Rapport de projet

Projet d'option N°16 :
Traction de navire par Kite

Jean CRESP
Abel PRUCHON
Jiakan ZHOU

Encadrant : Mr Baptiste LABAT

Table des matières

Introduction

L'objectif de ce projet est de réaliser un calcul de VPP (Velocity Prediction Program) pour le kiteboat AK650, développé par le projet ArmorKite.

Dans un premier temps, plus que l'aspect performance, nous devions chercher à dimensionner les efforts qui s'exercent sur l'ensemble du système. Nous avons choisi d'utiliser pour cela un modèle simplifié que nous codons en langage Python.

Dans un 2e temps, une fois notre modèle validé, nous pourrons nous devions nous intéresser à l'étude de l'influence des paramètres architecturaux du bateau (position du point de traction, position des dérives, angle de calage des dérives, etc..) dans le but d'optimiser la performance.

Ce rapport final comporte 2 grandes parties : la première correspond au premier modèle choisi et présenté à la mi-soutenance que nous avons finalement abandonné pour celui présenté dans la 2e partie.
Dans chacune des 2 parties nous présenterons les modèles dans leur globalité puis les résultats obtenus.

Présentation du problème

Le projet Armorkite

Armorkite est un projet visant à concevoir un bateau tracté par un voile de kite. L'objectif final serait d'équiper toutes sortes de navires avec ce genre de voile, de la coque de 5 m au porte-conteneurs géants.

L'AK650 d'ArmorKite, le bateau sur lequel nous travaillons, est un bateau d'une longueur de 6,50 m [1]. Il est tracté par une aile de kite gonflable allant de 13 à 25 m2 et atteint des vitesses de plus de 12 nœuds. C'est un bateau très rapide donc et très réactif en barre, il est léger et sans quille. Les lignes de la voile de kite sont directement attachées au dessus de la coque du navire sur un petit rail. En tirant sur les différentes lignes, on est capable de reproduire les manœuvres classiques de kitesurf. Le bateau possède également un moteur d'appoint en cas de souci.

Figure 1 : *Navigation sur le kiteboat AK 650*

Les différentes caractéristiques du bateau intéressantes sont répertoriées dans le tableau ci-dessous (Fig. 2). Les données ont directement été extraites de la fiche technique du kiteboat AK 650. Nous avons également reçu un fichier Rhinocéros confidentiel du bateau.

Bateau	
Longueur coque	6,50m
Longueur à la flottaison	6,05m
Bau maxi	2,22m
Tirant d'eau	0,07m
Déplacement lège	300kg
Voilure	
Lignes avants	En Y : 2/3mm
Lignes arrières	2*2mm
Ligne Sécurité	2mm
Surface d'aile petit temps	25m^2
Surface d'aile moyen temps	18m^2
Surface d'aile gros temps	13m^2
Surface de voile au près / déplacement	80m^2/T
Surface de voile au portant / déplacement	80m^2/T

Figure 2 : *Caractéristiques du kiteboat AK 650 et de son système de voilure (d'après [1])*

Première approche : approche dynamique

Mise en équation

Durant la première partie de ce projet, nous nous sommes donc attelés à modéliser le problème de la manière la plus crédible possible. Nous nous sommes inspirés pour cela des travaux qui avaient déjà été réalisés sur des sujets similaires, notamment le projet d'option Océan « Récupération de l'énergie éolienne par des ailes de kite » datant de 2013-2014 déjà en collaboration avec Monsieur Labat. Cette première approche était basée sur **une étude dynamique** du problème, avec une résolution temporelle des équations.

D'après [2] et [3], On introduit le système de coordonnées lié au navire (x, y, z).
A l'origine, le système lié au navire (x_1, y_1, z_1) a des axes parallèles par rapport au référentiel galiléen fixe (X_E, Y_E, Z_E).

On définit également :

- Lacet : ψ rotation autour de z_1
- Tangage : θ rotation autour de y_2
- Roulis : ϕ rotation autour de x_3

On fait d'abord l'embardée, puis le tangage et on finit par le roulis. On a $z_1 = z_2$, $y_2 = z_3$ et $x_3 = x$. Soit V la vitesse du navire du centre de gravité du bateau dans le référentiel du navire, (u, v, w) ses coordonnées.

On a : $\frac{dV}{dt} = \begin{pmatrix} \frac{du}{dt} \\ \frac{dv}{dt} \\ \frac{dw}{dt} \end{pmatrix} + \Omega.V$ (Voir thèse pour explications) avec $\Omega = \begin{pmatrix} p \\ q \\ r \end{pmatrix}$ l'angle de vitesse angulaire du navire.

Soit M la masse du navire et $\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$ les forces hydrodynamiques et aérodynamiques extérieures. En décomposant cette équation et en appliquant la 2nde loi de Newton, on obtient :

$$\begin{cases} M(\dot{u} + qw - rv) = X - Mgsin\theta \\ M(\dot{v} + ru - pw) = Y + Mgcos\theta.sin\phi \\ M(\dot{w} + pv - qu) = Z + Mgcos\theta.cos\phi \end{cases} \quad (1)$$

On peut obtenir une autre équation en travaillant sur les moments. On suppose que le bateau est symétrique suivant l'axe 0xz. On obtient :

$$\begin{cases} I_{44}\dot{p} - (I_{55} - I_{66})qr - I_{64}(\dot{r} + pq) = L \\ I_{55}\dot{q} - (I_{66} - I_{44})rp - I_{64}(r^2 - p^2) = M \\ I_{66}\dot{r} - (I_{44} - I_{55})pq - I_{64}(\dot{p} - qr) = N \end{cases} \quad (2)$$

Où I_{ij} est le produit de l'inertie respectant le système de coordonnées (x, y, z) et $\begin{pmatrix} L \\ M \\ N \end{pmatrix}$ les moments des forces externes au navire.

Pour résoudre les équations d'Euler ci-dessus, nous devons dériver les équations qui décrivent l'orientation du navire. On obtient finalement :

$$\begin{pmatrix} \frac{dX_E}{dt} \\ \frac{dY_E}{dt} \\ \frac{dZ_E}{dt} \end{pmatrix} = ABC \begin{pmatrix} u \\ v \\ w \end{pmatrix} \text{ avec } A = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}, B = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

et $C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}$.

Dans notre programme afin d'optimiser les temps de calcul, on travaillera en théorie des petits angles. Les matrices utilisées seront donc :

$$A = \begin{pmatrix} 1 & -\psi & 0 \\ \psi & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, B = \begin{pmatrix} 1 & 0 & \theta \\ 0 & 1 & 0 \\ -\theta & 0 & 1 \end{pmatrix} \text{ et } C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -\phi \\ 0 & \phi & 1 \end{pmatrix}.$$

Notations

Nous allons maintenant présenter les différentes notations utilisées pour résoudre ce problème :

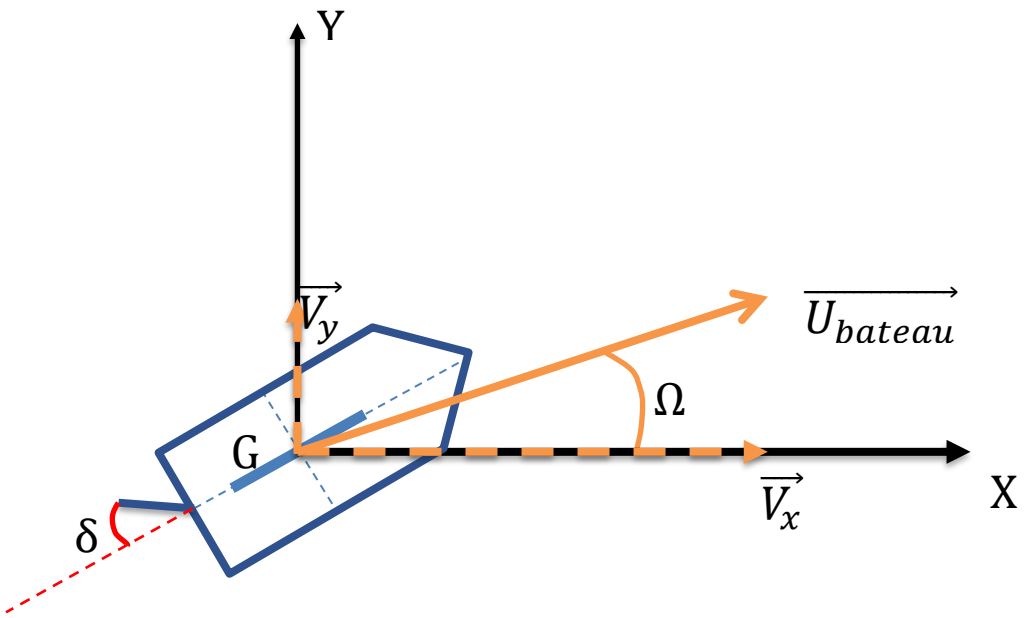


Figure 3 : *Notations utilisées dans la suite du problème (Première approche)*

Ces notations sont celles utilisées dans le programme python. U_{bateau} n'est pas aligné avec l'axe du bateau car il y a un effet de dérive qui entre en jeu. δ est l'angle de barre qui intervient lors de la force due au gouvernail.

Hypothèses

On fixe un cap au bateau et on regarde la position optimale du kite.

Pour simplifier le problème on réalise un certain nombre d'hypothèses récapitulées ci-dessous :

- Voile et fils du kite sans masse ;
- La voile de kite est assimilée un rectangle indéformable ;
- On considérera que la navire ne possède qu'une seule dérive pour l'instant (contre 2 symétriques sur le bateau d'Armorkite);
- La dérive et le kite ont un coefficient de moment nuls ;
- Le kite est relié au bateau par une seule ligne à un point fixe, au centre de gravité du navire (on s'intéressera par la suite au changement de performances si on l'attache à un autre endroit) ;
- Le bateau navigue sur un plan de mer infini, de profondeur infinie et au repos ;
- On se limite à une étude 2D pour l'instant : l'étude suivant l'axe z est peu intéressante donc on ne la mène pas ;
- Petits angles de giration ;
- On néglige pour l'instant les effets hydrodynamiques en x et y : c'est faux car la coque n'est pas une sphère il faut les trouver avec une étude ;

Forces en présence

- *La force de frottement du navire sur l'eau :*

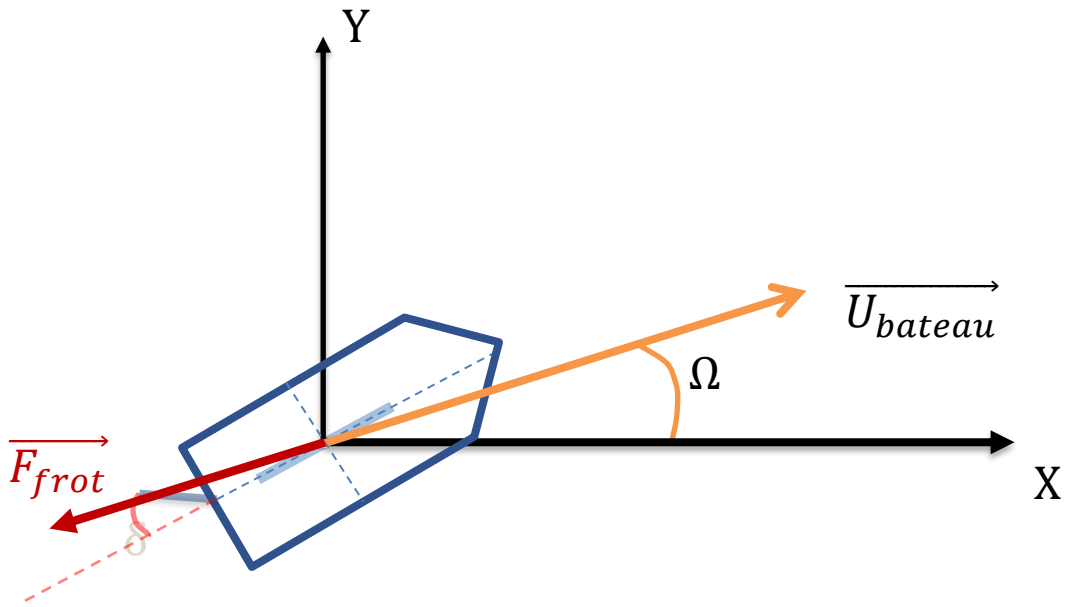


Figure 4 : *Force de frottement sur le navire (Première approche)*

Cette force est dirigée suivant la même direction que $\overrightarrow{U_{bateau}}$, et dans la direction opposée à celle-ci. On a :

$\overrightarrow{\boldsymbol{F_{frot}}} = -\frac{1}{2}\boldsymbol{\rho_{eau}} C_f \boldsymbol{S U_{bateau}} \overrightarrow{\boldsymbol{U_{bateau}}}$ avec Cf, le coefficient de frottement qui dépend du nombre de Reynolds selon la relation donnée par la loi ITTC 57 : $C_f = \frac{0.075}{(\log(Re)-2)^2}$.

- *La force due la dérive du navire :*

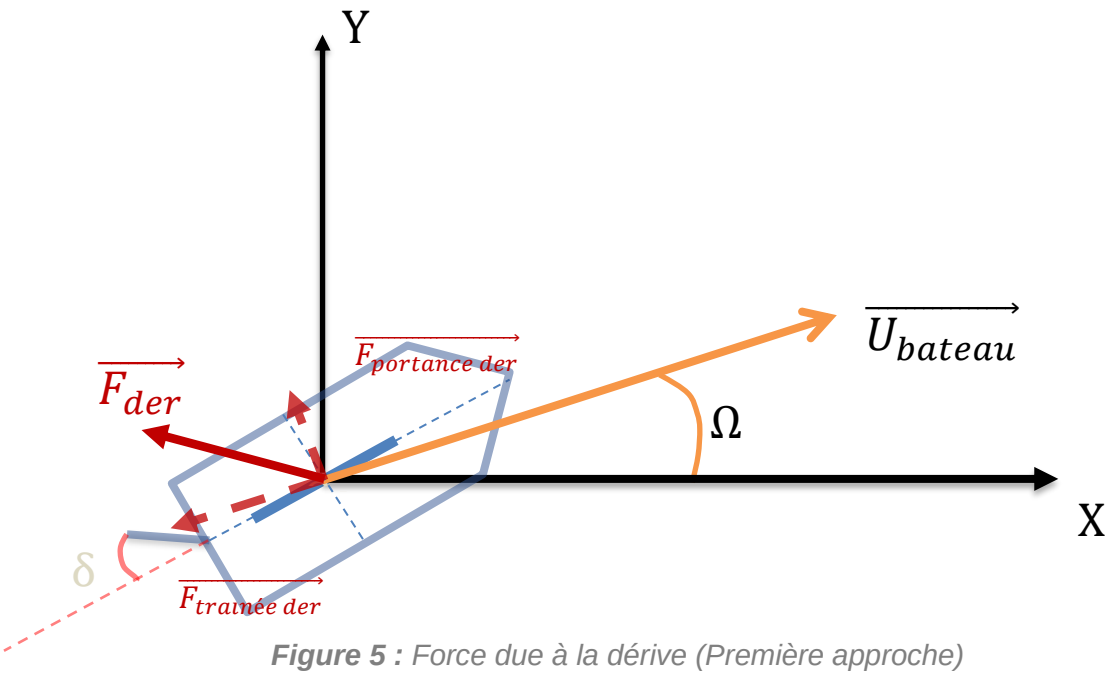


Figure 5 : *Force due à la dérive (Première approche)*

Pour calculer cette force, nous avons fait l'hypothèse qu'il n'y avait qu'une seule dérive, centrée au point G du navire, et qu'on pouvait la modéliser par un profil portant avec une force de portance et une force de trainée. On a pris un profil NACA 0012, et nous avons récupéré les différents coefficients à l'aide de Xfoil.

