MU5MEF39 : Optimisation en Aérodynamique

A. Belme

3 février 2022

Contenu 6ème séance :

- Algorithmes d'optimisation
- Algorithme SLSQP dans SU2 et premier exemple

2.6 Algorithmes d'optimalité

•On va présenter et analyser quelques **algorithmes** qui permetent de calculer numériquement **la solution** d'un problème d'optimisation.

• Il s'agit des **algorithmes itératifs** : à partir d'un u^0 on va construire une suite $(u^n)_{n\in\mathbb{N}}$ et nous discuterons : les critères (et hypothèses) de **convergence** et la **vitesse** de convergence

2.6.1 Algorithmes de type gradient : cas sans contraintes

On cherche a résoudre :

$$\inf_{v \in V} J(v) \tag{1}$$

avec J fonctionnelle (α-convexe différentiable) définie sur un espace de Hilbert V. D'après le chapitre 2,5 il existe une unique solution u, caractérisée par l'équation d'Euler :

$$J'(u) = 0$$

2.6.1 Algorithmes de type gradient (descente) : cas sans contraintes

L'algorithme du gradient consiste à se déplacer d'une itérée u^n en suivant une direction de descente : la ligne de plus grande pente associée à $J(v)$. On appel souvent ces méthodes **les méthodes de descente**.

De manière plus formelle, ces méthodes consistent à partir d'un u^0 , générer une suite u^n telle que : $J(u^{n+1}) < J(u^n)$, $\forall n \in \mathcal{N}$.

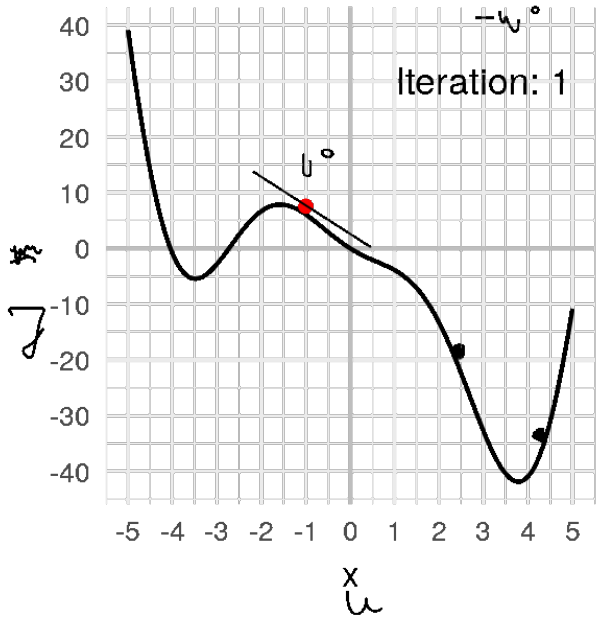
La direction de descente est donnée par le gradient $J'(u^n)$.

Si on cherche u^{n+1} sous la forme :

$$u^{n+1} = u^n - \mu^n w^n$$

avec $\mu^n > 0$ le pas de descente petit, alors la direction de descente w^n sera $w^n = J'(u^n)$.

2.6.1 Algorithmes de type gradient (descente) : cas sans contraintes



2.6.1 Algorithmes de type gradient (descente) : cas sans contraintes

La convergence de ces algorithmes dépendent de plusieurs paramètres (facteurs), nottament :

1. Direction de descente w^n
2. Pas de descente μ^n

2.6.1 Algorithmes de type gradient (descente) : cas sans contraintes

On appel **algorithme de descente à pas optimal** l'algorithme suivant :

A partir de u^0 quelconque, on construit la suite (u^n) définie par :

$$u^{n+1} = u^n - \mu^n J'(u^n)$$

où μ^n le pas de descente est choisi à chaque étape tel que :

$$J(u^{n+1}) = \inf_{\mu \in \mathbb{R}} J(u^n - \mu J'(u^n)).$$

2.6.1 Algorithmes de type gradient (descente) : cas sans contraintes

Un autre algorithme de gradient : **l'algorithme à pas fixe**, consiste en la construction de la suite

$$u^{n+1} = u^n - \mu J'(u^n)$$

avec μ **positif fixé**. Il s'agit d'un algorithme plus simple que le précédent et qui convergence sous certains hypothèses et contraintes sur μ.

