Formalisation de l'algèbre de Grassmann en Coq

Laurent Théry, Laurent Fuchs et Sylvain Charneau

23 juin 2008

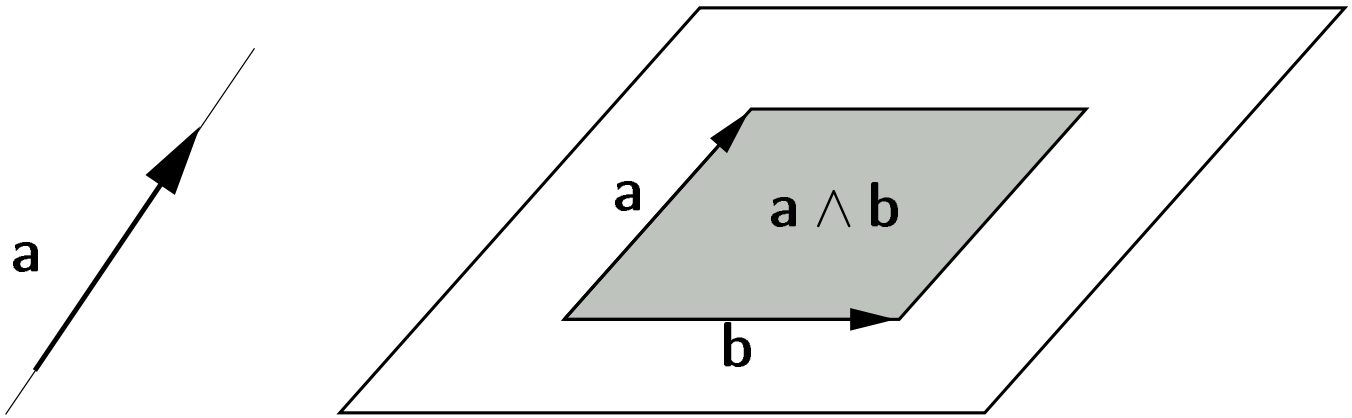
Visioconférence Galapagos du 23 juin 2008

En guise d'introduction

Le formalisme de l'algèbre de Grassmann

- Langage pour représenter et manipuler des sous-espaces (vectoriels)
- Sans faire référence à un système de coordonnées
- Généralisation des vecteurs ; notion de multi-vecteurs
- Exprimer la dépendance linéaire des vecteurs ($a \wedge b = 0$)

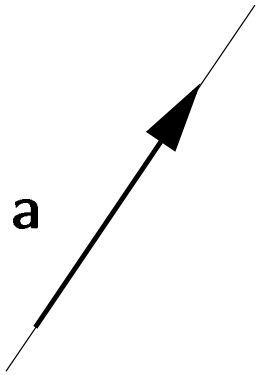
Algèbre géométrique (Grassmann)



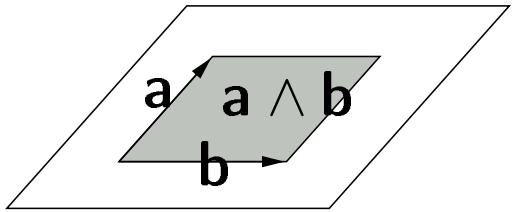
(a) Vecteur (b) Bi-vecteur

Base de $\mathcal{G}_2$: $(1, e_1, e_2, e_1 \wedge e_2)$

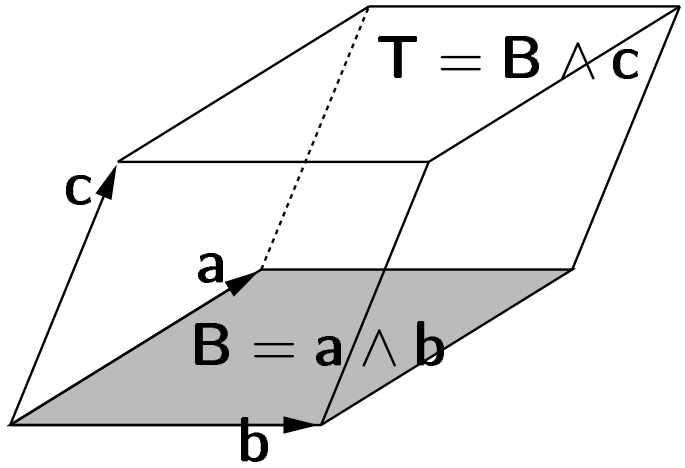
Algèbre géométrique (Grassmann)