- *La force due au gouvernail :*

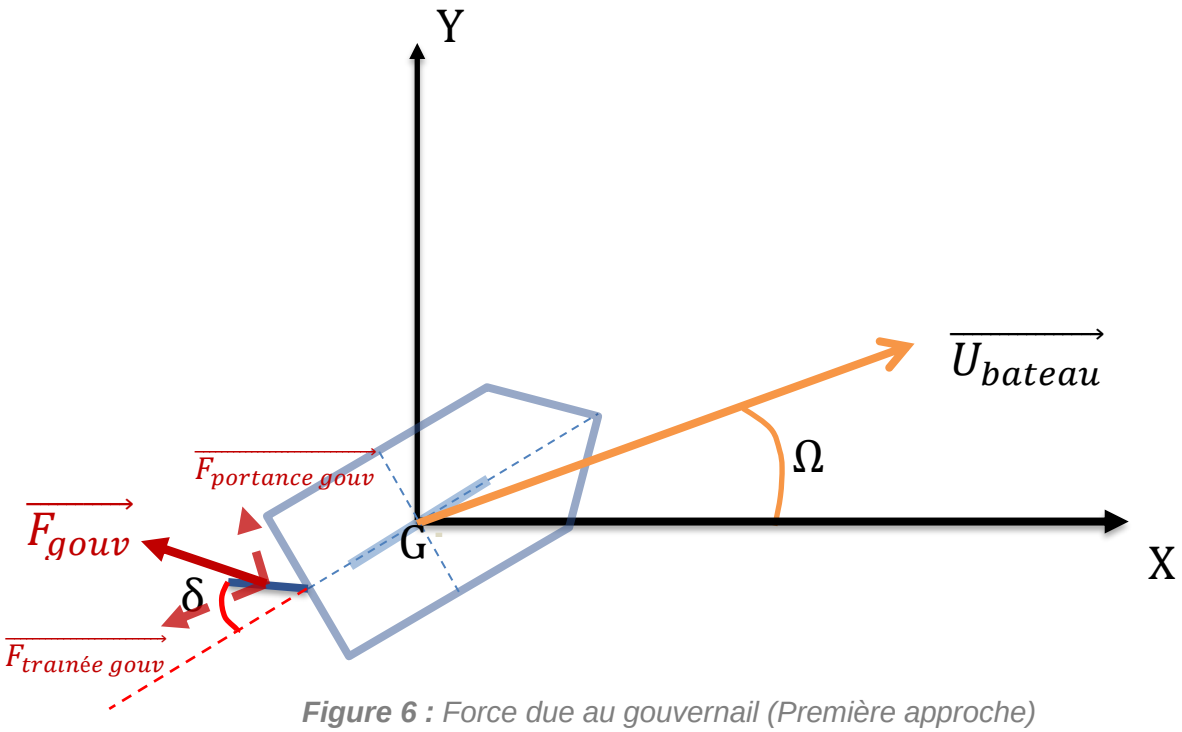


Figure 6 : *Force due au gouvernail (Première approche)*

Là encore, nous avons fait l'hypothèse que le gouvernail était un profil portant, qui a la même géométrie que la dérive. De plus, nous avons fait en première approximation l'hypothèse que la force s'exerce au milieu du gouvernail, ce qui entraîne l'apparition d'un moment autour de l'axe Z.

Définition de l'équilibre du kite et force exercée par le kite sur le navire :

En pratique, une voile de kite ne reste jamais à l'équilibre. En effet, certains mouvements de la voile permettent à la voile de prendre plus d'air et donc d'augmenter ses performances. Ici on considère, d'après [4], qu'il existe bien une position d'équilibre définie par un équilibre des moments et des forces s'exerçant sur le kite. Cela se traduit par : $\overrightarrow{F_{câble}} + \overrightarrow{F_{kite}} = \vec{0}$ (hypothèse d'une voile sans masse). De plus, comme l'aile de kite se comporte comme un profil portant, la force du kite se décompose en une force de portance et une force de traînée.

Ces deux forces vont créer un moment au point G du navire car il y a un bras de levier due à la longueur du câble reliant la voile au navire. Nous avons alors dit que le kite était à l'équilibre lorsque ces deux moments se compensent.

En réalité, nous avons fait l'hypothèse que le bord d'attaque du kite est toujours parallèle à la surface de la mer, et donc il n'y a que la force de trainée qui intervient dans nos calculs. On pourra ensuite modifier cela pour prendre en compte l'inclinaison du kite.

Il y a un autre paramètre qu'il est important de prendre en compte dans nos calculs, c'est le phénomène de *vent apparent*.

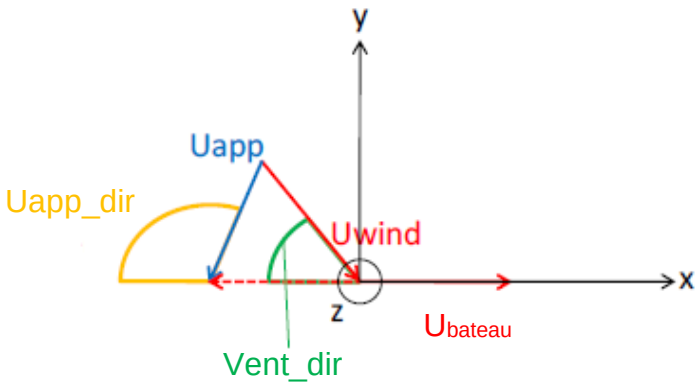


Figure 7 : *Schéma explicatif du vent apparent*

En effet, lors du déplacement du navire, le vent apparent n'est pas le même que le vent extérieur, et c'est celui qui nous intéresse pour calculer la force créée par le kite. C'est ainsi qu'on a obtenu le résultat suivant dans le plan xOy (le plan de la mer) :

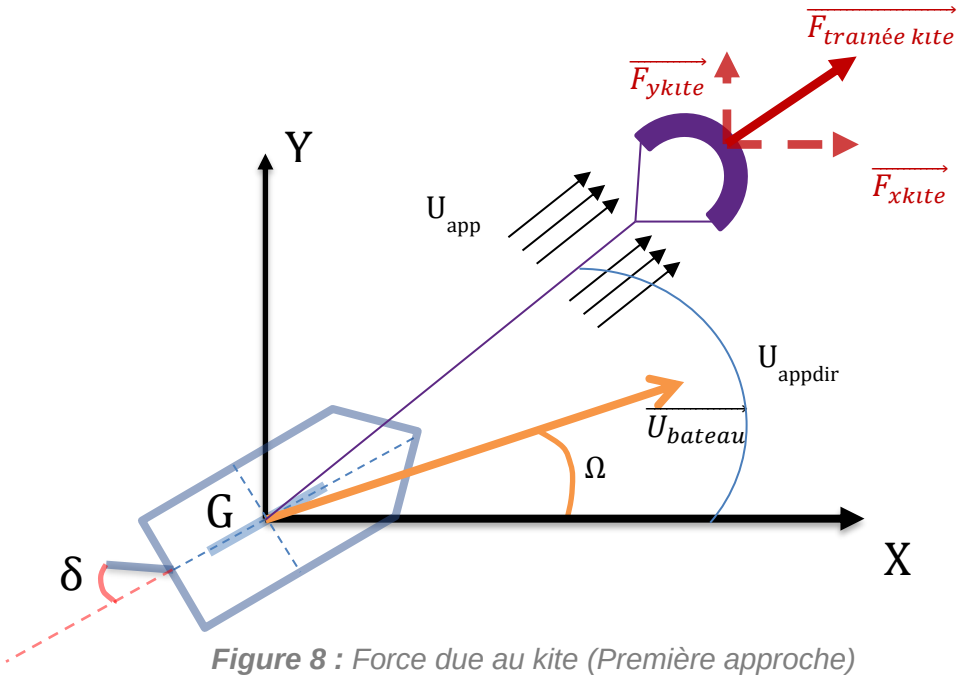


Figure 8 : *Force due au kite (Première approche)*

La force est orientée dans la direction du vent apparent. En effet, nous avons fait l'hypothèse que le kite s'alignerait toujours avec le vent, mais ce n'est pas une hypothèse valable car c'est l'un des paramètres majeurs pour régler le kite et optimiser l'effort reçu par le navire.

Résultats des modélisations

Après avoir fait toutes ces hypothèses, nous avons pu modéliser le problème sous python. Nous avions mal compris ce qui était demandé, et nous avons alors chercher à modéliser l'évolution temporelle jusqu'à un état d'équilibre du navire plutôt que de calculer directement l'équilibre des forces.

Pour cela, nous avons défini une classe 'vecteur', est nous avons résolu le système d'équation de la manière suivante, entre les instant t et t+dt :

-bateau.vit(t+dt)=bateau.vit(t)+bateau.acc(t) *dt

-calcul des nouvelles forces avec les nouvelles vitesses

-résolution du système A*bateau.acc(t+dt)=b(t+dt) avec :

$$A = \begin{pmatrix} M & 0 & 0 & 0 & 0 & 0 \\ 0 & M & 0 & 0 & 0 & 0 \\ 0 & 0 & M & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{44} & 0 & I_{46} \\ 0 & 0 & 0 & 0 & I_{55} & 0 \\ 0 & 0 & 0 & I_{64} & 0 & I_{66} \end{pmatrix}$$

$b(t+dt) = vecteur\ des\ efforts$ à $t+dt$ $obtenue$ $grâce$ à $(1) et (2)$

Nous avons alors obtenu différents résultats au cours des modélisations :

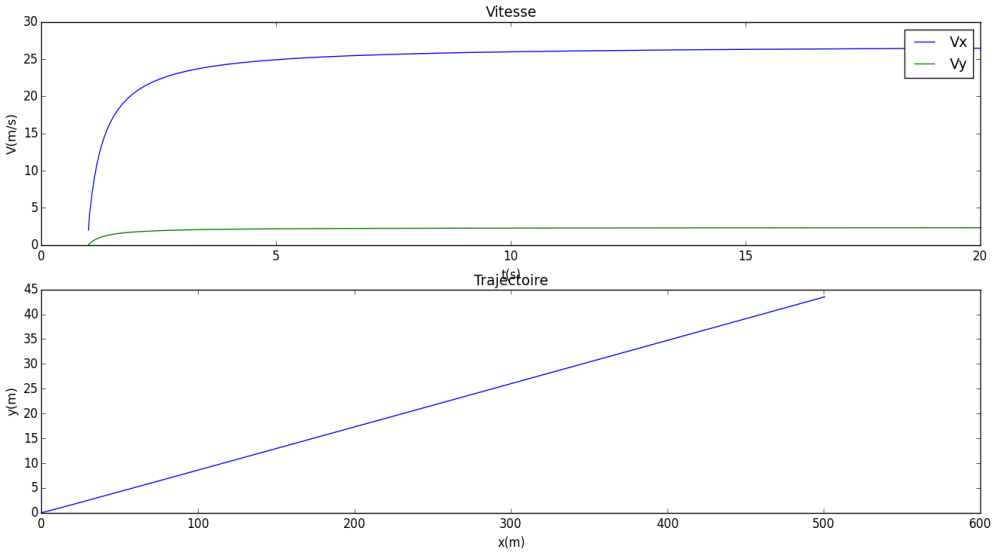


Figure 9 : *Vitesses et trajectoire du navire*

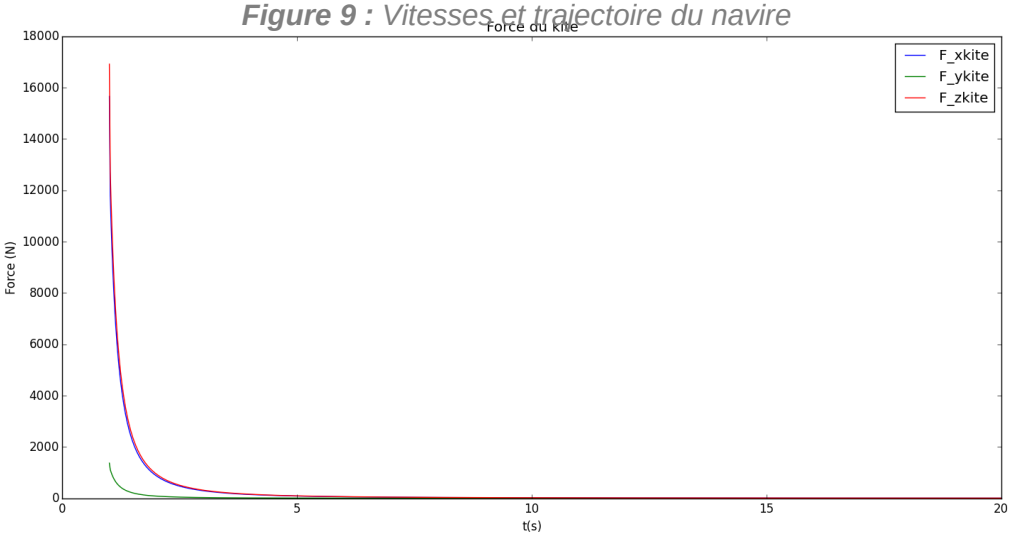


Figure 10 : *Forces du kite sur le navire en fonction du temps*

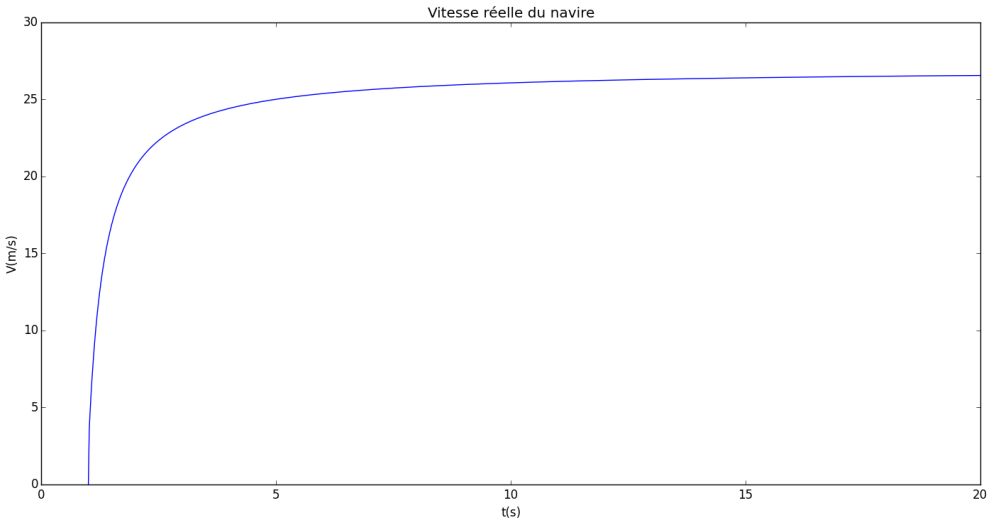


Figure 11 : *Vitesse du navire en fonction du temps*

Ces résultats ont été obtenus en prenant uniquement en compte la force de frottement et la force du kite, pour un vent orienté suivant un angle de 5° pour une vitesse de 15 nœuds. Nous voyons dans un premier temps que le problème converge bien vers un état d'équilibre dans ce cas-là, les forces en jeu convergent, tout comme les vitesses. De plus, la trajectoire du navire correspond bien à la trajectoire d'un objet tirés par une voile suivant un angle de 5°. Enfin, la vitesse du navire est légèrement inférieure à la vitesse du vent, ce qui est normal car nous sommes dans le cas d'un vent arrière (kite aligné avec le vent). Les résultats des autres modélisations (avec le gouvernail et la dérive) divergeaient totalement d'où la mise en place d'une nouvelle approche.