Il existe des nombreux autres méthodes de type gradient (descente), par exemple le gradient conjuqué (direction de descente dépend également de la direction à l'itération précédente) .

Il existe des extensions de ces algorithmes pour le cas des problèmes d'optimisation avec contrainte, nottament la méthode à pas fixe (avec projection, algorithme d'Uzawa)

Question : comment choisir un algorithme en pratique ?

2.6.2 Algorithmes de type Newton

Idée de base : On se place en dimension finie $V = \mathbb{R}^N$. On cherche a résoudre $F(x) = 0$, avec F une fonction de classe C^2 sur $\mathbb{R}^N$. On suppose connu x_k, on définit x_{k+1} comme le point d'intersection de l'axe des abscisses avec la tangente à la courbe de F en x_k. En dimension $N \neq 1$ on a :

$$x^{(k+1)} = x^{(k)} - [F'(x^{(k)})]^{-1} F(x^{(k)})$$

où $F'(x^{(k)})$ est la matrice jacobienne inversible. A chaque itération, il faut donc calculer cette matrice et résoudre un système linéaire $[F'(x^{(k)})]\delta x^{(k)} = -F(x^{(k)})$.

Avantage : méthode de Newton converge bien plus vite que les précédentes

2.6.2 Algorithmes de type Newton

Application de la méthode de Newton aux problèmes d'optimisation :

• Si on considère $J = F'$ alors on résoud la condition nécessaire d'optimalité $J'(u) = 0$.

• Pour des problèmes de minimisation avec contrainte d'égalité. Soit J fonction de classe C^3 de $\mathbb{R}^N$ dans $\mathbb{R}$, $G = (G_1, ..., G_M)$ de classe C^3 de $\mathbb{R}^N$ dans $\mathbb{R}^M$ $(M \leq N)$ et soit u un minimum local de :

$$\min_{v \in \mathbb{R}^N,\ G(v)=0} J(v)$$

2.6.2 Algorithmes de type Newton

Si les vecteurs $(G_1'(u), \dots G_M'(u))$ sont linéairement indépendants, la condition nécessaire d'optimalité est

$$J'(u) + \sum_{i=1}^{M} \lambda_i G_i'(u) = 0, \quad G_i(u) = 0, \ \forall 1 \leq i \leq M$$

On peut résoudre ce système de (N+M) équations à (N+M) inconnus (u, λ) par une méthode de Newton.
On pose :

$$F(u, \lambda) = \begin{pmatrix} J'(u) + \lambda G'(u) \\ G(u) \end{pmatrix} \tag{2}$$

2.6.2 Algorithmes de type Newton

Alors la matrice jacobienne est :

$$F'(u,\lambda) = \begin{pmatrix} J''(u) + \lambda G''(u) & G'(u) \\ G'(u) & 0 \end{pmatrix} \tag{2}$$

On peut appliquer la méthode de Newton a cette fonction $F(u,\lambda)$ si la matrice F' est inversible.

Plus encore, il est intéressant d'intérpreter cet algorithme comme une méthode de minimisation. On introduit le lagrangien $\mathcal{L}(v,\mu) = J(v) + \mu G(v)$, ses dérivées par rapport à v, $\mathcal{L}'$ et $\mathcal{L}''$.

2.6.2 Algorithmes de type Newton

On vérifie que l'équation :

$$(u^{n+1}, \lambda^{n+1}) = (u^n, \lambda^n) - (F'(u^n, \lambda^n))^{-1} F(u^n, \lambda^n)$$

est la condition d'optimalité pour que u^{n+1} soit un point de minimisation du problème quadratique à contraintes affines :

$$\min_{w \in \mathbb{R}^N, G(u^n) + G'(u^n)(w - u^n) = 0} Q^n(w)$$

avec

$$Q^n(w) = \left(\mathcal{L}(u^n, \lambda^n) + \mathcal{L}'(u^n, \lambda^n)(w - u^n) + \frac{1}{2} \mathcal{L}''(u^n, \lambda^n)(w - u^n)^2 \right)$$

