 i<-i+1  
 Fin Pour  
 Pour i<-0 à i < retenue.length  
 res.additionner(retenue[i])  
 i<-i+1  
 Fin Pour  
 Retourner res  
Fin

Et finalement l’utilisateur peut demander à obtenir les racines du polynôme. Cependant, notre programme n’est capable de calculer les racines des polynômes de degré 4 maximum. Et pour les polynômes de degré 4, il ne peut calculer que les racines réelles.

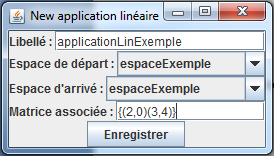


# IV – Applications Linéaires

Et dernièrement l’utilisateur peut créer des applications linéaires via cette fenêtre :

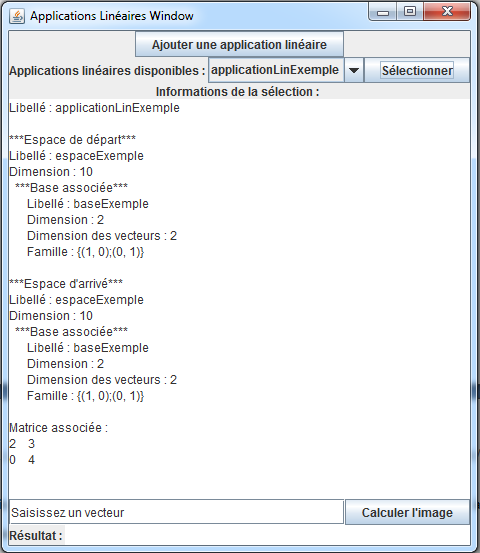


Ci-dessous un exemple de création :

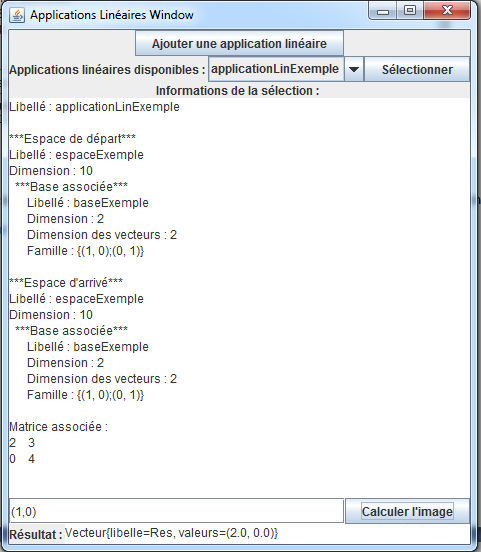


Comme le montre l’impression d’écran ci-dessus, l’application linéaire est définie par sa matrice associée, qui se saisie comme toute saisie de matrice dans le logicielle.

Une fois que l’utilisateur à créer une application linéaire, il peut accéder à ces informations via la fenêtre suivante :



Si l’utilisateur le désire, il peut calculer l’image de l’application linéaire sélectionnée, pour un vecteur donné, exemple ci-dessous :



Nous utilisons l’algorithme ci-dessous, pour calculer l’image :

Fonction calculerImage(Vecteur v : entré) : Vecteur  
 Variables locales :  
 double retenue<-0  
 String[v.dimension] coeffRes  
 Vecteur res(res, coeffRes)  
Début  
 Pour i<-0 à i < matAssociee[0].getValeurs().length //Récupère la dimension des vecteurs  
 Pour j<-0 à j < matAssociee.length  
 retenue<-retenue+(matAssociee[j].getElement(i))\*(v.getElement(j))  
 res.setElement(i, retenue)  
 j<-j+1  
 Fin Pour  
 retenue<-0  
 Fin Pour  
 Retourner res  
Fin

# V – Conclusion

Pour conclure nous pouvons d’abord commencer par dire que nous n’avons pas rempli entièrement le cahier des charges, puisque nous ne pouvons calculer les racines d’un polynôme de degré 4 maximum et seulement si les racines sont réelles. Nous n’avons pas non plus réussi à mettre en place l’algorithme du pivot de Gauss qui aurait pu nous permettre de calculer le rang d’une application linéaire (ce qu’on ne fait pas). De plus mettre en place cet algorithme nous aurait permis de pouvoir déterminer si l’utilisateur entre une famille de vecteur libre, à la création d’une base.

Mais nous avons réussi à remplis tous les autres objectifs du cahier des charges qui demandaient de mettre en place un système de sérialisation. Lors de la création ou de la sélection d’élément, nous utilisons l’interface serializable, qui nous permet de faire des sauvegardes et des chargements. De plus nous avons réservé un dossier pour chaque type d’élément (espace vectoriel, base, …) pour tenir l’espace de travail assez propre. Nous avons aussi mis en place une interface graphique en suivant le pattern MVC comme cela était stipulé dans le cahier de charges.