Deuxième approche : approche statique

Explication du changement de modèle

Suite à la réunion de mi-projet, nous sommes arrivés à la conclusion que nous n'étions absolument pas partis dans la bonne direction. En effet, la résolution d'un problème dynamique est bien trop compliquée, nous devions approximer les inerties car nous ne les connaissions pas. Notre de méthode de résolution était explicite, et donc instable pour les pas de temps choisis. De plus, les notations que nous avions choisies n'étaient pas les plus adaptées car notre repère était plus ou moins fixé au hasard, il ne correspondait pas à quelque chose de physique.

Il a donc fallu repenser tout le problème quasiment depuis le début. Le fait de changer de repère modifiait l'expression des forces, et donc il fallait les adapter au nouveau problème. De plus, on a choisi une autre méthode de résolution. Cette méthode permet d'optimiser un problème directement, on n'a plus l'aspect dynamique qui entre en jeu. Le problème revient uniquement à résoudre des équations en optimisant certains paramètres (ici la composante de la vitesse du bateau suivant l'axe défini par le cap du bateau).

Notations

Pour commencer, nous avons choisi un nouveau repère qui était fixé par rapport à une donné du problème : la direction du vent. En effet, comme on a fait l'hypothèse que la direction du vent ne changeait pas au cours du temps, il est plus logique qu'elle définisse notre référentiel.

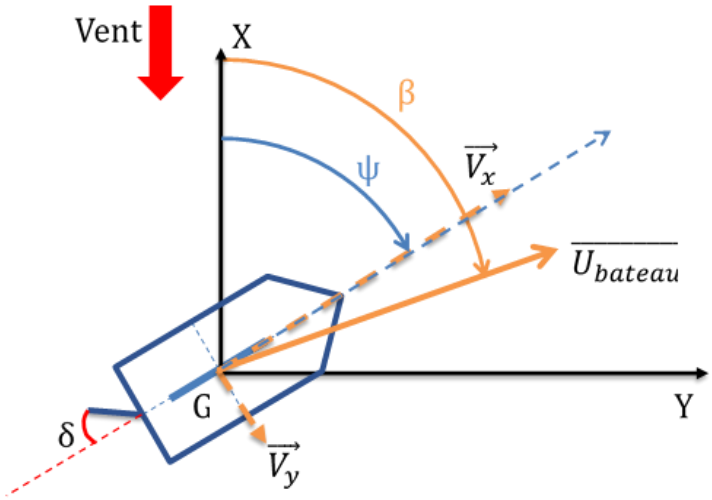


Figure 12 : *Notations utilisées pour la seconde approche*

Ensuite, on a repris différentes notations pour la vitesse du navire, avec encore une fois $\overrightarrow{U_{bateau}}$ la vitesse absolue du navire, et $\overrightarrow{V_x}$ *et* $\overrightarrow{V_y}$ les vitesses respectivement longitudinale et transversale du navire. Enfin nous avons défini les différents angles du problème : δ l'angle de barre, ψ l'angle de cap du navire par rapport au vent et β l'angle de la vitesse du bateau par rapport au vent.

Hypothèses

Pour ce nouveau problème, on a fait plusieurs hypothèses, la plupart identiques aux hypothèses précédentes.
On fixe un cap au bateau et on regarde la position optimale du kite.

Pour simplifier le problème on réalise un certain nombre d'hypothèses récapitulées ci-dessous :

- Voile et fils du kite sans masse ;
- La voile de kite est assimilée un rectangle indéformable ;
- La dérive et le kite ont un coefficient de moment nuls ;
- Le kite est relié au bateau par une seule ligne à un point fixe, au centre de gravité du navire (on s'intéressera par la suite au changement de performances si on l'attache à un autre endroit) ;
- Le bateau navigue sur un plan de mer infini, de profondeur infinie et au repos ;

- On se limite à une étude 2D pour l'instant : l'étude suivant l'axe z est peu intéressante donc on ne la mène pas ;
- On considère que les forces dues à la dérive et au gouvernail sont perpendiculaires aux plans de ces éléments ;
- Le kite est toujours aligné dans le sens du vent apparent dans un premier temps, et seule la trainée entre en jeu (kite parallèle au sol).
- Les frottements sont uniquement dirigés dans le sens opposé à la vitesse
- Le nombre de Reynolds utilisé pour calculer les coefficients de plan porteurs est considéré constant.

Résolution du problème

Forces en présence

- *La force de frottement du navire sur l'eau :*

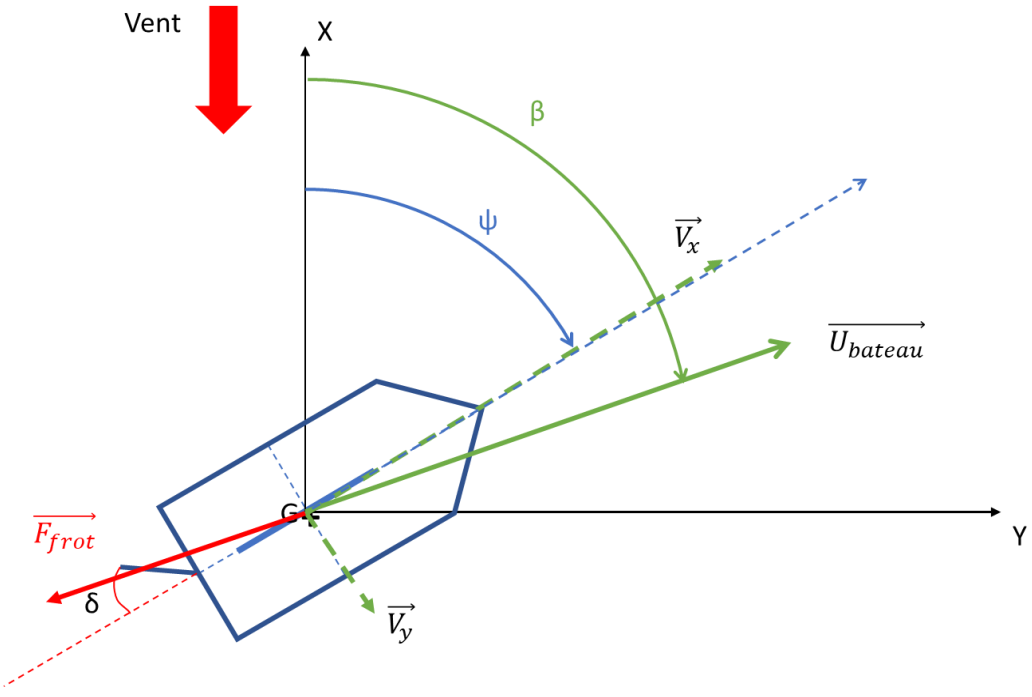


Figure 13 : *Force de frottement (Deuxième approche)*

Comme pour la première approche, nous avons supposé que la force de frottement s'opposait directement à la vitesse du navire. Ensuite, nous avons repris le coefficient de frottement précédent de la forme : $C_f = \frac{0.075}{(\log(Re)-2)^2}$ qui montre l'influence de Re sur cette force. La force est finalement de la forme :

$$\overrightarrow{F_{frot}} = -\frac{1}{2}\rho_{eau} C_f S \overrightarrow{U_{bateau}} U_{bateau}$$

- *La force due à la dérive du navire :*

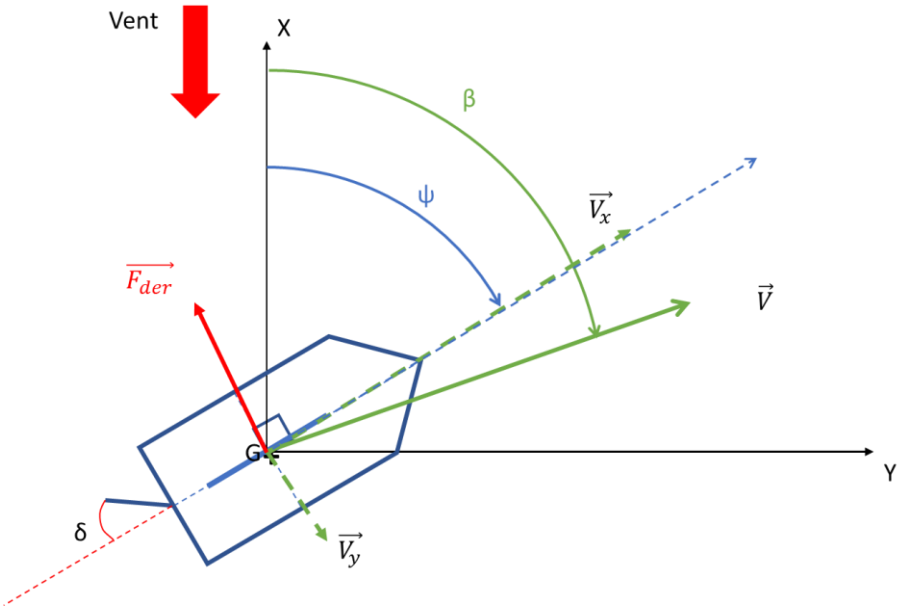


Figure 14 : *Force due à la dérive (Deuxième approche)*

Comme énoncé précédemment, nous avons fait l'hypothèse que la résultante de cette force était normale à la dérive. On a fait cette hypothèse sur les conseils de notre encadrant Mr Labat et nous avons pu vérifier que c'était une hypothèse correcte après avoir calculé la portance et la trainée de notre dérive pour les angles considérés.

Ainsi, pour calculer la norme de notre force, nous avons sommé les forces de trainée et de portance de la dérive, et nous les avons projetées sur l'axe normal à la dérive. Nous avons alors obtenu l'expression de la force de dérive suivante :

$$\overrightarrow{\mathbf{F_{der}}} = \frac{1}{2}\rho_{eau} S_{der} \left\|\vec{V}\right\|^2 \sqrt{C_l(i)^2 + C_d(i)^2}(\sin\psi\vec{x} - \cos\psi\vec{y})$$ avec i=β -ψ

L'expression de $\overrightarrow{F_{der}}$ est valable pour ψ<180°. Il faut inverser le signe sinon.

- *La force due au gouvernail du navire :*

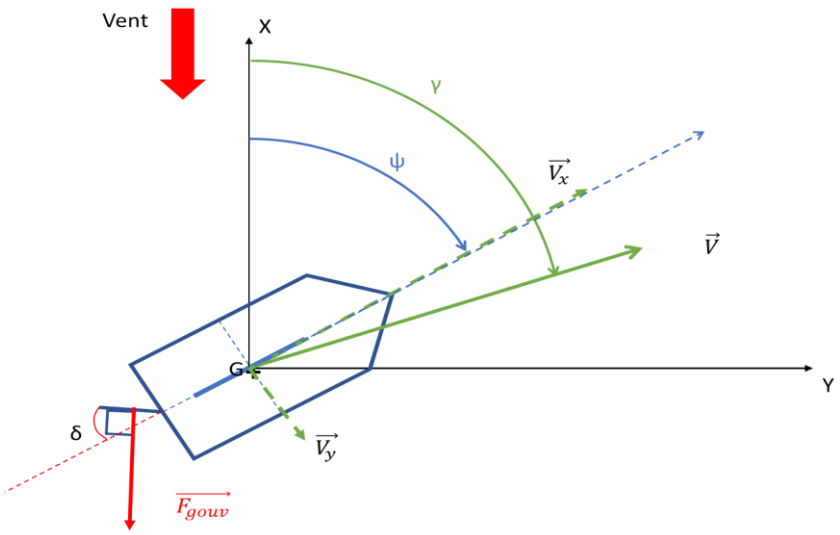


Figure 15 : *Force due au gouvernail (Deuxième approche)*

Là encore, nous avons fait les mêmes hypothèses que pour la dérive afin de faciliter les calculs. Nous sommes encore partis du principe qu'il s'agissait d'un profil portant (type NACA 0015). Nous avons fait une hypothèse supplémentaire : la force s'exerce au milieu du gouvernail, ce qui entraîne la encore l'apparition d'un moment.

D'après [5], nous avons ces expressions :

$$\overrightarrow{\mathbf{F_{gouv}}} = \frac{1}{2}\boldsymbol{\rho_{eau} S_{gouv} \|\vec{V}\|^2} \sqrt{\mathbf{C_l(i)'^2 + C_d(i)'^2}}\ \mathbf{sin(\delta)(sin(\psi + \delta)\vec{x} - cos\,(\psi + \delta)\vec{y})} \text{ avec i= } \beta\text{-}\psi\text{-}\delta$$

&

$$\overrightarrow{\boldsymbol{m_{gouv}}} = \overrightarrow{\boldsymbol{F_{gouv}}} \left(-\frac{\boldsymbol{L}}{2}\mathbf{cos}(\boldsymbol{\delta}) - \frac{\boldsymbol{L_{gouv}}}{2}\right)\vec{\boldsymbol{z}} \text{ avec } \vec{z} \text{ orienté vers le bas}$$

L'expression de $\overrightarrow{F_{gouv}}$ est valable pour $\psi<180°$. Il faut inverser le signe sinon.