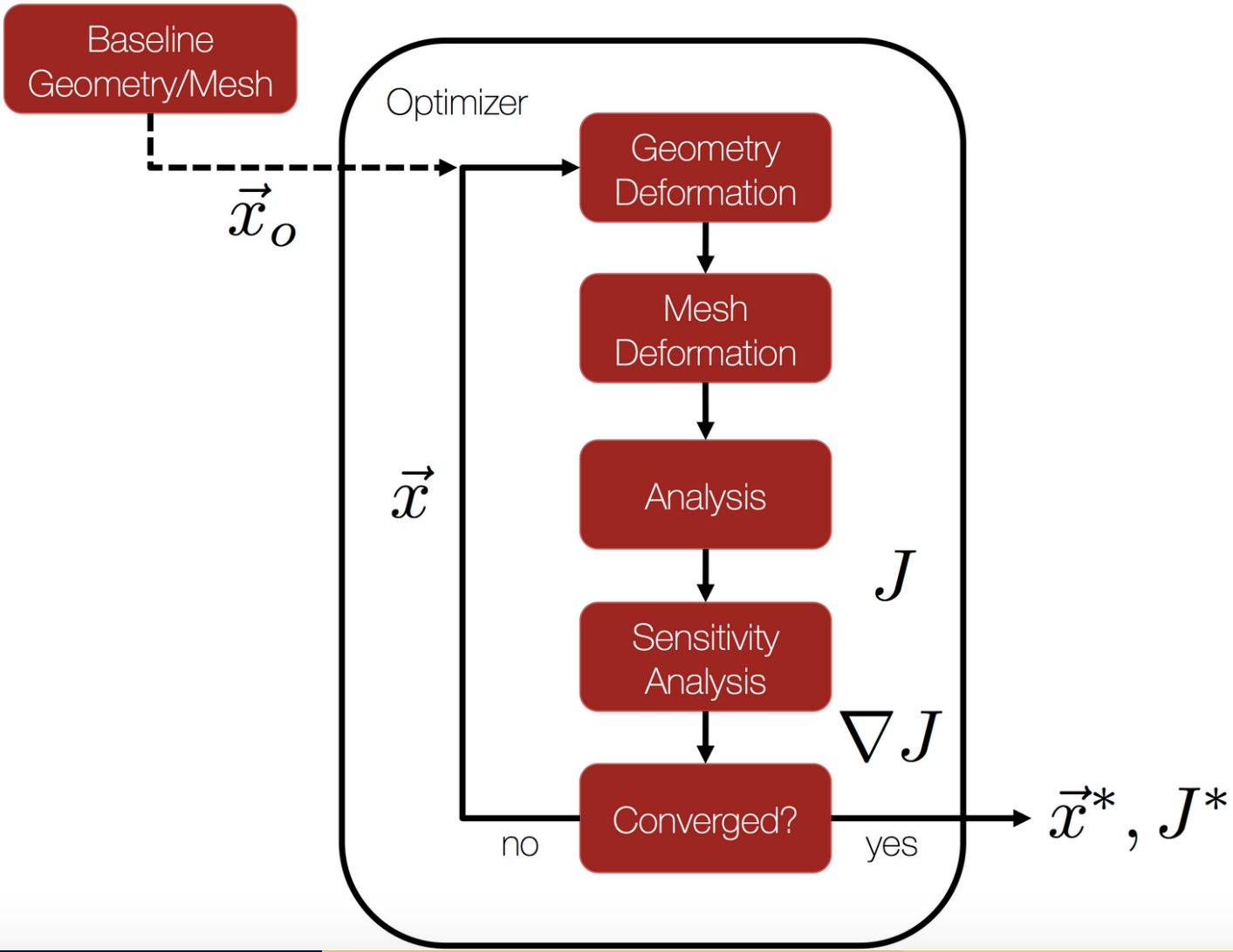
2.6.3 Algorithme d'optimisation dans SU2 : SLSQP

Dans SU2 : l'agorithme d'optimisation le plus souvent utilisé est le **SLSQP** (Sequential Least Squares Quadratic Programmation) (voir papier Dietr Kraft section Notes de Cours).
L'algorithme SLSQP (proche du SQP, problème avec contraintes) :

