4M016
Projet 2
Visualisation et optimisation de maillage

Alexandre MARIN

pour le 7 janvier 2019

Table des matières

1 Introduction

1.1 Présentation du projet

Il s'agit de visualiser avec Gnuplot des fonctions de $[-1,1]^2 \to \mathbb{R}$ (appartenant à $\mathcal{L}^2$) en les approximant par une fonction affine par morceaux, associée à un maillage de l'ensemble de définition composé de triangles. Pour cela on remplace la fonction sur chaque triangle K par sa projection orthogonale $\Pi_K(f)$ sur $P_1(K)$, ensemble des fonctions affines restreintes à K. Si l'erreur E_K

$$E_K = \int_K |f - \Pi_K(f)|^2 \,\mathrm{d}\lambda \tag{$*$}$$

est supérieure à ε on coupe le triangle en deux puis on recommence jusqu'à ce que la contrainte soit respectée sur chaque triangle du maillage obtenu.

Si f est une fonction à visualiser et $\Pi_K(f)(x,y) = ax+by+c$, alors il faut résoudre le système linéaire suivant pour déterminer (a,b,c),

$$\begin{pmatrix} \int_K x \,\mathrm{d}x\,\mathrm{d}y & \int_K y \,\mathrm{d}x\,\mathrm{d}y & |K| \\ \int_K x^2 \,\mathrm{d}x\,\mathrm{d}y & \int_K xy \,\mathrm{d}x\,\mathrm{d}y & \int_K x \,\mathrm{d}x\,\mathrm{d}y \\ \int_K xy \,\mathrm{d}x\,\mathrm{d}y & \int_K y^2 \,\mathrm{d}x\,\mathrm{d}y & \int_K y \,\mathrm{d}x\,\mathrm{d}y \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} \int_K f(x,y) \,\mathrm{d}x\,\mathrm{d}y \\ \int_K f(x,y)x \,\mathrm{d}x\,\mathrm{d}y \\ \int_K f(x,y)y \,\mathrm{d}x\,\mathrm{d}y \end{pmatrix}. \tag{$\dagger$}$$

Pour la quadrature des intégrales, il était demandé d'utiliser la formule

$$\int_K g \,\mathrm{d}\lambda = \frac{|K|}{2} \sum_{i=1}^{N} w_i \; g(F_K(\hat{z}_i)) \tag{$\ddagger$}$$

où les réels w_i et les points de $\hat{z}_i \in \mathbb{R}^2$ sont donnés dans un fichier de nom `coords.txt`.

1.2 Organisation

Ci-dessous se trouve un schéma récapitulant l'organisation du projet.

Les liens représentent les dépendances entre les modules : l'origine d'une flèche est le module dépendant. Ces modules sont décrits dans la suite du rapport.

Outre les fichiers en-tête et sources correspondant aux modules présentés, le répertoire du projet contient ces fichiers :

un `Makefile` : pour compiler automatiquement le projet ;

des scripts Gnuplot : pour afficher les résultats demandés ;

`README` : résume le projet et décrit la manière de lancer l'exécutable ;

`coords.txt` : contient les données pour la quadrature des intégrales ;

`project.cpp` : contient la fonction `main` et génère les données pour visualiser f_1 et f_2, en créant des fichiers `f_1.data` et `f_2.data`.