Nous verrons par la suite que nous avons n'avons pas pris ce moment en compte dans nos calculs car il n'était pas nécessaire.

Afin d'avoir des coefficients plausibles pour ces 2 dernières forces, nous avons simulé des portances et des traînées sur 360° du logiciel QBlade (en fait sur [-20°, 20] puis extrapolées à l'aide d'une méthode directement implémentée sur le logiciel (Montgomerie)). On considère les 2 profils comme des NACA 0015 (on les différencie juste par leur surface au final). Nous avons pris un Reynolds représentatif de notre bateau (pour une vitesse de 10 m/s qui est la vitesse du vent dans toutes nos simulations). On pouvait alors facilement récupérer la valeur des coefficients pour un angle donné grâce à une interpolation linéaire. On obtient les 2 courbes suivantes :

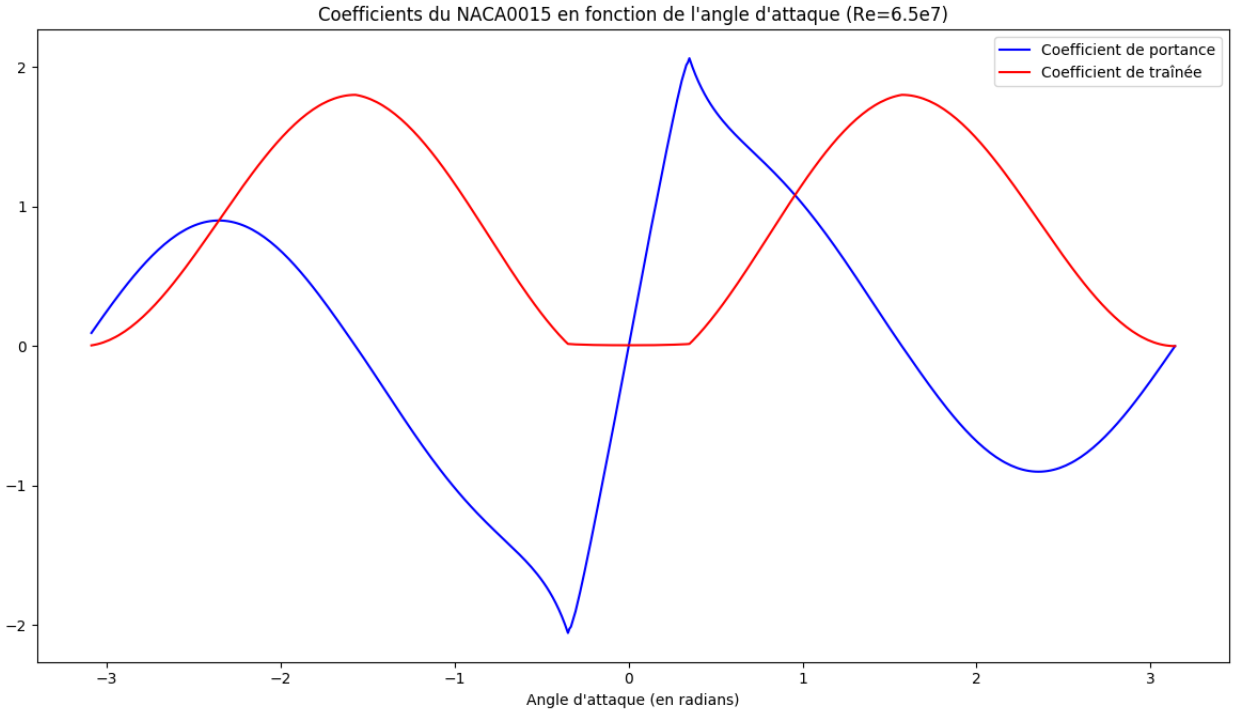


Figure 16 : *Coefficients du NACA0015 en fonction de l'angle d'attaque*

- *La force du kite sur le navire :*

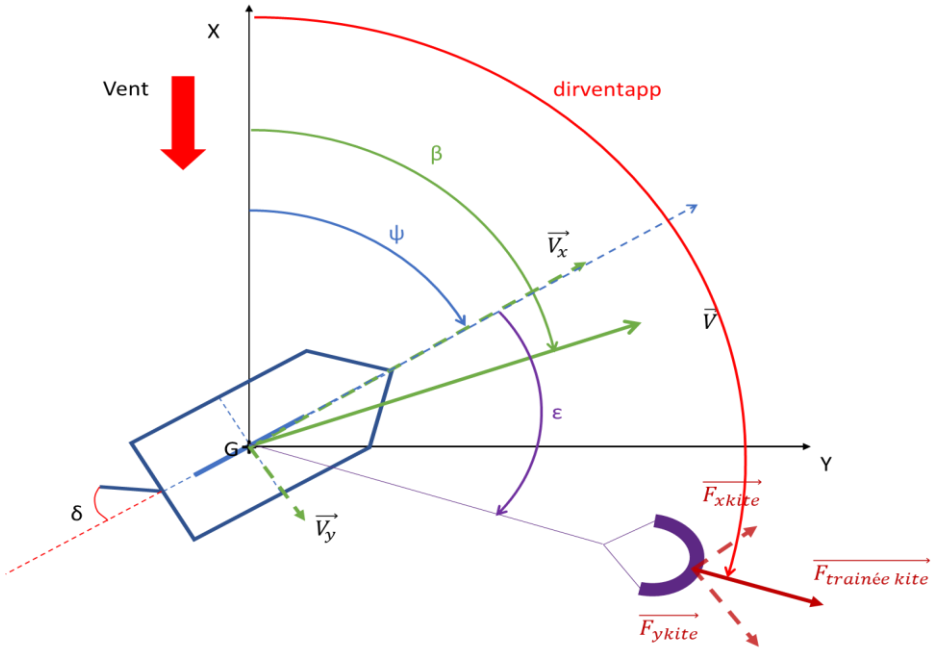


Figure 17 : *Force due au kite (Deuxième approche)*

$$\overrightarrow{\mathbf{F_{kite}}} = \frac{1}{2}\boldsymbol{\rho_{air} S_{kite}} \left\|\overrightarrow{U_{app}}\right\|^2 \mathbf{C_x}\ (\mathbf{cos(dirventapp)}\vec{x} + \ \mathbf{sin(dirventapp)})\vec{x}\text{-}$$

Avec :

$$\left\|\overrightarrow{U_{app}}\right\| = \sqrt{(U_{bateau} sin(\beta))^2 + (U_{vent} + U_{bateau} cos(\beta))^2}$$

$$\mathbf{cos(dirventapp)} = \frac{-(U_{vent} + U_{bateau} cos(\beta))}{\left\|\overrightarrow{U_{app}}\right\|}$$

$$\mathbf{sin(dirventapp)} = \frac{-(U_{bateau} sin(\beta))}{\left\|\overrightarrow{U_{app}}\right\|}$$

Nous avons séparé les cos et sin car cela permettait de remonter facilement à l'angle dans nos calculs sans passer par des Arctan, Arccos ou Arcsin où nous aurions due nous soucier de problèmes de continuité.

Cette fois-ci nous avons fait l'hypothèse que le bord d'attaque du kite était parallèle à la mer ce qui revient à dire que seule la force de trainée entre en jeu et sa valeur dépend de l'angle d'incidence. C'est une hypothèse forte, qui consiste uniquement à bloquer un angle de rotation du kite. Cela signifie qu'on va uniquement chercher à optimiser l'angle d'incidence du kite dans un premier temps. Ensuite, on a aussi fait l'hypothèse que la ligne du kite allait toujours s'aligner avec la direction du vent apparent ce qui est physiquement impossible et faux mais cela simplifiait grandement les calculs dans un premier temps. Nous avons donc prix un C_x constant et égal à 0.01 et une surface du kite de prise au vent également constante. Il aurait également fallu optimiser l'angle d'incident du kite.

Nous ne reviendrons pas sur la notion d'équilibre du kite car cela ne change pas de la première approximation, nous avons juste respecté l'équilibre des forces projeté dans le plan de la mer. De même, la définition de vent apparent reste identique.

Méthode d'optimisation

Pour résoudre notre problème, nous avons utilisé la méthode de l'*optimisation quadratique successive*, et nous avons pu utiliser directement la fonction de python. Cet algorithme se base grossièrement sur l'algorithme de Newton qui revient à chercher le zéro d'une fonction à l'aide de sa dérivée. En réalité, la fonction utilisée a le grand avantage de pouvoir nous laisser la possibilité de rajouter des contraintes à notre problème tout en optimisant les fonctions voulues.

Toutefois, pour que cet algorithme fonctionne, il faut respecter quelques critères. Dans un premier temps, comme pour la méthode de Newton, il faut initialiser la fonction avec un vecteur X_0 qui doit se trouver suffisamment proche de la solution finale pour pouvoir converger vers celle-ci. Ensuite, il faut qu'il y ait un nombre de contraintes inférieur ou égal au nombre de paramètres entrant en jeu dans nos équations. Enfin, il faut un nombre d'itérations suffisant pour avoir une approximation correcte de la solution optimale lorsqu'elle existe.

En réalité, notre problème peut se réduire à un problème d'optimisation de la forme :

$$\begin{cases} inf_x\, f(x) avec\, f(x)\, notre\, fonction\, \text{à}\, minimiser \\ c_e(x) = 0\, avec\, c_e\, les\, contraintes\, d'\text{égalité} \\ c_i(x) \leq 0\, avec\, c_i\, les\, contraintes\, d'\text{inégalité} \end{cases}$$

Commençons d'abord par détailler ces différentes équations :

- La fonction f(x) à minimiser est la suivante : $f(x) = -U_{bateau}\cos(\beta - \psi)$, c'est la vitesse du navire en prenant en compte la dérive, projetée sur l'axe du cap du navire. Autrement dit, on cherche à maximiser la vitesse du navire suivant le cap de de celui-ci.

- Ensuite il y a plusieurs $c_e(x)$ qui entre en jeu. Dans un premier temps, il y a $\sum \vec{F} = \vec{0}$, ce qui revient à deux équations car c'est une égalité vectorielle. Donc on a deux conditions d'égalité à respecter (somme des forces nulles suivant les axes x et y). Ensuite, il y a aussi $\sum \vec{M} = \vec{0}$, mais comme il y a uniquement le moment du gouvernail, cela revient à dire que $M_{gouv}=0$. Nous n'avons donc pas implémenter cette condition dans notre programme car nous n'en avions pas besoin.

- Enfin, les contraintes d'inégalité $c_i(x)$ correspondent quant à elle à des contraintes physiques : angles de barres et de dérives compris entre -90° et 90°, vitesse du navire entre 0.1 et 100m/s. Si la vitesse du bateau prend comme valeur 100m/s, on saura que notre calcul n'est pas bon directement. L'autre borne traduit le fait que la vitesse recherchée est une norme. De plus, on a remplacé les conditions d'égalité par des conditions d'inégalité (encadrement entre deux valeurs limites) pour s'assurer une convergence.

<u>Explication sur le fonctionnement de l'algorithme</u>

D'après les schémas [6] et [7], dans un premier temps, on pose le lagrangien l(x,λ) du problème :

$l(x,\lambda) \coloneqq f(x) + \lambda^T c(x) = f(x) + \sum_{i=1}^{m} \lambda_i c_i(x)$ où le vecteur λ porte le nom de multiplicateur ou variable duale. L'algorithme est dit primal-dual car il génère une suite de couples (x_k, λ_k) où x_k approche une solution x_{opt} du problème et λ_k approche un multiplicateur optimal λ_{opt}.

On cherche alors à résoudre les conditions d'optimalité de KKT :

$$\begin{cases} \nabla f(x_{opt}) + c'(x_{opt})^* \lambda_{opt} = 0 \\ C_E(x_{opt}) = 0 \\ 0 \leq (\lambda_{opt})_I \perp C_I(x_{opt}) \leq 0 \end{cases}$$

La dernière ligne signifie que :

$0 \leq (\lambda_{opt})_I$: $positivité\ des\ multiplicateurs\ optimaux\ associés\ aux\ contraintes\ d'inégalité$

$C_I(x_{opt}) \leq 0$: $satisfaction\ des\ contraintes\ d'inégalité$

$$(\lambda_{opt})_I \perp C_I(x_{opt}) : complémentarité$$

Ensuite, à l'aide de l'algorithme de Josephy-Newton, qui permet de trouver le zéro d'une somme de fonctions, on peut résoudre ces équations en passant d'un itéré à un autre, jusqu'à obtenir une solution qui converge.

Pour passer d'un itéré à un autre, on résout le système d'équation suivant :

$$\begin{cases} inf_d(\langle \nabla f(x_k), d\rangle + \frac{1}{2}\langle L_k d, d\rangle) \\ c_E(x_k) + c'_E(x_k)d_k = 0 \\ c_I(x_k) + c'_I(x_k)d_k \leq 0 \end{cases} \text{ avec } \begin{cases} L_k : la\ hessienne\ \nabla^2_{xx} l(x_k, \lambda_k) \\ d \in \mathbb{R}^m \end{cases}$$

On pose alors $\begin{cases} x_{k+1} = x_k + d_k \\ \lambda_{k+1} = \lambda_k^{PQ} \end{cases}$.

Il faut bien comprendre que ce n'est pas nous qui avons fait ces calculs, nous avons uniquement repris la fonction minimize de la bibliothèque python scipy.optimize. Cette fonction possède en entrée :

- La fonction à minimiser
- Un premier paramètre qui permet d'initialiser la recherche
- Un tuple dans lequel sont retranscris toutes les consignes
- La méthode d'optimisation choisie (ici « SQSLP », c'est-à-dire la méthode d'optimisation quadratique expliquée précédemment)
- Une série d'option possible à rajouter : le nombre maximum d'itérations, dire s'il y a eu convergence ou non, etc…

Au cours de nos recherches, nous avons fait de nombreux essais, qui nous ont permis d'aboutir à des résultats finaux concluants.

Premier essai : avec les dérivées

Parmi les options possibles à ajouter pour le bon fonctionnement de la fonction minimize, on pouvait ajouter les dérivés des contraintes. En effet, étant donné que le programme se base sur la méthode de Josephy-Newton, proche de la méthode de Newton pour trouver le zéro d'une fonction, rentrer les dérivées dans la fonction était supposé améliorer les résultats. Nous avons rencontré différents problèmes à ce niveau-là :

- Le calcul des dérivées était très long et fastidieux, en effet, il fallait dériver toutes les forces en fonction de toutes les variables. Il y avait alors de nombreux risques d'erreurs.
- Une fois les dérivées calculées, on s'est aperçu en lançant notre programme que celui-ci nous donnait des résultats incohérents, en prenant plus de temps de calcul.