- Est un algorithm de type Newton
- On cherche la solution u^n du problème d'optimisation en utilisant un sous-problème quadratique puis a partir de cette solution on construit u^{n+1}.
- l'idée de base des constructions de ces sous-problèmes est de remplacer :
  - La fonctionelle par son approximation quadratique :
    $J(u) \approx J(u^n) + J'(u^n)(u - u^n) + \frac{1}{2}(u - u^n)^2 J''(u^n)$
  - Les contraintes par leurs approximations affines

L'algorithme d'optimisation dans SU2

Regardons de plus près l'optimisation de design avec SU2 :



Optimisation sans contrainte : NACA0012

On considère l'écoulement de fluide parfait, non-visqueux, autour d'un profil d'aile NACA0012. On souhaite minimiser le coefficient de traînée (C_D). Il s'agit d'un premier exemple d'optimisation **sans contrainte**.

Copier le maillage et fichier configuration qui se trouve sur Moodle section Code SU2 dans un répertoire local que vous avez crée pour ce problème. Il s'agit du même problème que celle vue en séance 4.

Avant d'apeller le script Python d'optimisation assurez vous que la solution de l'équation d'Euler ainsi que la solution de l'adjoint sont correctes.

Optimisation sans contrainte : NACA0012

Ouvrez le fichier de configuration et regardez les nouvelles options :

```
% ----------------- ADJOINT-FLOW NUMERICAL METHOD DEFINITION -------------------%
% Adjoint problem boundary condition (DRAG, LIFT, SIDEFORCE, MOMENT_X,
%                                     MOMENT_Y, MOMENT_Z, EFFICIENCY,
%                                     EQUIVALENT_AREA, NEARFIELD_PRESSURE,
%                                     FORCE_X, FORCE_Y, FORCE_Z, THRUST,
%                                     TORQUE)
OBJECTIVE_FUNCTION= DRAG
%
% Convective numerical method (JST, LAX-FRIEDRICH, ROE-1ST_ORDER,
%                              ROE-2ND_ORDER)
CONV_NUM_METHOD_ADJFLOW= JST
%
% Slope limiter (VENKATAKRISHNAN, SHARP_EDGES)
SLOPE_LIMITER_ADJFLOW= VENKATAKRISHNAN
%
% 2nd, and 4th order artificial dissipation coefficients
ADJ_JST_SENSOR_COEFF= ( 0.0, 0.02 )
```

Pour voir toutes les fonctionnelles implémentées dans SU2 ouvrez *config_template.cfg* dans le répertoire opt/SU2

Optimisation sans contrainte : NACA0012

Jetons un oeil aux options d'optimisation maintenant :

```
% ------------------- OPTIMAL SHAPE DESIGN DEFINITION ---------------------%
%
% Optimization objective function with scaling factor, separated by semicolons.
% To include quadratic penalty function: use OPT_CONSTRAINT option syntax within the O
PT_OBJECTIVE list.
% ex= Objective * Scale
OPT_OBJECTIVE= DRAG
%
% Optimization constraint functions with pushing factors (affects its value, not the g
radient
% in the python scripts), separated by semicolons
% ex= (Objective = Value ) * Scale, use '>','<','='
OPT_CONSTRAINT= NONE
%
% Factor to reduce the norm of the gradient (affects the objective function and gradie
nt in the python scripts)
% In general, a norm of the gradient ~1E-6 is desired.
OPT_GRADIENT_FACTOR= 1E-6
%
% Factor to relax or accelerate the optimizer convergence (affects the line search in
SU2_DEF)
% In general, surface deformations of 0.01'' or 0.0001m are desirable
OPT_RELAX_FACTOR= 1E3
%
% Maximum number of optimizer iterations
OPT_ITERATIONS= 100
%
% Requested accuracy
OPT_ACCURACY= 1E-10
```