(c) Vecteur



(d) Bi-vecteur



(e) Tri-vecteur

Base de $\mathcal{G}_3$: $(1, e_1, e_2, e_3, e_1 \wedge e_2, e_1 \wedge e_3, e_2 \wedge e_3, e_1 \wedge e_2 \wedge e_3)$

En Coq...

Comment formaliser ?

- Soit une représentation abstraite de l'algèbre
- Soit se définir un modèle (presque une structure de données)
  - A terme ne plus regarder dans la structure
  - Compromis entre un objet qui calcule et un objet sur lequel on fait des preuves

Représentation des éléments

- En dimension n, un élément ; arbre binaire complet de hauteur n

```
Function vect (n : nat) : Set := match n with 0 => V |
S n1 => (vect n1 * vect n1)%type end.
```

Définition des opérations

- Somme de 2 éléments

```
Function vplus (n : nat) : vect n -> vect n ->
vect n :=
match n return (vect n -> vect n -> vect n) with
| 0%nat => fun a b => a + b
| S n1 =>
fun l1 l2 =>
let (l3, l5) := l1 in
let (l4, l6) := l2 in (vplus n1 l3 l4, vplus n1 l5
l6)
end.
```

Définition des opérations

- Produit par un scalaire

```
Function vscal (n : nat) (k : V) struct n : vect n
-> vect n :=
match n return (vect n -> vect n) with
| O| S n1 =>
fun l1 =>
let (l2, l3) := l1 in (vscal n1 k l2, vscal n1 k
l3)
end.
```

Combinaison linéaire ; génération d'une base de l'algèbre.

En Coq...

Définition des opérations

- Produit extérieur

```
Function vprod (n : nat) : vect n -> vect n ->
vect n :=
match n return (vect n -> vect n -> vect n) with
| 0%nat => fun a b => a * b
| S n1 =>
fun l1 l2 =>
let (l3, l5) := l1 in
let (l4, l6) := l2 in (vplus n1
(vprod n1 l3 l6)
(vprod n1 (vopp n1 false l5) l4),
vprod n1 l5 l6)
end.
```

Preuve formelle ; produit extérieur anticommutatif pour les vecteurs.

À finir...

Théorème

(Décomposabilité des multi-vecteurs) *Soit $M = x_1 \wedge \cdots \wedge x_p$ un multi-vecteur décomposable de $\bigwedge(E)$. Si M est non nul, alors l'ensemble des vecteurs x de E vérifiant $x \wedge M = 0$ décrit un sous-espace vectoriel de E, dont la famille $(x_1, \ldots, x_p)$ est une base.*

Reste à montrer que la famille est libre.

À finir...

La contraction

- Utile pour calculer des intersections
- Définition :

$$\langle \phi, x_1 \wedge \cdots \wedge x_p \rangle = \frac{1}{(p-1)!} \sum_{\sigma} \epsilon_\sigma \phi(x_{\sigma(1)}) x_{\sigma(2)} \wedge \cdots \wedge x_{\sigma(p)}$$

- Difficulté ; représenter ϕ, actuellement image par ϕ des vecteurs de la base de E.

Une cela fait on peut envisager de parler des incidences.

À faire. . .

- Pour un multi-vecteur donné ;
  - Tester s'il est décomposable,
  - Trouver une décomposition.
  - On sait faire "sans Coq". Mais intéressant à formaliser car résultat non trivial et travail réutilisable.
- Ceci permettra simplifier le calcul d'intersections et d'union de sous-espaces.