➔Il y avait soit des erreurs dans nos dérivées, soit des erreurs dans notre programme initial (ce qu'on avait déjà observé en lançant le programme sans les dérivés), ou les deux. Nous avons alors choisi de travailler sans les dérivées.

Deuxième essai : sans le gouvernail

Comme nous avions observé qu'il y avait un problème dans le fonctionnement de notre programme, nous avons essayé de déterminer d'où il venait en représentant différentes grandeurs entrant en jeu (les forces, les angles, les vitesses, etc…). Nous nous sommes aperçus que le problème venait de la force du gouvernail, et nous avons vérifié qu'en supprimant cette force dans nos calculs (le problème n'était plus physique), nos calculs donnaient des résultats plus cohérents. Nous avons alors cherché où était l'erreur dans l'expression de notre force qui était bien trop importante. Après s'être renseigné plus en profondeur sur les efforts exercés sur le gouvernail, nous avons vu qu'il y avait un facteur sin(δ) qu'il fallait ajouter devant l'expression de la force.

Troisième essai : variation des paramètres initiaux

Dans les paramètres de la fonction minimize, il y avait aussi le paramètre permettant d'initialiser le calcul qui était important, nous savions qu'il fallait être proche de la solution optimale. Pour cela, nous avons décidé de lancer plusieurs simulations dans les mêmes conditions, avec seulement la « vitesse initiale » qui varie. Cela nous a permis d'observer l'importance de ce paramètre tout en relevant la meilleure valeur optimale, car nous observions des paliers lors de la convergence du calcul. De plus, on a appelé ce paramètre « vitesse initiale », mais cela n'a rien de physique, c'est juste une valeur donnée pour initialiser le calcul et non la vraie vitesse initiale du navire.

Résultats obtenus

On fixe les paramètres suivants pour nos simulations :

```
M = 2000                # Masse du navire (en kg)
g = 9.81                # Accélération de la pesanteur (en m/s**2)
Cx = 0.01;              # Coefficient de traînée du kite fixé
Skite = 10;             # Surface du kite prise par le vent (en m**2)
rho = 1;                # Masse volumique de l'air (en kg.m**(-3))
rhoeau = 1000;          # Masse volumique de l'eau (en kg.m**(-3))
L = 6.5;                # Longueur du bateau (en m)
psi = np.pi-5*np.pi/180 # Angle de cap (en radians)
vitessevent = 10        # Vitesse du vent à l'altitude du kite (en m/s)
Sbateau = 0.5           # Surface du bateau entrant dans le calcul de la loi ITTC 75 (en m**2)
Sderive = 1             # Surface de la dérive (en m**2)
Sgouv   = 0.5           # Surface du gouvernail (en m**2)
```

Figure 18 : *Paramètres utilisés pour notre simulation (fichier paremetres.py)*

Pour la polaire, nous changeons l'angle de cap à chaque fois.

Nous avons commencé à chaque fois par nous assurer que nos calculs pouvaient trouver des solutions. Pour cela, on trace à une vitesse donnée (non atteignable forcément physiquement mais de l'ordre de grandeur de ce que l'on peut trouver typiquement Uvent) un graphique en 2D avec β et δ acceptables de la somme des forces suivant x pour voir si celle-ci s'annule à un endroit. Voici ce que l'on obtient typiquement pour un cap de 175° :

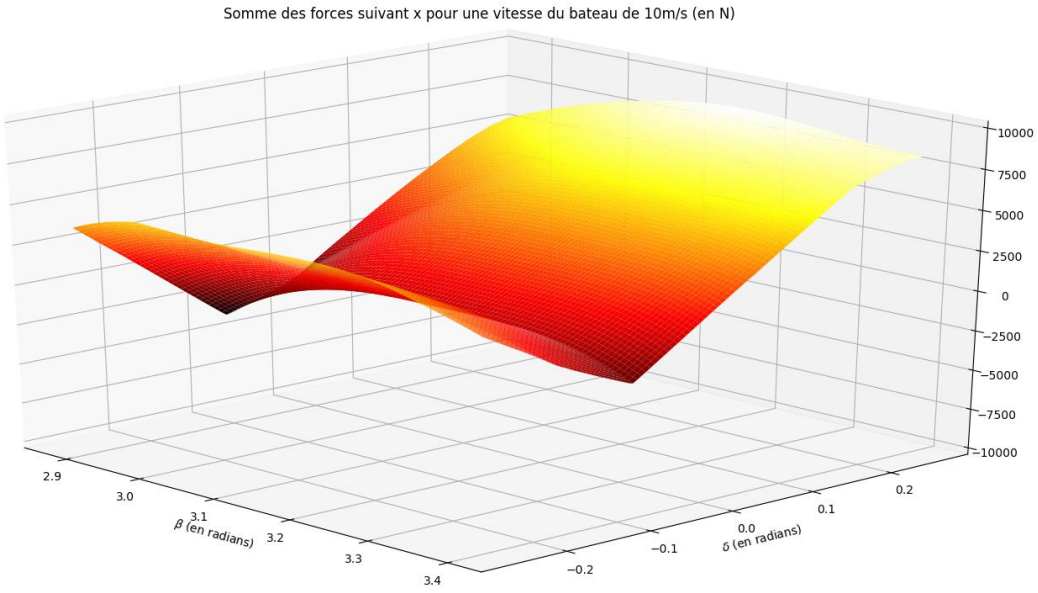


Figure 19 : *Somme des forces suivant X avec Ubateau=10m/s (en N)*

On voit grossièrement que l'on a des solutions en corrélant avec le même graphique des forces suivant Y (on croise 0, il ne faut pas oublier que la vitesse bouge aussi car pour une seule vitesse on a que 2 droites solutions pour chaque somme de forces !).

Pour la méthode minimize, nous avons besoin de rentrer un vecteur de coordonnées « initiales ». Nous pensions au début que nos résultats dépendaient grandement de ces valeurs initiales rentrées et notamment de la valeur de Ubateau, nous avons donc bouclé sur sa valeur. Les autres angles sont fixés au cap pour β et à 0 pour δ, qui sont des valeurs choisies car elles sont assez proches de ce que l'on va trouver : cela augmente grandement la vitesse de l'optimisation. De même on choisit des valeurs proches de la vitesse du vent et de celle qu'on doit trouver. On obtient pour Ubateau la courbe suivante :

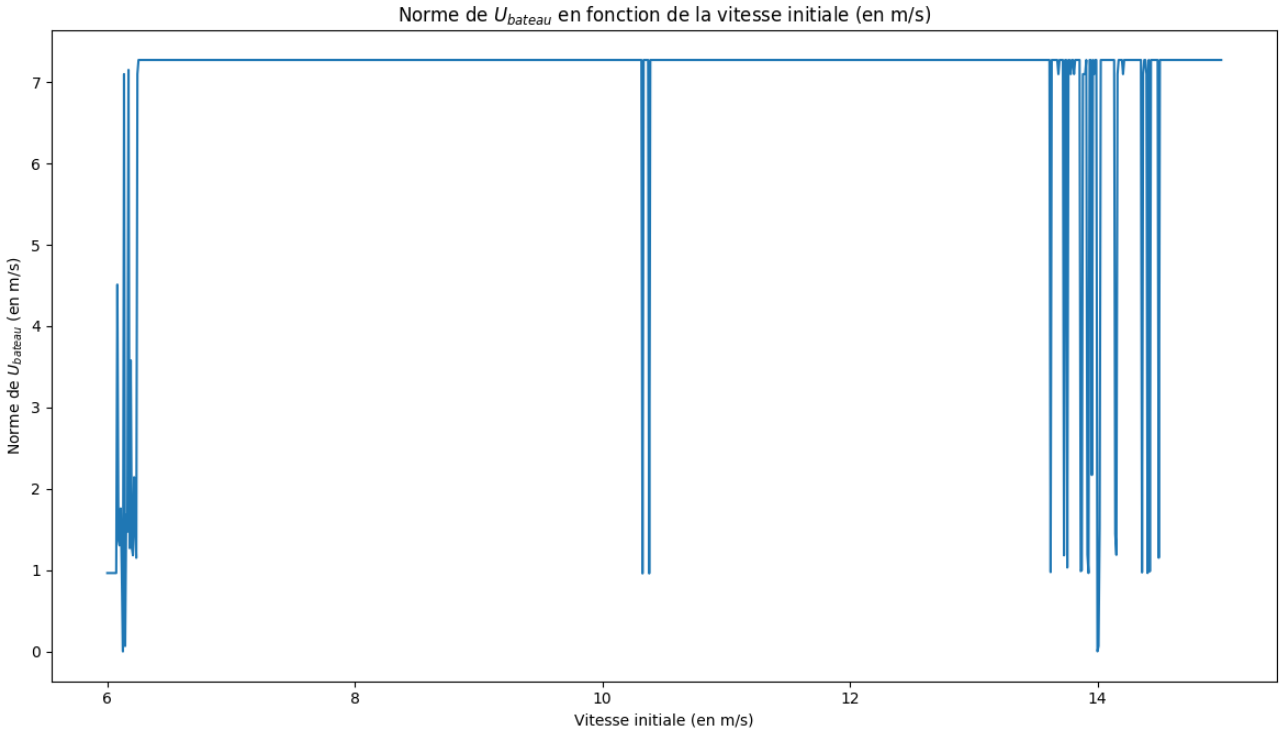
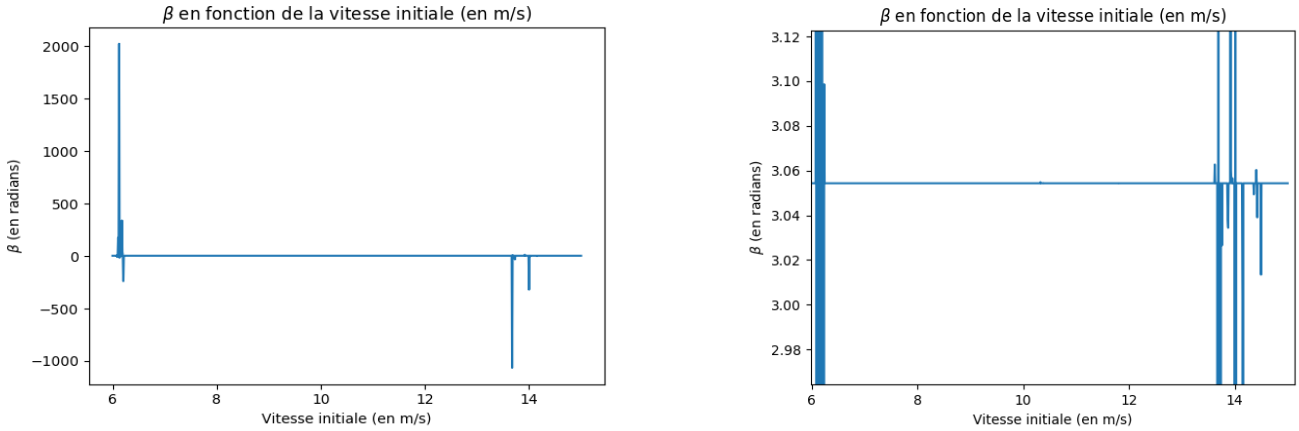


Figure 20 : *Norme de Ubateau optimisée en fonction de la vitesse initiale pour un cap de 175°(en m/s)*

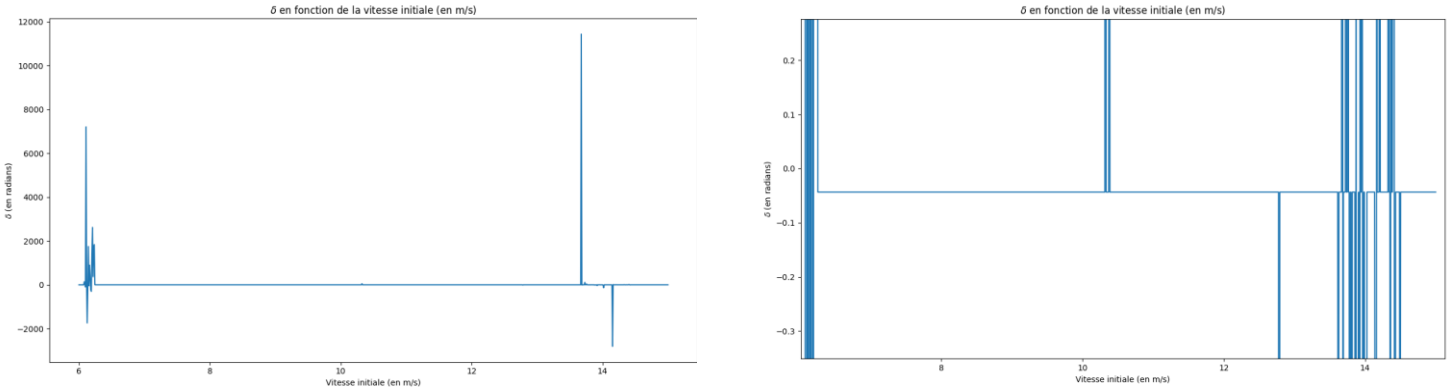
On se doit à chaque fois de vérifier que les sommes des forces sont dans nos intervalles de tolérance (0 +/- 10^{-2}N soit environ 0,01% de l'ordre de grandeur de nos forces ce qui est largement acceptable) pour que notre solution soit physiquement possible.

On affiche beaucoup de valeurs (N = 1000) afin d'afficher clairement un palier qui semble correspondre à une valeur attendue. En effet, certaines valeurs de vitesse initiale font converger le calcul à d'autres vitesses. Nous n'avons cependant pas pu les interpréter physiquement car nous ne savions pas à quoi elles correspondaient. Nous aurions pu améliorer les graphiques en supprimant des valeurs en-dehors de l'écart-type mais nous estimions intéressants de noter la présence d'autres valeurs. De plus, relever la valeur des paliers est assez facile en regardant dans nos tableaux affichés (nous avons même une très bonne précision). Nous avons affiché toutes les valeurs des forces et des angles afin de débeuger le programme quand celui-ci ne fonctionnait pas.

On affiche également β et δ optimisés en fonction de la vitesse initiale :



Figures 21&22 **:** *β optimisé en fonction de la vitesse initiale pour un cap de 175°(en m/s) (Zoomé à droite)*



Figures 23&24 **:** *δ optimisé en fonction de la vitesse initiale pour un cap de 175°(en m/s) (Zoomé à droite)*

Là aussi on note la présence du même palier ce qui est plutôt logique : on prend garde à bien relever des valeurs pour des simulations similaires et à ne pas prendre des valeurs correspondant à la même vitesse initiale.