Optimisation sans contrainte : NACA0012

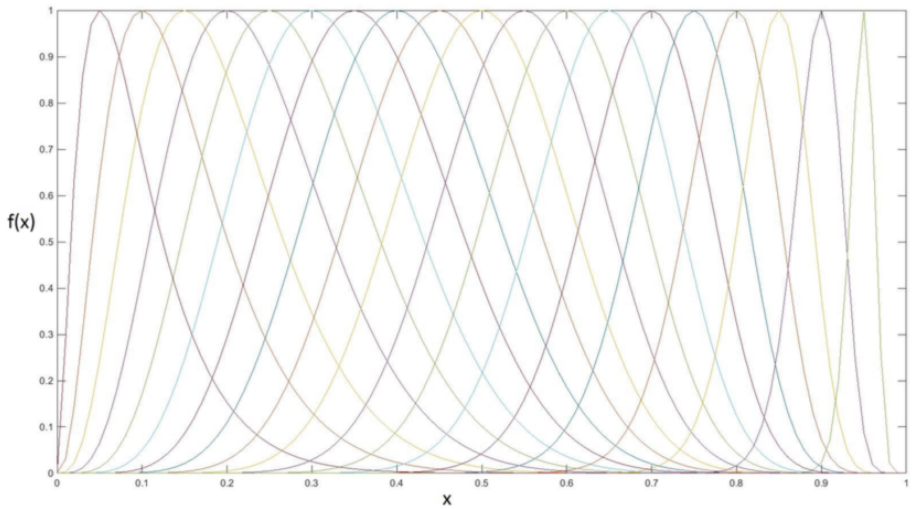
- Le script **shape_optimization.py** se trouve dans votre dossier **bin**. Il utilise l'optimisation SLSQP (sequencial least-square algorithm) de SciPy Python.
- Plusieurs paramètres du fichier configuration seront lues par la méthode : OPT_ITERATIONS (nr. maximal d'itérations), OPT_ACCURACY (précision souhaitée), OPT_BOUND_UPPER, OPT_BOUND_LOWER (bornes des variables de design). Les solveurs adjoint et Euler seront appelés plusieurs fois au cours du processus d'optimisation.

Optimisation sans contrainte : NACA0012

La surface du profil d'aile est paramétrisée et modifié en utilisant un ensemble de fonctions bosse Hicks-Henne, décrites par :

$$f(x) = \left[\sin(\pi x^{\frac{\log 0.5}{\log t_1}})\right]^{t_2}, \quad 0 \leq x \leq 1$$

avec t_1 la location x du maximum et t_2 est la largeur de la bosse.



Optimisation sans contrainte : NACA0012

Et encore :

```
% Upper bound for each design variable
OPT_BOUND_UPPER= 0.1
%
% Lower bound for each design variable
OPT_BOUND_LOWER= -0.1
%
% Optimization design variables, separated by semicolons
DEFINITION_DV= ( 1, 1.0 | airfoil | 0, 0.05 ); ( 1, 1.0 | airfoil | 0, 0.10 ); ( 1, 1.
0 | airfoil | 0, 0.15 ); ( 1, 1.0 | airfoil | 0, 0.20 ); ( 1, 1.0 | airfoil | 0, 0.25
); ( 1, 1.0 | airfoil | 0, 0.30 ); ( 1, 1.0 | airfoil | 0, 0.35 ); ( 1, 1.0 | airfoil
| 0, 0.40 ); ( 1, 1.0 | airfoil | 0, 0.45 ); ( 1, 1.0 | airfoil | 0, 0.50 ); ( 1, 1.0
| airfoil | 0, 0.55 ); ( 1, 1.0 | airfoil | 0, 0.60 ); ( 1, 1.0 | airfoil | 0, 0.65 );
( 1, 1.0 | airfoil | 0, 0.70 ); ( 1, 1.0 | airfoil | 0, 0.75 ); ( 1, 1.0 | airfoil | 0
, 0.80 ); ( 1, 1.0 | airfoil | 0, 0.85 ); ( 1, 1.0 | airfoil | 0, 0.90 ); ( 1, 1.0 | a
irfoil | 0, 0.95 ); ( 1, 1.0 | airfoil | 1, 0.05 ); ( 1, 1.0 | airfoil | 1, 0.10 ); (
1, 1.0 | airfoil | 1, 0.15 ); ( 1, 1.0 | airfoil | 1, 0.20 ); ( 1, 1.0 | airfoil | 1,
0.25 ); ( 1, 1.0 | airfoil | 1, 0.30 ); ( 1, 1.0 | airfoil | 1, 0.35 ); ( 1, 1.0 | air
foil | 1, 0.40 ); ( 1, 1.0 | airfoil | 1, 0.45 ); ( 1, 1.0 | airfoil | 1, 0.50 ); ( 1,
1.0 | airfoil | 1, 0.55 ); ( 1, 1.0 | airfoil | 1, 0.60 ); ( 1, 1.0 | airfoil | 1, 0.6
5 ); ( 1, 1.0 | airfoil | 1, 0.70 ); ( 1, 1.0 | airfoil | 1, 0.75 ); ( 1, 1.0 | airfoi
l | 1, 0.80 ); ( 1, 1.0 | airfoil | 1, 0.85 ); ( 1, 1.0 | airfoil | 1, 0.90 ); ( 1, 1.
0 | airfoil | 1, 0.95 )
```

