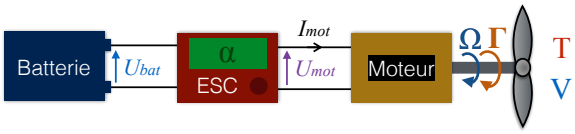
PropOptim v1.0

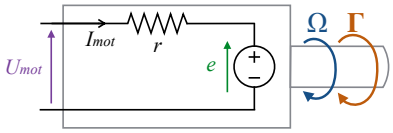
PropOptim est un outil qui optimise la chaine de propulsion (couplage moteur-hélice et batterie) pour des drones en tenant compte des différents paramètres et critères. Il utilise un modèle linéarisé pour avoir un court temps de calcul.

I. Equations

Cette première version utilise des équations simplifiées. Elles sont particulièrement simplifiées pour les batteries et les variateurs, donc, par exemple, les temps de vol calculés ne sont que très approximatifs.



I.1. Moteur



Les trois paramètres principaux d'un moteur sont: La constate de vitesse K_v (qui en unités SI est égale a sa constate de couple K_c) sa résistance interne *r* et son courant à vide I_0.
On peut trouver les relations entre la tension en entrée du moteur U_{mot}, le courant I_{mot} la vitesse de rotation Ω et le couple Γ sur l'arbre en utilisant les équation du moteur et l'équation électrique.

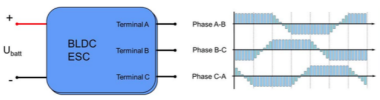
$$\Gamma = \frac{I - I_0}{K_c} \quad (1)$$

$$U_{mot} = rI_{mot} + \frac{\Omega}{K_v} \quad (2)$$

Pour fonctionner dans leur plage de rendement optimal, on doit satisfaire des contraintes: Le courant dois rester dans une plage de courant nominal $[I_{Nmin}, I_{Nmax}]$ et le moteur doit être alimenté par une batterie avec un nombre spécifique d'unités (cellules en série) qui se traduit par une tension nominale.

I.2. ESC

Le *Electronic Speed Controler* ou variateur, est l'unité chargée d'alimenter le moteur à partir de la batterie. Comme les moteurs *brushless* on besoin d'un courant alternative l'ESC que le rôle d'onduleur.



Mais pour pouvoir contrôler la vitesse le ESC joue aussi le rôle de hacheur, en *multipliant* le signal par des impulsions de largeur réglable à une fréquence plus élevé que la fréquence du signal générée. Cela a comme conséquence une valeur moyenne de la tension ressentie par le moteur qui varie proportionnellement au rapport cyclique alpha (largeur de l'impulsion sur la période de hachage). On a donc :

$$U_{mot} = \alpha U_{bat} \quad (3)$$

I.3. Batterie

On n'étudie ici que les batteries Li-Po (Polymère de Lithium). Les paramètres principaux d'une batterie sont : Le nombre de cellules en série N_S (3S, par exemple) qui donne la tension moyenne de la batterie (3.7 V par cellule pour les Li-Po), la capacité de charge la batterie en mAh Q_{bat}, le taux de décharge maximal D_{bat} en nombre de C qui donne un courant maximal au quel la batterie peut décharger en fonction de sa capacité et la masse de la batterie. Dans cette première version on considère la tension de la batterie comme constate pendant la décharge. Les équations simplifiés sont alors:

$$T_{vol} = \frac{Q_{bat}}{I_{bat}} \quad (4)$$

$$I_{max} = D_{bat} \cdot Q_{bat} \quad (5)$$

$$U_{bat} = N_S \cdot 3.7\ V \quad (6)$$

On peut aussi augmenter la capacité de charge en mettant plusieurs batteries du même type en parallèle, ce qui conserve la même tension.