Nous réalisons nos simulations pour des angles de cap allant de 120° à 240° (la convergence n'est plus assurée après ces valeurs même si le bateau devrait avancer de 90° à 270°). On relève à chaque fois les valeurs des paliers et on le répertorie dans le fichier PolaireOpti.txt (en annexe) ce qui nous permet de tracer polaire suivante :

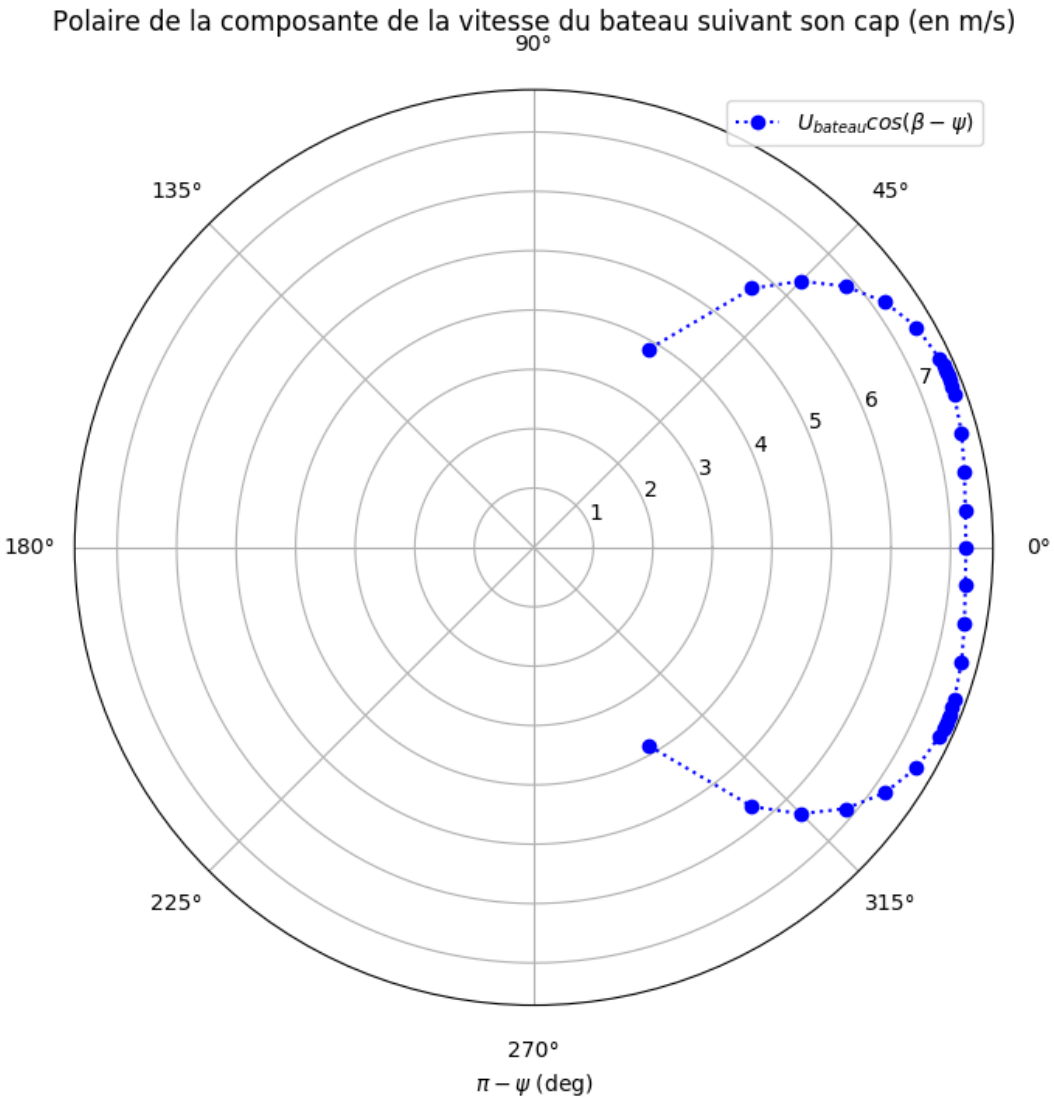


Figure 25 : *Polaire de la composante de la vitesse du bateau suivant son cap (en m/s)*

Nous voyons que pour 2 angles symétriques on obtient une vitesse optimale (+/- 23°). Mr Labat nous avait annoncé ce résultat : on va plus vite en se décalant un tout petit peu de la direction du vent réel avec un kite. Cependant notre voile de kite tout comme notre force de frottement sont modélisées beaucoup trop simplement et mêmes si les maxima peuvent avoir du sens, nous devons être critiques sur les valeurs en norme des vitesses obtenues. En revanche optimiser avec l'angle d'orientation du kite (une variable d'optimisation en plus) en plus est assez aisé mais nous n'avons pas eu le temps de le faire. Quelqu'un avec ce programme pourrait très rapidement y arriver et avoir des résultats très justes notamment avec une bonne base de données de portance et de trainée de kites.

Nous affichons également les graphiques obtenus pour β et δ :

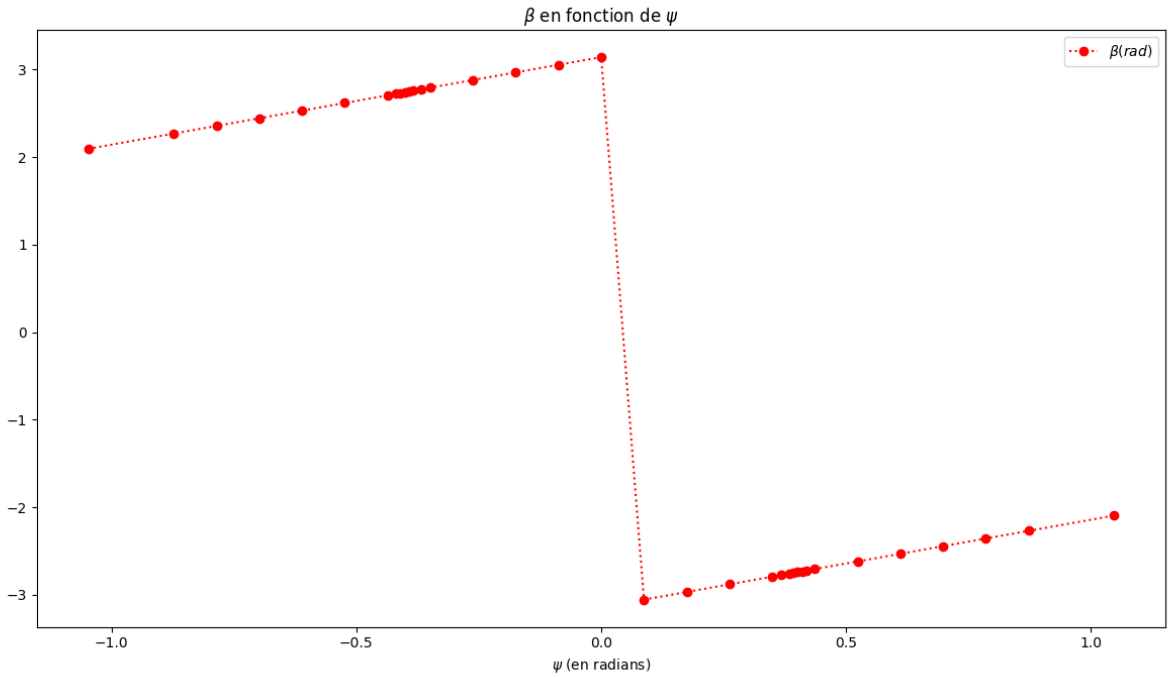


Figure 26 : *β en fonction du cap (en rad)*

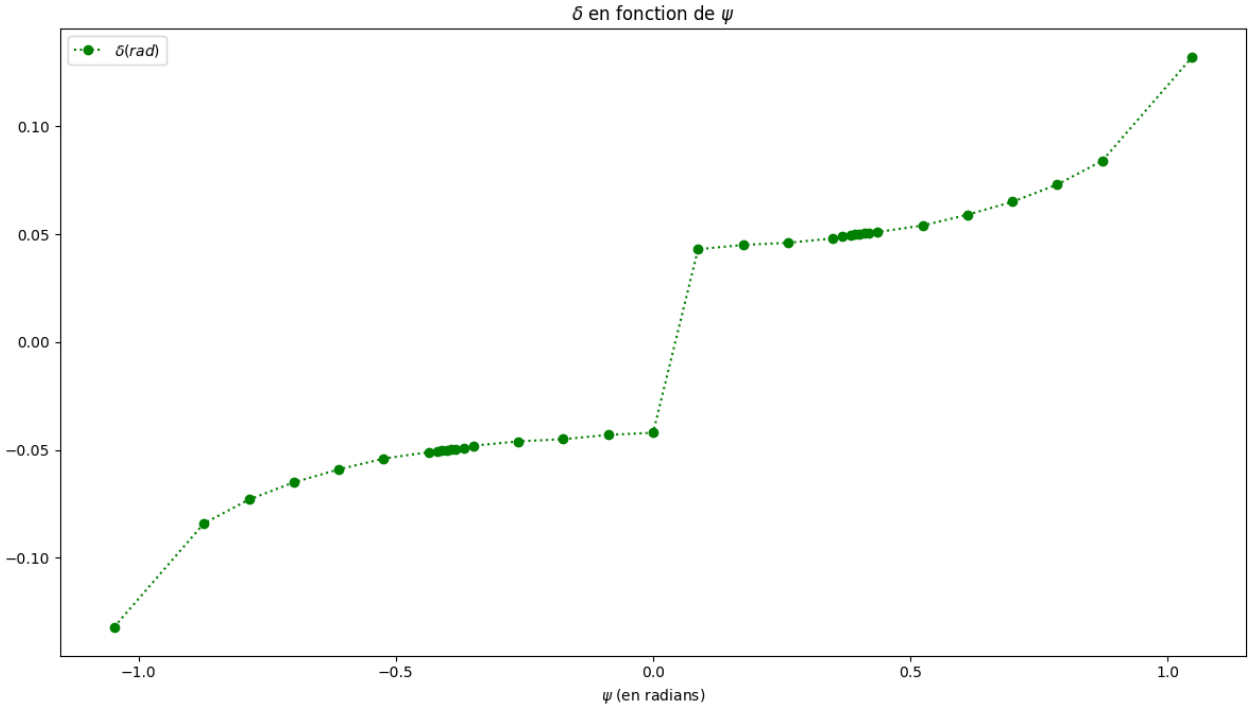


Figure 27 : *δ en fonction du cap (en rad)*

Pour ces des 2 graphiques il s'agit de π – ψ en abscisse.

B est de forme linéaire (congru à π) ce qui n'est pas étonnant vu qu'on a pas pris en compte d'angle d'inclinaison du kite. En revanche pour δ, nous avons un offset de 0.04rad (quand nous changeons de formule à 0°) que nous n'arrivons pas à expliquer. Cela pourrait peut-être provenir de l'extrapolation réalisée par Qblade même si celle-ci est réalisée pour des angles plus élevés.

Conclusion

Notre projet s'est articulé en deux grandes parties. Dans un premier temps, nous nous sommes attelés à un problème de dynamique très compliqué, qui consistait à prendre en compte les inerties et les masses du navire. Ce problème était bien initialisé quand il n'y avait que la force du kite et la force de frottement, mais nous nous sommes aperçus que la méthode était instable lorsqu'on a pris en compte la dérive et le gouvernail. Cette première approche nous aura tout de même permis d'appréhender le problème, de bien comprendre quelles sont les forces qui entraient en jeu, et quels allaient être les problèmes auxquels nous pouvions être confrontés.

Après en avoir discuté avec notre responsable projet, il nous a réorienté vers une approche plus statique. Il a alors fallu revoir tout le système de coordonnées, notamment pour les forces, car désormais la direction du vent était fixée dans le repère et il fallait exprimer les forces dans ce nouveau repère. C'est ce qui nous a posé le plus de problème. Une fois que nous avions trouvé la fonction minimize de python, il nous suffisait alors de rentrer les bonnes fonctions pour que le problème soit résolu. En réalité, ce n'était pas aussi simple que ça car on a rencontré plein de problèmes dus aux coefficients, aux sens des forces, aux contraintes à rentrer…

Finalement, après avoir effectué de nombreuses simulations dans différents contextes, nous avons obtenu des résultats cohérents et une polaire du navire pour des angles de cap compris entre -60° et 60°. Toutefois, ce programme n'est pas une fin en soi. Nous n'avons pas eu le temps de coupler ce résultat avec les résultats obtenus lors du précédent projet [8], qui s'intéressait plus particulièrement à l'optimisation des paramètres propres au kite. De plus, dans notre projet, nous nous sommes placés dans des conditions « triviales » car nous ne dirigeons pas du tout la direction du kite (angle ε sur le schéma), mais le kite est toujours aligné avec le vent apparent. Enfin, nous n'avons pas eu le temps d'optimiser la récupération des valeurs optimales, mais c'était possible en utilisant l'écart type puis la moyenne pour avoir la valeur la plus précise possible, et ce de manière automatique.

Références

[1] « Fiche technique du AK 650 », *Armorkite*
http://www.armorkite.fr/index.php/construction/

[2] « Kite Dynamics for Ship Propulsion », *George Dadd*

[3] « Hydrodynamics of High-Speed Marine Vehicles », *Odd M. Faltinsen*

[4] « Stabilité de route des navires tractés par cerf-volant », *ENSTA Bretagne*

[5] https://www.ensta-bretagne.fr/jaulin/jaulincifa2004.pdf

[6] https://fr.wikipedia.org/wiki/Optimisation_quadratique_successive

[7] https://fr.wikipedia.org/wiki/Algorithme_de_Josephy-Newton

[8] « Récupération de l'énergie éolienne par des ailes de kite »,
Vincent Arnal, Quentin Renaud, Charles Spraul et Mr Baptise Labat

Table des figures

Table des figures (suite)

Annexes

- **Profil d'un NACA0015 (Fichier d'entrée de QBlade) :**

```
NACA 0015
1.0000    0.00158
0.9500    0.01008
0.9000    0.01810
0.8000    0.03279
0.7000    0.04580
0.6000    0.05704
0.5000    0.06617
0.4000    0.07254
0.3000    0.07502
0.2500    0.07427
0.2000    0.07172
0.1500    0.06682
0.1000    0.05853
0.0750    0.05250
0.0500    0.04443
0.0250    0.03268
0.0125    0.02367
0.0000    0.00000
0.0125    -0.02367
0.0250    -0.03268
0.0500    -0.04443
0.0750    -0.05250
0.1000    -0.05853
0.1500    -0.06682
0.2000    -0.07172
0.2500    -0.07427
0.3000    -0.07502
0.4000    -0.07254
0.5000    -0.06617
0.6000    -0.05704
0.7000    -0.04580
0.8000    -0.03279
0.9000    -0.01810
0.9500    -0.01008
1.0000    -0.00158
```

- **Lisezmoi.txt**

Projet de traction d'un navire par kite
#Réalisé par Abel Pruchon, Jiakan Zhou et Jean Cresp

Ce programme permet d'afficher une vitesse projetée optimale d'un bateau tracté par un kit en fonction du cap du navire.