Optimisation sans contrainte : NACA0012

DEFINITION_DV designe les variables de design separées par ;

La première valeur entre parenthèses est le type de variable, qui est 1 pour une fonction bosse de Hicks-Henne. La deuxième valeur est l'échelle de la variable (généralement laissée à 1,0). Le nom entre les barres verticales est la balise du marqueur où les déformations variables seront appliquées. Seule la surface portante sera déformée dans ce problème. Les deux dernières valeurs entre parenthèses spécifient si la fonction bosse est appliquée au côté supérieur (1) ou inférieur (0) et à l'emplacement x de la bosse entre 0 et 1 (nous supposons une longueur de corde de 1,0 pour les bosses de Hicks-Henne), respectivement.
Notez que de nombreux autres types de variables de design sont disponibles dans SU2 et que chacun a son propre format d'entrée.

Optimisation sans contrainte : NACA0012

• Une paramétrisation permet de réduire la dimension de l'espace dans lequel s'effectue la recherche de la forme optimale.

• Cette forme de paramétrisation a l'avantage de permettre un contrôle local de la forme.

• Ces fonctions se sont révélées particulièrement adaptées pour des calculs d'optimisation de forme avec un nombre assez réduit de paramètres.

Optimisation sans contrainte : NACA0012

Executer le script Python d'optimisation avec la commande :

python shape_optimization.py -g CONTINUOUS_ADJOINT -o SLSQP -f inv_NACA0012_basic.cfg

Beaucoup d'informations seront sauvegardées :
DESIGNS/DSN_*/DIRECT/ contient les solutions des équations d'Euler pour chaque nouveau design
Le fichier *history_project.csv* contient les valeurs de J pour chaque itération dans le processus d'optimisation.

Attention : Python, NumPy, and SciPy sont nécessaire pour l'execution de ce script !

Optimisation sans contrainte : NACA0012

Sauvegardez les résultats (images) suivantes :

- La densité , ainsi que la densité de l'adjoint
- Cp vs. x/c pour le profil initial et le dernier design sur une même figure
- L'évolution de Cd avec les itérations d'optimisation
- La densité sur le design final, a comparer avec celle obtenu sur le design d'origine

Les screenshots de ces résultats seront a déposer sur moodle avant la fin de la séance. Les commentaires de ces résultats sont aussi les bienvenues.