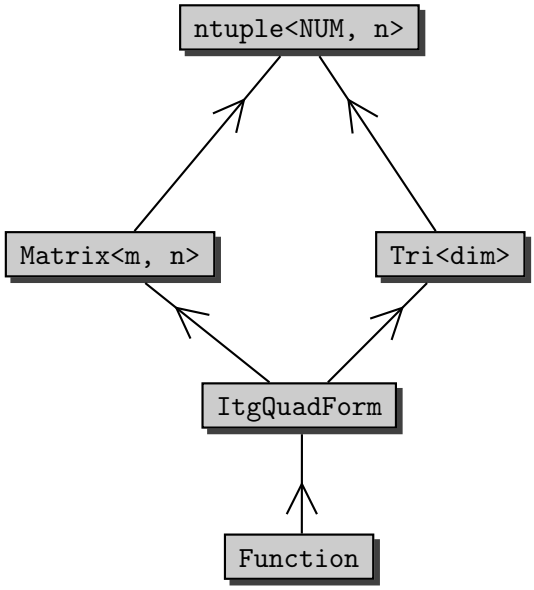

FIGURE 1 – Classes et modules programmés

2 Description des classes et modules programmés

2.1 Classe `ntuple`

Ce patron de classe `ntuple<n, NUM>` permet de représenter des `n`-uplets dont les coefficients sont codés grâce au type `NUM`. Il est essentiellement spécialisé pour utiliser des n-uplets de réels. Les opérations fournies correspondent aux opérations courantes sur les vecteurs de $\mathbb{R}^n$. Les opérateurs `[]` et `<<` sont définis respectivement pour l'indexation et l'écriture formatée.

2.2 Classe `Tri`

Ce patron de classe `Tri<dim>` désigne des triangles dans $\mathbb{R}^{\texttt{dim}}$. Voici les principales fonctionnalités :

`Tri<d>::Tri(const ntuple<d, double>&, const ntuple<d, double>&, const tuple<d, double>&)` :
construit un triangle à partir de trois points de $\mathbb{R}^{\texttt{d}}$.

`Tri<3>::Tri(const Tri<2>&, const std::vector<double>&)` :
construit un triangle de $\mathbb{R}^3$ à partir d'un triangle de $\mathbb{R}^2$ en ajoutant à ce dernier les trois éléments du `vector` comme coordonnées en z.

opérateur `<<` :
écrit au format Gnuplot le triangle en argument.

opérateur `[]` :
donne accès à l'un des sommets du triangle (l'indice doit être compris entre 0 et 2).

`Tri<2>::split(Tri[6])` :
cette méthode met dans le tableau passé en argument les six triangles que l'on peut obtenir en coupant un triangle de $\mathbb{R}^2$ selon une médiane, pour chaque médiane.

`static std::vector<Tri<2> > Tri<2>::generateMesh(double a,double b,double c,double d,int nb)` :
renvoie un vecteur représentant un maillage de $[\texttt{a}, \texttt{b}] \times [\texttt{c}, \texttt{d}]$ constitué d'au moins `nb` triangles.

2.3 Classe `Matrix`

Ce patron de classe `Matrix<m, n>` évoque l'ensemble $\mathcal{M}_{\texttt{m},\,\texttt{n}}(\mathbb{R})$, avec `m` et `n` « petits ». Il est instancié pour les valeurs $\texttt{m} = 3$ et $\texttt{n} = 4$ afin de matérialiser le système †.
Les principales fonctionnalités sont :

`double Matrix<m, n>::operator()(int i, int j)` :
donne accès au coefficient de la matrice d'indices $(i + 1, j + 1)$.

`Matrix<m, n>& Matrix<m, n>::gauss(std::vector<ntuple<2, int> >& ldg)` :

applique l'algorithme de Gauß-Jordan à l'argument implicite, renvoie une référence vers l'argument implicite et enregistre les positions de pivots dans `ldg`.

2.4 Classe `ItgQuadForm`

Cette classe a pour rôle de mettre en œuvre les formules de quadrature pour les intégrales ‡. Au début de l'exécution du programme, la classe crée une unique instance qui se chargera de lire le fichier `coords.txt` afin de récupérer les poids et points de quadrature. Les deux méthodes de classe sont :

`static const std::vector<ntuple<3, double> >& ItgQuadForm::getpw(int d)` :
donne accès aux poids et points de quadrature permettant une intégration exacte des fonctions polynômiales de degré au plus `d`. Chaque élément du vecteur en sortie est sous la forme

$$[\hat{z}_{i_1}, \hat{z}_{i_2}, w_i].$$

La taille du vecteur en sortie est le nombre de points de quadrature N dans ‡.

`static Matrix<3, 4>& ItgQuadForm::get_lin_sys(const Tri<2>& K)` :
renvoie une référence vers une matrice correspondant au système † pour $K =$ `K`. La dernière colonne n'est cependant pas initialisée. La matrice en question est une variable de la classe `ItgQuadForm`.

2.5 Classe `Function`

Classe abstraite

Cette classe abstraite permet de manipuler des fonctions de la forme $\mathbb{R}^{\mathtt{p}} \to \mathbb{R}^{\mathtt{q}}$. Les dimensions des espaces de départ et d'arrivée sont définies avec le constructeur et accessibles en lecture seule via `getp()` et `getq()`.