La direction du vent fixe notre repère.

Lancement du programme :

1) Ouvrir tous les fichiers Python du Dossier "Code"

2) Jouer sur les différents paramètres du bateau dans le fichier "parametres.py" notamment sur son cap
(Pour des valeurs acceptables ie. [120°;240°] à mettre en radians)
On peut également jouer sur la vitesse du vent et les coefficients du kite.

3) Dans "opti.py", choisir les vitesses initiales pour lesquelles on optimise (Nombre et intervalle)
Les résultats sont meilleurs pour des valeurs proches de la vitesse du vent.

4) Lancer "opti.py" qui lance automatiquement "extractionccoefs.py".

5) Le programme affiche automatiquement les courbes d'intérêt. Vérifier que les sommes des forces sont bien dans les intervalles de tolérance pour que la solution soit acceptable.

6) Les solutions optimisées correspondent à des paliers sur les figures 1, 2 et 3.

7) Récupérer les différentes valeurs dans le fichier PolaireOpti.txt puis lancer "tracepolaire.py"
Le programme affiche la polaire d'intérêt et les courbes de beta et delta en fonction du cap.

- **PolaireOpti.txt (Résultats de l'optimisation pour différents caps)**

angle(en deg)	Ubateau	psi(en rad)	delta(en rad)
-60.0	3.8554	2.0944	-0.132
-50.0	5.6980	2.2689	-0.084
-45.0	6.3454	2.3560	-0.073
-40.0	6.8446	2.4435	-0.065
-35.0	7.2020	2.5307	-0.059
-30.0	7.4236	2.6180	-0.054
-25.0	7.5206	2.7050	-0.051
-24.0	7.5266	2.7227	-0.0506
-23.5	7.5281	2.7314	-0.0503
-23.0	7.5287	2.7402	-0.0501
-22.5	7.5283	2.7489	-0.0498
-22.0	7.5270	2.7576	-0.0495
-21.0	7.5219	2.7750	-0.049
-20.0	7.5138	2.7925	-0.048
-15.0	7.4393	2.8797	-0.046
-10.0	7.3445	2.9670	-0.045
-5.0	7.2748	3.0543	-0.043
0.0	7.2588	3.1416	-0.042
5.0	7.2748	-3.0543	0.043
10.0	7.3445	-2.9670	0.045
15.0	7.4393	-2.8797	0.046
20.0	7.5138	-2.7925	0.048
21.0	7.5219	-2.7750	0.049
22.0	7.5270	-2.7576	0.0495
22.5	7.5283	-2.7489	0.0498
23.0	7.5287	-2.7402	0.0501
23.5	7.5281	-2.7314	0.0503
24.0	7.5266	-2.7227	0.0506
25.0	7.5206	-2.7050	0.051
30.0	7.4236	-2.6180	0.054
35.0	7.2020	-2.5307	0.059
40.0	6.8446	-2.4435	0.065
45.0	6.3454	-2.3560	0.073
50.0	5.6980	-2.2689	0.084
60.0	3.8554	-2.0944	0.132

- **Fichiers Python**

parametres.py

```
# -*- coding: utf-8 -*-
"""
On fait varier les différents paramètres du problème
Caractéristiques du bateau, du vent, etc...
"""


# Projet de traction d'un navire par kite
# Réalisé par Abel Pruchon, Jiakan Zhou et Jean Cresp

# Définition des différents paramètres du problème

import numpy as np

M = 2000              # Masse du navire (en kg)
g = 9.81              # Accélération de la pesanteur (en m/s**2)
Cx = 0.01;            # Coefficient de traînée du kite fixé
Skite = 10;           # Surface du kite prise par le vent (en m**2)
rho = 1;              # Masse volumique de l'air (en kg.m**(-3))
rhoeau = 1000;        # Masse volumique de l'eau (en kg.m**(-3))
L = 6.5;              # Longueur du bateau (en m)
psi = np.pi-5*np.pi/180 # Angle de cap (en radians)
vitessevent = 10      # Vitesse du vent à l'altitude du kite (en m/s)
Sbateau = 0.5         # Surface du bateau entrant dans le calcul de la loi ITTC 75 (en m**2)
Sderive = 1           # Surface de la dérive (en m**2)
Sgouv   = 0.5         # Surface du gouvernail (en m**2)
```

Extractionsccoefs.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar  8 15:23:03 2018

@author: apruchon2016
"""


# Projet de traction d'un navire par kite
# Réalisé par Abel Pruchon, Jiakan Zhou et Jean Cresp

# Récupération des coefficients de trainee et de portance

#Ce programme récupère des Cl=f(alpha) et Cd=g(alpha) extrapolés sur [-pi;pi] à partir de fichiers d'entrée

import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt


#On extrait les coefficients du gouvernail et de la dérive
filename       = 'Coefs/NACA_0015_T1_Re65.000.txt'
coefshydro     = np.genfromtxt(filename, dtype=float,     skip_header=19)


AoAhydro_index, CLhydro_index, CDhydro_index  = range(3)
coefshydro[:, AoAhydro_index] = coefshydro[:, AoAhydro_index]*np.pi/180 # On convertit les angles en radians
tckCLderive = interpolate.splrep(coefshydro[:, AoAhydro_index], coefshydro[:, CLhydro_index]) # Meme notation que dans la doc scipy
tckCDderive = interpolate.splrep(coefshydro[:, AoAhydro_index], coefshydro[:, CDhydro_index]) # Meme notation que dans la doc scipy


#Affichage des coefficients obtenues
```

```
plt.figure(29)
plt.plot(coefshydro[:, AoAhydro_index],coefshydro[:, CLhydro_index], "b", label = "Coefficient de portance")
plt.xlabel("Angle d'attaque (en radians)")

plt.plot(coefshydro[:, AoAhydro_index],coefshydro[:, CDhydro_index], "r", label = "Coefficient de traînée")
plt.title("Coefficients du NACA0015 en fonction de l'angle d'attaque (Re=6.5e7)")
plt.xlabel("Angle d'attaque (en radians)")