Optimisation avec contrainte : profil d'aile ONERA M6

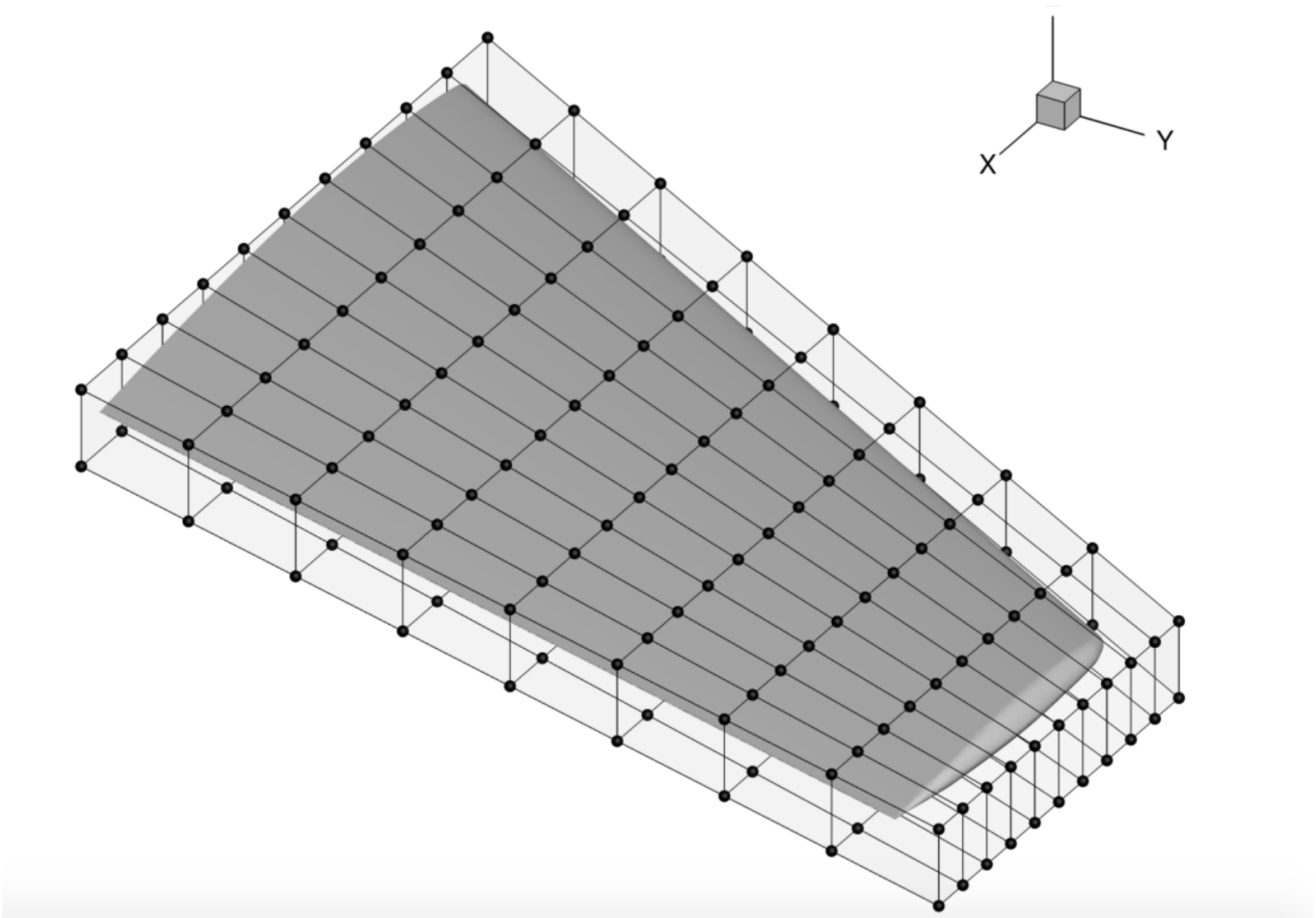
Données du problème (disponibles sur moodle) :

- Fichier configuration inv_ONERAM6_adv.cfg
- Fichier maillage mesh_ONERAM6_inv_FFD.su2 (ce fichier contient déjà les informations concernant la méthode de déformation du profil)

Optimisation avec contrainte : profil d'aile ONERA M6

Le but de ce problème de design optimal est de minimiser le coefficient de traînée en modifiant la forme de l'aile tout en imposant des contraintes sur la portance et l'épaisseur de section d'aile.
Pour les variables de design nous utiliserons une approche de déformation de forme libre (FFD). Dans cette approche, un réseau de points de contrôle constituant un cadre de délimitation est placé autour de la géométrie et le mouvement de ces points de contrôle déforme progressivement la forme de la surface de la géométrie à l'intérieur.
Nous commençons par une géométrie 3D à voilure fixe à vitesse transsonique dans l'air.

Optimisation avec contrainte : profil d'aile ONERA M6



Etude libre (1)

On souhaite maintenant approfondir nos études sur des benchmarks proposés par le groupe de travail AIAA Design Optimization Discussion Group, principalement Case 1 et Case 2.

Choisisez un benchmark a traiter. Vous pouvez apporter des simplifications (sur l'écoulement considéré, la paramétrisation, l'algorithme d'optimisation, etc).

Comparez vos résultats obtenus avec ceux de l'article : "Aerodynamic Shape Optimisation of Benchmark Problems Using SU2" (Guangda Yang and Andrea Da Ronch, AIAA 2018)

Etude libre (2)

Des études supplémentaires et intéressantes sont aussi proposés ici par Jameson et Vassberg, nottament pour l'optimisation de design de NACA0012