Le membre `deg`, réglable via `setdeg(int)`, est l'ordre qui sera utilisé lors de l'intégration numérique.

L'opérateur `()` permet d'évaluer la fonction qui appelle la méthode en plusieurs points de son domaine de définition : il faut cependant que les points soient concaténés dans un vecteur.

La méthode de clonage rend possible l'utilisation du polymorphisme : on peut alors par exemple traiter des listes hétérogènes de fonctions.

Les quatre dernières méthodes listées ci-dessous réalisent respectivement les actions suivantes (la fonction manipulée étant l'argument implicite) :

— l'intégration sur un triangle ;
— le calcul du triplet (a, b, c) du système $\dagger$;
— le calcul de E_K donnée par $*$;
— le raffinement du maillage du domaine de définition et la sauvegarde des données correspondantes au format Gnuplot.

Function
`const int p, q` `int deg = 2`
`Function(int, int)`

`virtual std::vector<R> operator()`
`  (const std::vector<R>&) const = 0`
`virtual Function* clone() const = 0`
`const int getp() const`
`const int getq() const`
`void setdeg(int)`

`R integrate(const Tri<2>&) const`
`ntuple<3, R> projection_on_P1(const Tri<2>&) const`
`R get_E_K(const Tri<2>&, ntuple<3, R>&) const`
`std::vector<Tri<2> > exportGnuplot(`
`  const std::vector<Tri<2> >&, const std::string&) const`

FIGURE 2 – Classe `Function`

Classes concrètes

Voici les classes dérivant de `Function` :

Nom de classe	Sens
`LinCbnFct`	combinaison linéaire de fonctions
`CompFct`	composition de fonctions
`RtoRCppFct`	fonction $\mathbb{R} \to \mathbb{R}$
`R2toRCppFct`	fonction $\mathbb{R}^2 \to \mathbb{R}$
`ProdFct`	produit de fonctions $\mathbb{R}^p \to \mathbb{R}$
`F_K`	fonction F_K définie par le projet, fonction affine envoyant le triangle unité $\hat{K}$ sur K

FIGURE 3 – Classes dérivant de `Function`

Les objets caractérisant ces fonctions sont définis lors de l'appel au constructeur. Souvent, il faut passer comme argument une liste d'adresses d'instances de `Function` ou un pointeur vers une fonction au sens du C++.

Les constructeurs vérifient que les créations de fonctions mathématiques ne sont pas absurdes : par exemple pour une composition de fonctions, les domaines et codomaines doivent correspondre.

Variables globales

Les variables globales suivantes sont définies :

Nom	Sens
`Square`	$\mathbb{R} \to \mathbb{R} : x \mapsto x^2$
`Cube`	$\mathbb{R} \to \mathbb{R} : x \mapsto x^3$
`IndR2`	$\mathbb{R}^2 \to \mathbb{R} : (x, y) \mapsto 1$
`pr1`	$\mathbb{R}^2 \to \mathbb{R} : (x, y) \mapsto x$
`pr2`	$\mathbb{R}^2 \to \mathbb{R} : (x, y) \mapsto y$

FIGURE 4 – Variables globales

3 Conclusion

3.1 Résultats

On obtient les graphiques ci-dessous en exécutant le programme en ligne de commande de cette façon : `./project 1e-6 32` ,

puis en exécutant les scripts avec Gnuplot.