I.4. Hélice

Le point de fonctionnement d'une hélice est déterminée par sa vitesse de rotation Ω et sa vitesse dans l'air V. On peut aussi définir

l'avancement J et les coefficients de poussée C_t et de puissance C_p par:

$$J = \frac{V}{n\,D} \quad (7)$$

$$T = \rho \cdot C_t \cdot n^2 \cdot D^4 \quad (8)$$

$$P = \rho \cdot C_p \cdot n^3 \cdot D^5 \quad (9)$$

Où T est la pousse générée par l'hélice, P est la puissance nécessaire, $n = 2\pi\Omega$, ρ est la masse volumique de l'air, et D est le diamètre de l'hélice.

On base ce programme sur la base de donnés expérimentale UIUC, où plusieurs hélices ont été testés à plusieurs RPM et plusieurs vitesses. On a donc plusieurs tableaux par hélice à différents RPM donnant J, C_t et C_p. On peut alors facilement transformer ces tableaux en $T/\rho(V, RPM)$ et $P/\rho(V, RPM)$. On a aussi accès au couple avec la relation:

$$\Gamma = \frac{P}{\Omega} \quad (10)$$

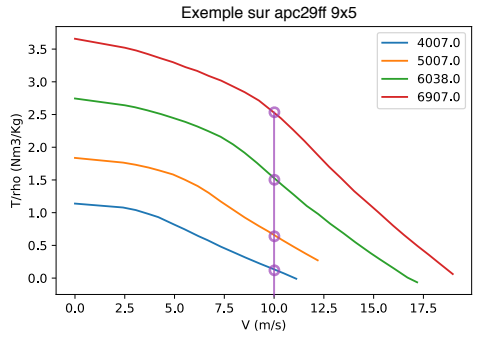
La base de données à aussi un tableau à $V = 0$ par hélice donnant RPM, C_t et C_p. On peut donc prolonger les courbes jusqu'à $V = 0$ en interpolant pour les valeurs les RPM respectives les valeurs en 0 (un polynôme d'ordre 2 pour $T/\rho(V, RPM)$ et d'ordre 3 pour $P/\rho(V, RPM)$)

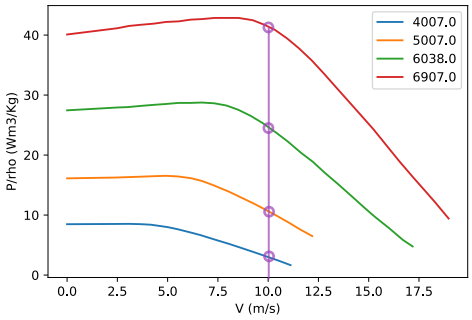
I.4.1 Obtention du point de fonctionnement

Pour obtenir le point de fonctionnement on a besoin de la vitesse air de la poussée nécessaire et la masse volumique.

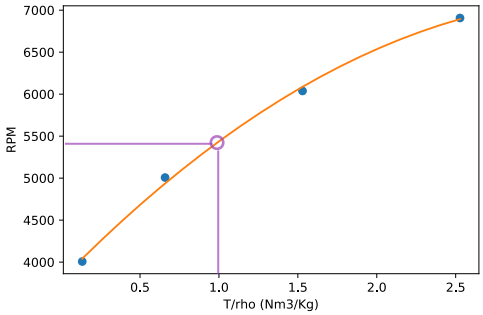
Pour trouver le point de fonctionnement on utilise la méthode employée dans DroneCalc [ref]

On doit d'abord obtenir le la vitesse de rotation. Pour cela on trouve le point sur caque courbe correspondant a la vitesse souhaitée par interpolation.

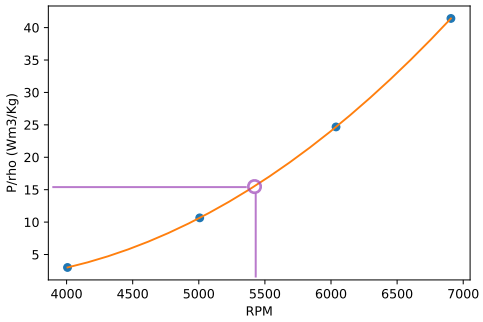




On interpole en suite avec un polynôme d'ordre 2 les point $(RPM, T/\rho)$ pour obtenir la vitesse de rotation correspondant à la poussée



nécessaire.
En suite en interpolant les points $(RPM, P/\rho)$ avec un polynôme d'orde 3 on déduit la puissance nécessaire et donc le couple avec



(10).