plt.legend()
```

Opti.py (Programme principal)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Mar  6 14:03:38 2018

@author: apruchon2016
"""

# Projet de traction d'un navire par kite
# Réalisé par Abel Pruchon, Jiakan Zhou et Jean Cresp

# Optimisation de la vitesse projetée

from scipy.optimize import minimize
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate
import parametres
import extractionccoefs as extrac
from mpl_toolkits.mplot3d import Axes3D

# Définition des différentes variables du problème
'''
Beta    = angle du bateau
delta   = angle de barre
psi     = cap
dirvent = angle du vent réel fixé à 0
gamma   = angle du vent apparent
'''

# Calcul du nombre de Reynolds
def Reynolds(u, rho, L):
    visc = 1.07*10**(-3)
    Re   = np.abs(u)*rho*L/visc
    return Re


# Calcul du coefficient de frottement du bateau selon une loi empirique
def Cfbateau(Re):
    #print(Re)
    if Re == 0:
        return 0
    else:
        # Loi ITTC 57
        Cf = 0.075/(np.log10(Re)-2)**2
        return Cf

# Détermination des frottements sur la coque du navire
def Frottements(Ubateau, rho, L, u):
    Re =    Reynolds(Ubateau, rho, L)
    Cf =    Cfbateau(Re);
    frot = -0.5*Cf*rho*parametres.Sbateau*u**2
    return frot

# Calcul de la norme et de la direction du vent apparent
def Ventapparent(uvent, Ubateau, beta):
    """
```

```
    Uvent:   vitesse réelle du vent
    Ubateau: vitesse du bateau
    beta:    angle de la vitesse du bateau
    """

    Uapp =      np.hypot(Ubateau*np.sin(beta), uvent + Ubateau*np.cos(beta))

    #On calcule le cos et le sin de l'angle afin d'être sur de sa valeur
    cosuappdir=-(uvent+Ubateau*np.cos(beta))/Uapp
    sinuappdir=-Ubateau*np.sin(beta)/Uapp

    return (Uapp, cosuappdir, sinuappdir)


#Pour un angle quelconque en radians, cette fonction renvoie sa valeur entre -pi et pi
def moduloangle(alpha):
    alpha = alpha%(2*np.pi)
    if alpha > np.pi :
        alpha = ((alpha+np.pi)%(2*np.pi)) -np.pi
    return alpha


# Somme des forces suivant l'axe x

def sumFx(D):  # D = (Ubateau, beta, delta)

    # Interpolation des coefficients
    CLder =     interpolate.splev(moduloangle(D[1]-parametres.psi), extrac.tckCLderive, der=0)
    CDder =     interpolate.splev(moduloangle(D[1]-parametres.psi), extrac.tckCDderive, der=0)
    CLgouv =    interpolate.splev(moduloangle(D[1]-parametres.psi-D[2]), extrac.tckCLderive, der=0)
    CDgouv =    interpolate.splev(moduloangle(D[1]-parametres.psi-D[2]), extrac.tckCDderive, der=0)

    # Définition des forces
    Fderive_x = 0.5*parametres.rhoeau*parametres.Sderive*D[0]**2*np.sqrt(CLder**2+CDder**2)*(np.sin(parametres.psi))

    Fgouv_x =   0.5*parametres.rhoeau*parametres.Sgouv*D[0]**2*np.sqrt(CLgouv**2+CDgouv**2)*np.sin(D[2])*(np.sin(parametres.psi+D[2]))

    [Uapp, cosuappdir, sinuappdir] = Ventapparent(parametres.vitessevent,D[0],D[1])
    Fkite_x =   -0.5*parametres.rho*parametres.Skite*Uapp**2*cosuappdir

    Ffrott_x =  Frottements(D[0], parametres.rhoeau, parametres.L, D[0])*np.cos(D[1])**2

    return      Fderive_x + Fgouv_x + Ffrott_x + Fkite_x


# Somme des forces suivant l'axe y
def sumFy(D):  # D = (Ubateau, beta, delta)

    # Interpolation des coefficients
    CLder =     interpolate.splev(moduloangle(D[1]-parametres.psi), extrac.tckCLderive, der=0)
    CDder =     interpolate.splev(moduloangle(D[1]-parametres.psi), extrac.tckCDderive, der=0)
    CLgouv =    interpolate.splev(moduloangle(D[1]-parametres.psi-D[2]), extrac.tckCLderive, der=0)
    CDgouv =    interpolate.splev(moduloangle(D[1]-parametres.psi-D[2]), extrac.tckCDderive, der=0)


    # Definition des forces
    Fderive_y = 0.5*parametres.rhoeau*parametres.Sderive*D[0]**2*np.sqrt(CLder**2+CDder**2)*(-np.cos(parametres.psi))

    Fgouv_y =   0.5*parametres.rhoeau*parametres.Sgouv*D[0]**2*np.sqrt(CLgouv**2+CDgouv**2)*np.sin(D[2])*(-np.cos(parametres.psi+D[2]))

    [Uapp, cosuappdir, sinuappdir] = Ventapparent(parametres.vitessevent,D[0],D[1])
    Fkite_y =   -0.5*parametres.rho*parametres.Skite*Uapp**2*sinuappdir

    Ffrott_y =  Frottements(D[0], parametres.rhoeau, parametres.L, D[0])*np.sin(D[1])**2

    return      Fderive_y + Fgouv_y + Ffrott_y + Fkite_y



# Gradient de la somme des forces suivant l'axe x
# Sert dans la fonction d'optimisation commecritère de qualité
```

```
def sumdFx(D):# D= (Ubateau, beta, delta)
    [Uapp, cosuappdir, sinuappdir] = Ventapparent(parametres.vitessevent,D[0],D[1])
    # Interpolation des coefficients
    CLder =     interpolate.splev(moduloangle(D[1]-parametres.psi), extrac.tckCLderive, der=0)
    CDder =     interpolate.splev(moduloangle(D[1]-parametres.psi), extrac.tckCDderive, der=0)
    CLgouv =    interpolate.splev(moduloangle(D[1]-parametres.psi-D[2]), extrac.tckCLderive, der=0)
    CDgouv =    interpolate.splev(moduloangle(D[1]-parametres.psi-D[2]), extrac.tckCDderive, der=0)
    dCLder =    interpolate.splev(moduloangle(D[1]-parametres.psi), extrac.tckCLderive, der=1)
    dCDder =    interpolate.splev(moduloangle(D[1]-parametres.psi), extrac.tckCDderive, der=1)
    dCLgouv =   interpolate.splev(moduloangle(D[1]-parametres.psi-D[2]), extrac.tckCLderive, der=1)
    dCDgouv =   interpolate.splev(moduloangle(D[1]-parametres.psi-D[2]), extrac.tckCDderive, der=1)

    # Définition des dérivées partielles des forces
    d0Fderive_x = parametres.rhoeau*parametres.Sderive*D[0]*np.sqrt(CLder**2+CDder**2)*(np.sin(parametres.psi))
    d0Fgouv_x =   parametres.rhoeau*parametres.Sgouv*D[0]*np.sqrt(CLgouv**2+CDgouv**2)*(np.sin(parametres.psi+D[2]))

    d0Uapp =      (parametres.vitessevent*np.cos(D[1])+D[0])/Uapp
    d0diruapp=    -(parametres.vitessevent*np.sin(D[1]))/(Uapp**2)

    d0Fkite_x =   (d0Uapp*parametres.rho*parametres.Skite*Uapp*cosuappdir
                -0.5*d0diruapp*parametres.rho*parametres.Skite*Uapp**2*sinuappdir)
    d0Ffrott_x                                  =                                                                   (0.5*parametres.rhoeau*parametres.Sbateau)*((-
0.075*np.cos(D[1])**2*D[0]*(2*np.log10(Reynolds(D[0],parametres.rhoeau,parametres.L))**2-
8*np.log10(Reynolds(D[0],parametres.rhoeau,parametres.L))
                +                   8                   -                   np.log10(Reynolds(D[0],parametres.rhoeau,parametres.L))/np.log(10)                  +
4/np.log(10))/(np.log10(Reynolds(D[0],parametres.rhoeau,parametres.L))-2)**4))

    d1Fderive_x                             =                           0.5*parametres.rhoeau*parametres.Sderive*D[0]**2*((dCLder*CLder                  +
dCDder*CDder)/np.sqrt(CLder**2+CDder**2))*(np.sin(parametres.psi))
    d1Fgouv_x                   =                                       0.5*parametres.rhoeau*parametres.Sgouv*D[0]**2*((dCLgouv*CLgouv                  +
dCDgouv*CDgouv)/np.sqrt(CLgouv**2+CDgouv**2))*(np.sin(parametres.psi+D[2]))

    d1Uapp =      -D[0]*parametres.vitessevent*np.sin(D[1])/Uapp
    d1diruapp =   -(D[0]*parametres.vitessevent*np.cos(D[1])+D[0]**2)/Uapp**2

    d1Fkite_x =   (d1Uapp*parametres.rho*parametres.Skite*Uapp*cosuappdir
                -0.5*d1diruapp*parametres.rho*parametres.Skite*Uapp**2*sinuappdir)

    d1Ffrott_x =  Frottements(D[0], parametres.rhoeau, parametres.L, D[0])*(-2*np.cos(D[1]*np.sin(D[1])))


    d2Fderive_x = 0
    d2Fgouv_x                   =                                       (0.5*parametres.rhoeau*parametres.Sgouv*D[0]**2*(-(dCLgouv*CLgouv                  +
dCDgouv*CDgouv)/np.sqrt(CLgouv**2+CDgouv**2))*(np.sin(parametres.psi+D[2]))
                + 0.5*parametres.rhoeau*parametres.Sgouv*D[0]**2*np.sqrt(CLgouv**2+CDgouv**2)*(np.cos(parametres.psi+D[2])))
    d2Fkite_x =   0
    d2Ffrott_x =  0


    return np.array([d0Fderive_x + d0Fgouv_x + d0Fkite_x + d0Ffrott_x,
                     d1Fderive_x + d1Fgouv_x + d1Fkite_x + d1Ffrott_x,
                     d2Fderive_x + d2Fgouv_x + d2Fkite_x + d2Ffrott_x])



# Gradient de la somme des forces suivant l'axe x
# Sert dans la fonction d'optimisation commecritère de qualité

def sumdFy(D):  # D = (Ubateau, beta, delta)

    [Uapp, cosuappdir, sinuappdir] = Ventapparent(parametres.vitessevent,D[0],D[1])

    #Interpolation des coefficients
    CLder =     interpolate.splev(moduloangle(D[1]-parametres.psi), extrac.tckCLderive, der=0)
    CDder =     interpolate.splev(moduloangle(D[1]-parametres.psi), extrac.tckCDderive, der=0)
    CLgouv =    interpolate.splev(moduloangle(D[1]-parametres.psi-D[2]), extrac.tckCLderive, der=0)
    CDgouv =    interpolate.splev(moduloangle(D[1]-parametres.psi-D[2]), extrac.tckCDderive, der=0)
    dCLder =    interpolate.splev(moduloangle(D[1]-parametres.psi), extrac.tckCLderive, der=1)
    dCDder =    interpolate.splev(moduloangle(D[1]-parametres.psi), extrac.tckCDderive, der=1)
    dCLgouv =   interpolate.splev(moduloangle(D[1]-parametres.psi-D[2]), extrac.tckCLderive, der=1)
```

```
    dCDgouv =   interpolate.splev(moduloangle(D[1]-parametres.psi-D[2]), extrac.tckCDderive, der=1)

    #Definition des dérivées partielles des forces
    d0Fderive_y = parametres.rhoeau*parametres.Sderive*D[0]*np.sqrt(CLder**2+CDder**2)*(-np.cos(parametres.psi))
    d0Fgouv_y =   parametres.rhoeau*parametres.Sgouv*D[0]*np.sqrt(CLgouv**2+CDgouv**2)*(-np.cos(parametres.psi+D[2]))

    d0Uapp =      (parametres.vitessevent*np.cos(D[1])+D[0])/Uapp
    d0diruapp=    -(parametres.vitessevent*np.sin(D[1]))/(Uapp**2)
    d0Fkite_y =   (d0Uapp*parametres.rho*parametres.Skite*Uapp*sinuappdir
                +0.5*d0diruapp*parametres.rho*parametres.Skite*Uapp**2*cosuappdir)

    d0Ffrott_y                                =                                                                                (0.5*parametres.rhoeau*parametres.Sbateau)*((-
0.075*np.sin(D[1])**2*D[0]*(2*np.log10(Reynolds(D[0],parametres.rhoeau,parametres.L))**2-
8*np.log10(Reynolds(D[0],parametres.rhoeau,parametres.L))
                +                    8                    -                    np.log10(Reynolds(D[0],parametres.rhoeau,parametres.L))/np.log(10)                    +
4/np.log(10))/(np.log10(Reynolds(D[0],parametres.rhoeau,parametres.L))-2)**4))

    d1Fderive_y     =     0.5*parametres.rhoeau*parametres.Sderive*D[0]**2*((dCLder*CLder     +     dCDder*CDder)/np.sqrt(CLder**2+CDder**2))*(-
np.cos(parametres.psi))
    d1Fgouv_y =     0.5*parametres.rhoeau*parametres.Sgouv*D[0]**2*((dCLgouv*CLgouv + dCDgouv*CDgouv)/np.sqrt(CLgouv**2+CDgouv**2))*(-
np.cos(parametres.psi+D[2]))

    d1Uapp =      -D[0]*parametres.vitessevent*np.sin(D[1])/Uapp
    d1diruapp =   -(D[0]*parametres.vitessevent*np.cos(D[1])+D[0]**2)/Uapp**2
    d1Fkite_y =   (d1Uapp*parametres.rho*parametres.Skite*Uapp*sinuappdir
                +0.5*d1diruapp*parametres.rho*parametres.Skite*Uapp**2*cosuappdir)

    d1Ffrott_y =  Frottements(D[0], parametres.rhoeau,parametres.L, D[0])*(2*np.cos(D[1])*np.sin(D[1]))


    d2Fderive_y = 0
    d2Fgouv_y =   (0.5*parametres.rhoeau*parametres.Sgouv*D[0]**2*(-(dCLgouv*CLgouv + dCDgouv*CDgouv)/np.sqrt(CLgouv**2+CDgouv**2))*(-
np.cos(parametres.psi+D[2]))
                + 0.5*parametres.rhoeau*parametres.Sgouv*D[0]**2*np.sqrt(CLgouv**2+CDgouv**2)*(np.sin(parametres.psi+D[2])))
    d2Fkite_y =   0
    d2Ffrott_y =  0



    return np.array([d0Fderive_y + d0Fgouv_y + d0Fkite_y + d0Ffrott_y,
                     d1Fderive_y + d1Fgouv_y + d1Fkite_y + d1Ffrott_y,
                     d2Fderive_y + d2Fgouv_y + d2Fkite_y + d2Ffrott_y])



#Définition de la fonction à optimiser
#Correspond à la composante de la vitesse du bateau suivant son cap
def Fopti(D):
    return      -D[0]*np.cos(D[1]-parametres.psi)


#Gradient de la fonction à optimiser
def dFopti(D):
    dfdD0 =     -np.cos(D[1]-parametres.psi)
    dfdD1 =     D[0]*np.sin(D[1]-parametres.psi)
    dfdD2 =     0
    return      np.array([dfdD0, dfdD1, dfdD2])



#AFFICHAGE 3D des valeurs de la somme des forces
#Permet de regarder globalement si on a des solutions pour une vitesse non forcément atteignable physiquement

def affichage3D():

    fig = plt.figure(100)
    ax =  Axes3D(fig)
    V = 10                #Valeur de la vitesse où l'on souhaite voir s'il y a des solutions (en m/s)
                        #Ici on teste égale à la vitesse du vent
```

```
    X =   np.linspace(np.pi-15*np.pi/180, np.pi+15*np.pi/180, 100) #Betas possibles(en radians)
    Y =   np.linspace(-15*np.pi/180, 15*np.pi/180, 100)            #Deltas possibles(en radians)
    Z =   np.zeros((len(X),len(X)))

    for i in range(len(X)):
        for j in range(len(Y)):
            Z[i,j] = sumFx([V,X[i],Y[j]])

    X, Y = np.meshgrid(X, Y)
    ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='hot') #Affichage 3D
    ax.set_zlim(-10000,10000)
    plt.title('Somme des forces suivant x pour une vitesse du bateau de 10m/s (en N)')
    plt.xlabel(r"$\beta$ (en radians)")
    plt.ylabel(r"$\delta$ (en radians)")
    plt.show()

affichage3D()


#Définition des contraintes de l'optimisation
#On accepte une tolérance de 10^(-2)N sur la somme des forces nulle
cons = ({'type':'ineq',
         'fun' :lambda x: -sumFx(x)+10**(-2)},

        {'type':'ineq',
         'fun' :lambda x: sumFx(x)+10**(-2)},

        {'type':'ineq',
         'fun' :lambda x: -sumFy(x)+10**(-2)},

         {'type':'ineq',
         'fun' :lambda x: sumFy(x)+10**(-2)},

        {'type':'ineq',
         'fun' :lambda x: np.array([x[0]-0.0])},

        {'type':'ineq',
         'fun' :lambda x: np.array([-x[0]+100])})



#Affichage des résultats de l'optimisation
#On définit les vecteurs dans lesquels on récupère tous nos résultats
N =1000                    #Nombre de vitesses initiales testées
Vini = np.linspace(6,15,N)  #Intervalle des vitesses initiales
U = np.zeros(N)
beta = np.zeros(N)
delta = np.zeros(N)
Fx=np.zeros(N)
Fy=np.zeros(N)

#On récupère les forces dans les tableaux pour vérifier leurs ordres de grandeur
Fderive_x=np.zeros(N)
Fgouv_x=np.zeros(N)
Fkite_x=np.zeros(N)
Ffrott_x=np.zeros(N)



print("Starting optimization loop")
for i in range(N):
    print(str(i)+"/"+str(N))

    #On réalise l'optimisation pour plusieurs vitesses initiales
    res = minimize(Fopti, np.array([Vini[i],parametres.psi,0.0]),
                constraints = cons, method='SLSQP', options={'maxiter': 10000})

    #Récupération du vecteur optimisé
    U[i]=res.x[0]
    beta[i]=res.x[1]
    delta[i]=res.x[2]
```

```
    #On vérifie la cohérence de toutes les valeurs
    CLder = interpolate.splev(moduloangle(res.x[1]-parametres.psi), extrac.tckCLderive, der=0)
    CDder = interpolate.splev(moduloangle(res.x[1]-parametres.psi), extrac.tckCDderive, der=0)
    CLgouv = interpolate.splev(moduloangle(res.x[1]-parametres.psi-res.x[2]), extrac.tckCLderive, der=0)
    CDgouv = interpolate.splev(moduloangle(res.x[1]-parametres.psi-res.x[2]), extrac.tckCDderive, der=0)
    [Uapp, cosuappdir, sinuappdir] = Ventapparent(parametres.vitessevent,res.x[0],res.x[1])

    Fkite_x[i] = -0.5*parametres.rho*parametres.Skite*Uapp**2*cosuappdir
    Ffrott_x[i] = Frottements(res.x[0], parametres.rhoeau, parametres.L, res.x[0])*np.cos(res.x[1])**2
    Fderive_x[i] = 0.5*parametres.rhoeau*parametres.Sderive*res.x[0]**2*np.sqrt(CLder**2+CDder**2)*(np.sin(parametres.psi))
    Fgouv_x[i] =
0.5*parametres.rhoeau*parametres.Sgouv*res.x[0]**2*np.sqrt(CLgouv**2+CDgouv**2)*np.sin(res.x[2])*(np.sin(parametres.psi+res.x[2]))

    Fx[i]=sumFx([U[i],beta[i],delta[i]])
    Fy[i]=sumFy([U[i],beta[i],delta[i]])


#Avec cette valeur, on voit si notre solution est physiquement acceptable (à corréler avec la somme des forces)
plt.figure(1)
plt.plot(Vini,U)
plt.title('Norme de $U_{bateau}$ en fonction de la vitesse initiale (en m/s)')
plt.xlabel('Vitesse initiale (en m/s)')
plt.ylabel('Norme de $U_{bateau}$ (en m/s)')
plt.show()


plt.figure(2)
plt.plot(Vini,beta)
plt.title(r'$\beta$ en fonction de la vitesse initiale (en m/s)')
plt.xlabel('Vitesse initiale (en m/s)')
plt.ylabel(r'$\beta$ (en radians)')
plt.show()

plt.figure(3)
plt.plot(Vini,delta)
plt.title(r'$\delta$ en fonction de la vitesse initiale (en m/s)')
plt.xlabel('Vitesse initiale (en m/s)')
plt.ylabel(r'$\delta$ (en radians)')
plt.show()

plt.figure(4)
plt.plot(Vini,Fx)
plt.title('Somme des forces suivant X en fonction de la vitesse initiale (en m/s)')
plt.xlabel('Vitesse initiale (en m/s)')
plt.ylabel('Somme des forces suivant X (en N)')
plt.show()

plt.figure(5)
plt.plot(Vini,Fy)
plt.title('Somme des forces suivant Y en fonction de la vitesse initiale (en m/s)')
plt.xlabel('Vitesse initiale (en m/s)')
plt.ylabel('Somme des forces suivant Y (en N)')
plt.show()
```

tracepolaire.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 26 20:17:21 2018

@author: apruchon2016
"""

# Projet de traction d'un navire par kite
# Réalisé par Abel Pruchon, Jiakan Zhou et Jean Cresp

# Tracé de la polaire des vitesses optimisées

import numpy as np
import matplotlib.pyplot as plt


#On récupère les données dans le fichier d'entrée
filename        = 'PolaireOpti.txt'
coefs           = np.genfromtxt(filename, dtype=float, skip_header=2)

#On recalcule la vitesse projetée
V = np.zeros(len(coefs[:,1]))

for i in range (len(coefs[:,1])):
   V[i]= -coefs[i,1]*np.cos(coefs[i,0]*np.pi/180-coefs[i,2])


#On affiche les différentes données
plt.figure(31)
plt.polar(coefs[:,0]*(np.pi/180),V, "b:o", label=r"$U_{bateau}cos(\beta-\psi)$")
plt.title('Polaire de la composante de la vitesse du bateau suivant son cap (en m/s)')
plt.xlabel(r"$\pi-\psi$ (deg)")
plt.legend()

plt.figure(32)
plt.plot(coefs[:,0]*(np.pi/180), coefs[:,2],  "r:o", label = r"$\beta (rad)$")
plt.xlabel(r"$\psi$ (en radians)")
plt.title(r"$\beta$ en fonction de $\psi$")
plt.legend()

plt.figure(33)
plt.plot(coefs[:,0]*(np.pi/180), coefs[:,3],  "g:o", label = r"$\delta (rad)$")
plt.xlabel(r"$\psi$ (en radians)")
plt.title(r"$\delta$ en fonction de $\psi$")
plt.legend()
```