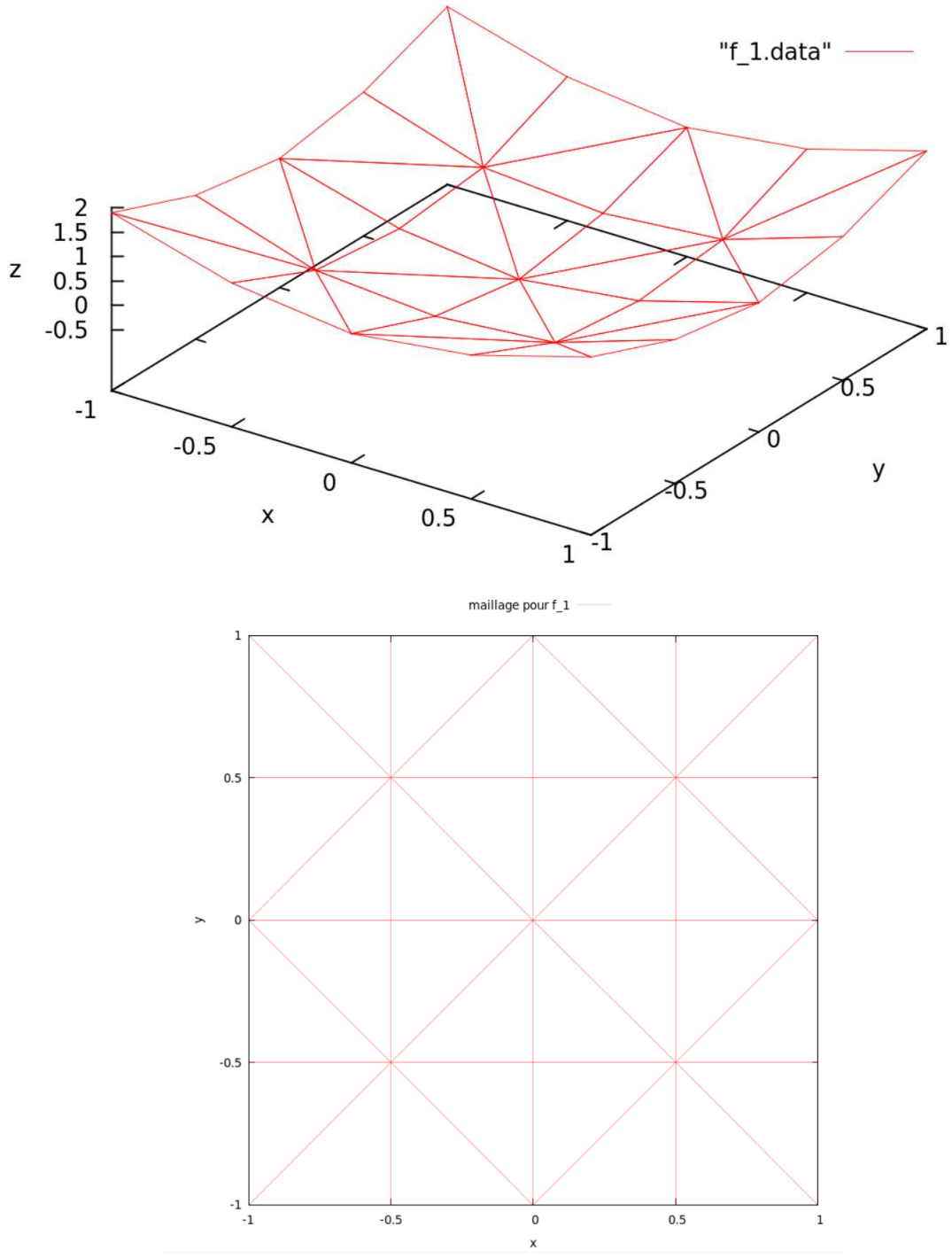

FIGURE 5 – représentation par Gnuplot de $f_1(x, y) = x^2 + y^2$ et du maillage raffiné