II. Optimisation

Pour avoir un programme rapide on utilise la MILP (programmation linéaire mixte avec entiers).

II.1. Modélisation

On note N_m le nombre de moteurs, N_p le nombre d'hélices et N_b le nombre de batteries.

On modélisé le choix de moteur, de batterie et de hélice par de variables binaires (1 si choisi 0 sinon). On a les équations:

$$X_{m\_i} \in \{0,1\} \quad \forall i \in [\![1,N_m]\!] \quad \text{et} \quad \sum_{i=1}^{N_m} X_{m\_i} = 1$$

$$X_{p\_j} \in \{0,1\} \quad \forall j \in [\![1,N_p]\!] \quad \text{et} \quad \sum_{j=1}^{N_p} X_{p\_j} = 1$$

$$X_{b\_k} \in \{0,1\} \quad \forall k \in [\![1,N_b]\!] \quad \text{et} \quad \sum_{k=1}^{N_b} X_{b\_k} = 1$$

On note pour chaque moteur: K_{v_i} sa constante de vitesse, r_i sa résistance interne, I_{0_i} son courant à vide, $I_{max\_m\_i}$ son courant maximal, M_{m_i} sa masse, Ns_{m_i} son nombre cellules en série dans la batterie pour avoir sa tension optimale $U_{N\_m\_i}$ et $[I_{Nmin\_i}, I_{Nmax\_i}]$ sa plage de courant nominal. Et on a:

$$\forall i \in [\![1,N_m]\!] \; U_{N\_m\_i} = 3.7 \cdot Ns_{m\_i}$$

On note pour chaque hélice: D_{p_j} son diamètre, Ω_j sa vitesse de rotation (obtenue par la méthode de la partie I.4.1) Γ_j le couple obtenu, et P_j la puissance nécessaire où $P_j = \Gamma_j \cdot \Omega_j$.

On note pout chaque batterie: Q_{b_k} sa capacité, D_{b_k} sa décharge max qui donne $I_{max\_b\_k}$ son courant maximal, M_{b_k} sa masse et Ns_{b_i} son nombre de cellules en série qui donne sa tension nominale U_{b_k}. Et on a:

$$\forall k \in [\![1,N_b]\!] \; U_{b\_k} = 3.7 \cdot Ns_{b\_k}\,,$$
$$I_{max\_b\_k} = D_{b\_k} \cdot Q_{b\_k}$$

Les variables continues du problème sont le courant I_{mot} et la tension U_{mot} du moteur, la tension de la batterie U_{bat}, la vitesse de rotation du moteur (et de l'hélice) Ω, le couple exercé par le moteur sur l'hélice Γ, le rapport cyclique du ESC α, la masse M de la motorisation (moteur et batterie) et l'inverse de l'autonomie inv_End.

On note $[\alpha_{min}, \alpha_{max}]$ la plage de rapport cyclique pour ce problème spécifique choisie par l'utilisateur. On doit laisser de la marge pour pouvoir augmenter la puissance en montée, par exemple. On note aussi M_{max} la masse maximale autorisée la motorisation (moteur et batterie) et End_{min} l'autonomie minimale du problème.