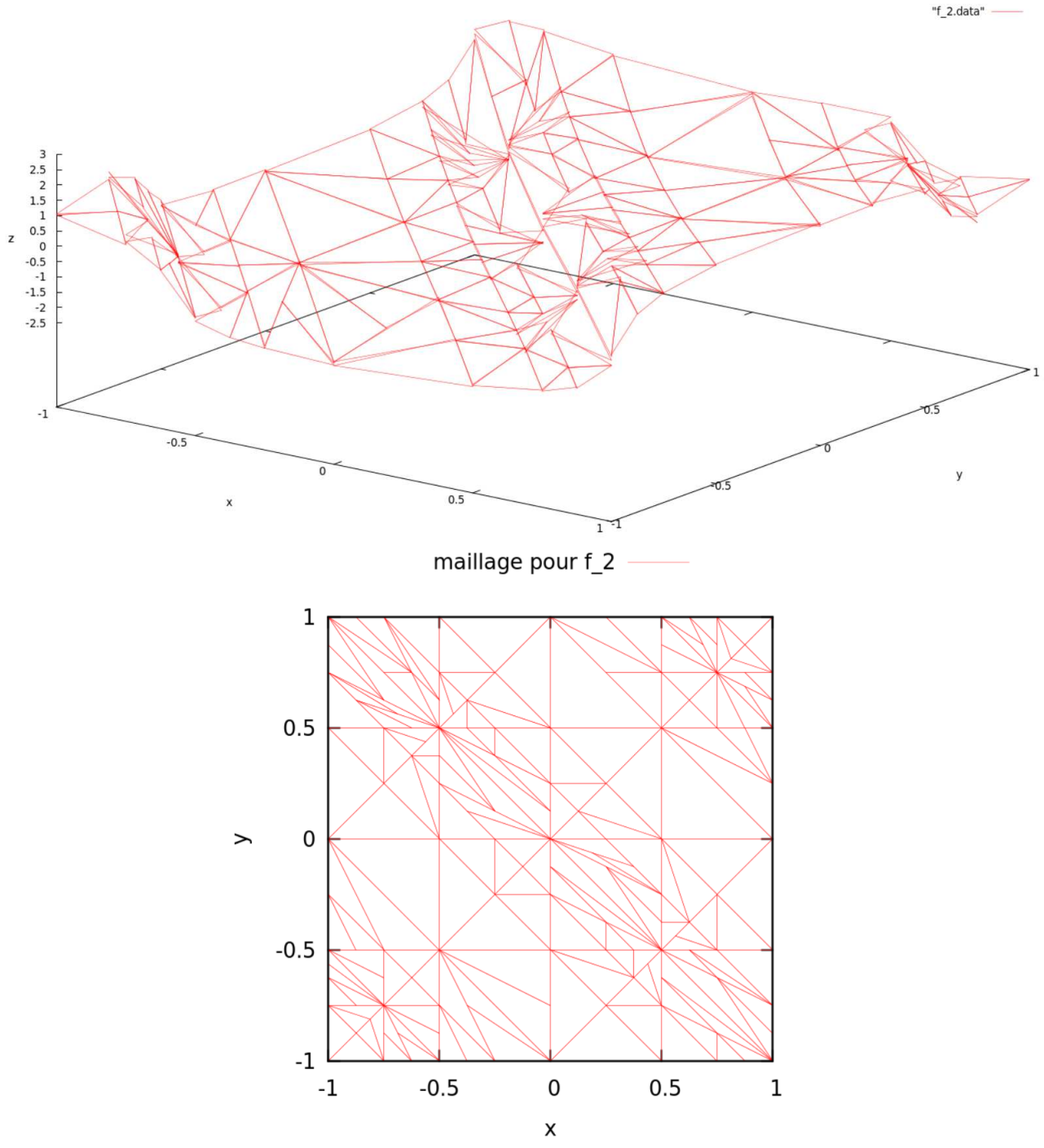

FIGURE 6 – représentation par Gnuplot de $f_2(x,y) = x^2 + y^3 + \tanh(5\sin(2(y+x)))$ et du maillage raffiné

3.2 Explications supplémentaires

Afin de rendre l'interprétation du code plus simple, un type `R` synonyme de `double` a été défini.

Les modules `ItgQuadForm` et `Function` définissent des classes, fonctions et variables uniquement dans l'espace de noms `fct`.

La quantité ε est représentée par la variable locale au module `Function`, de nom `epsilon`. Sa valeur peut être modifiée par `void fct::setepsilon(R)` et peut être lue via `R fct::getepsilon()`.

Les constructeurs (par défaut, par copie et par déplacement) et opérateurs d'affectation (par copie et par déplacement) ont été :

— soit redéfinis ;
— soit supprimés ;
— soit générés explicitement dans leur version proposée par défaut.

Cela permet d'éviter des erreurs liées à la création d'objets.

3.3 Améliorations

Afin de rendre l'exécutable plus rapide, on pourrait définir des constructeurs et opérateurs d'affectation par déplacement en manipulant davantage de pointeurs. Néanmoins pour cela il faudrait moins de champs déclarés `const` dans le cas des classes héritant de `Function`.

Pour améliorer la précision des calculs lors de la résolution du système †, on pourrait faire en sorte d'utiliser la formule d'intégration ‡ sans estimer l'aire du triangle : en effet la quantité $|K|$ apparaît dans chaque coefficient de la matrice augmentée du système linéaire.