On a les équations:

$$\Omega = \sum_{j=1}^{N_p} \Omega_j \cdot X_{p\_j}$$

$$\Gamma = \sum_{j=1}^{N_p} \Gamma_j \cdot X_{p\_j}$$

$$U_{bat} = \sum_{k=1}^{N_b} U_{b\_k} \cdot X_{b\_k}$$

On utilise l'inverse car on ne peut pas faire de divisions dans un modèle linéaire. On doit aussi réécrite les équations non linéaires (multiplication des variables) pour les rendre linéaires. Pour cela on utilise des inégalités de la façon suivante:

Pour avoir l'équation (1) on devrait écrite :

$$\Gamma = (I - \sum_{i=1}^{N_m} I_{0_i} \cdot X_{m\_i}) / \sum_{i=1}^{N_m} K_{v\_i} \cdot X_{m\_i})$$

Ce qui est non linéaire. Soit Γ_M le couple maximal possible (en pratique juste un nombre très grand devant le couple), On écrit alors:

(1) $\forall i \in [\![1,N_m]\!]$,
$$\Gamma \leq (I_{mot} - I_{0\_i}) \;/\; K_{v\_i} \; + \; (1 - X_{m\_i}) \cdot \Gamma_M$$
$$\Gamma \geq (I_{mot} - I_{0\_i}) \;/\; K_{v\_i} \; - \; (1 - X_{m\_i}) \cdot \Gamma_M$$

On n'as alors l'égalité pour le moteur i lorsque sa variable binaire vaut 1. Les autres inégalités sont alors "désactivées". On obtient globalement l'égalité (1).

On a de même:

(2) $\forall i \in [\![1,N_m]\!]$,
$$Imot \leq \frac{Umot}{r_i} - \frac{\Omega}{r_i \cdot K_{v\_i}} + \; (1 - X_{m\_i}) \cdot I_M$$
$$Imot \geq \frac{Umot}{r_i} - \frac{\Omega}{r_i \cdot K_{v\_i}} - \; (1 - X_{m\_i}) \cdot I_M$$
(3) $\forall i \in [\![1,N_b]\!]$,
$$Umot \leq \alpha \cdot U_{b\_k} \; + \; (1 - X_{b\_k}) \cdot U_M$$
$$Umot \geq \alpha \cdot U_{b\_k} \; - \; (1 - X_{b\_k}) \cdot U_M$$
(4) $\forall i \in [\![1,N_b]\!]$,
$$inv\_End \leq I_{mot} \;/\; Q_{b\_k} \; + \; (1 - X_{b\_k}) \cdot 1/E_m$$
$$inv\_End \geq I_{mot} \;/\; Q_{b\_k} \; - \; (1 - X_{b\_k}) \cdot 1/E_m$$

Avec I_M et le courant et tension maximales possibles (en théorie) et E_m l'autonomie minimale possible (en relaté elle devrait être nulle mais on prend un nombre très petit).

On rester dans la plage nominale pour le moteur:

$$U_{bat} = \sum_{k=1}^{N_b} U_{N\_m\_i} \cdot X_{b\_k}$$

$$I_{mot} \geq \sum_{i=1}^{N_m} X_{m\_i} \cdot I_{Nmin\_i}$$

Et on a aussi les autres contraintes à respecter:

$$M \leq M_{max}$$

$$inv\_end \leq 1/End_{min}$$

$$\alpha \leq \alpha_{max}$$

$$\alpha \geq \alpha_{min}$$

$$I_{mot} \leq \sum_{k=1}^{N_b} X_{b\_k} \cdot I_{max\_b\_k}$$

$$I_{mot} \leq \sum_{i=1}^{N_m} X_{m\_i} \cdot I_{max\_m\_i}$$

Il ne reste à définir que la fonction objective. Si on veut maximiser l'autonomie (avec une masse donnée, par exemple) on cherche alors $Min\ inv\_End$. Pour minimiser la masse (avec une autonomie minimale donnée, par exemple) on cherche $Min\ M$

II.2. Implementation

On implémente le problème sur Python avec l'interface PuPL [ref] (open source) et le solveur CPLEX [ref] de IBM (Payant mais licence éducation disponible)

On décompose le programme en plusieurs modules (ou fonctions):

get_prop_chars_UIUC

Le module, *get_prop_chars_UIUC,* défini une fonction qui prend comme entrée le fichier de la base de données correspondant à une hélice, la poussée désirée, la masse volumique et la vitesse et le diamètre de l'hélice. Ce module réalisé les interpolations décrites en I.4.1 pour obtenir le point de fonctionnent de l'hélice. Pour les interpolation on a besoin de au moins 3 points à une vitesse donnée; si ce n'est pas le cas le module ne donne pas un point de fonctionnement
Ce module renvoie: un binaire qui vaut 1 si un point de fonctionnent à été trouvée et 0 sinon, la vitesse de rotation (en RPM), la puissance nécessaire, le rendement et le couple qu'il faut fournir.

fetch_data

Ce module définit trois fonctions:

fetch_motor_data
Cette fonction importe la base de données moteur. La fonction prend en entrée une liste Exclude_list de moteurs à exclure (qui peut être vide) et une liste de moteurs à inclure exclusivement Include_only_list (qui peut être vide). La fonction renvoie la base de données sous forme d'une liste de *dictionaries* (un *dictionary* par moteur) avec le nom, K_v (en SI), I_0, r, M_m (en g), Ns_m, I_{Nmax} et I_{max_m} (avec les clés `'name','k','i0','r','mass',nb_s','i_nom_max'` et `'i_max'` respectivement).

fetch_prop_data
Utilise la fonction *get_prop_chars_UIUC* pour calculer le point de fonctionnement des hélices. La fonction prend comme paramètres d'entrée la poussée désirée, la vitesse, la masse volumique, le diamètre minimal et maximal désiré par l'utilisateur ainsi que des listes Exclude_list et Include_only_list (qui peuvent être vides). Elle renvoi une liste de dictionnaires (un par hélice qui est dans l'intervalle de diamètres souhaité et dont on a trouvé un point de fonctionnement) contenant : le nom, le diamètre, la vitesse de rotation (en rpm), la puissance nécessaire, le couple et le rendement (avec les clés `'name', 'diam', 'RPM', 'W', 'torque'` et `'eta'` respectivement).

fetch_battery_data
Importe la base de donnés des batteries. Prend comme entrée la masse maximale de la motorisation Mmax, le nombre maximal de batteries en parallèle Nbmax et des listes Exclude_list et Include_only_list (qui peuvent être vides). Cette fonction "assemble" des batteries du même type en parallèle (dans la limite Nbmax et de Mmax) pour en faire des nouvelles avec les mêmes caractéristiques mais plus de capacité. Elle renvoi une liste de dictionnaires contenant: son nom (qui contient le nombre de batteries en parallèle), sa masse, sa capacité (en Ah), sa tension nominale et son courant maximal (avec les clés `'name','mass','cap','U'`et `'i_max'` respectivement).

prop_optim

Fonction qui réalise l'optimisation. Elle prend comme entrée la poussée désirée, la vitesse, la

masse volumique, la masse maximale de la motorisation, le nombre maximal de batteries en parallèle, l'autonomie minimale, le rapport cyclique minimal et maximal, le diamètre minimal et maximal ainsi que des listes Exclude_list et Include_only_list (qui peuvent être vides) pour des moteurs et des hélices et un vecteur qui donne les coefficients de la fonction objective ([1,0,0] pour optimiser l'endurance seule , [0,1,0] pour optimiser la masse seule, [0,0,1] pour la puissance batterie seule)

Elle fait appel aux fonctions du module *fetch_data* pour obtenir les donnés des moteurs, hélices et batteries, utilise le formalisme PuLP pour définir le problème et le *solver* CPLEX pour le résoudre.

Formalisme PuLP
Pour définir le problème on écrit:

```
prob = pulp.LpProblem("propOptim",
pulp.LpMinimize)
```

En suite on définit les vecteurs de variables binaires, par exemple:

```
xm = np.empty((Nmot), dtype=object)
    for j in range(Nmot):
          xm[j] = pulp.LpVariable(
          'Xm_{}'.format(j),
          cat=pulp.LpBinary)
```

On définit aussi les variables continues, par exemple:

```
Omega=pulp.LpVariable('Omega')
```

On impose en suite les contraintes d'égalité, comme:

```
 prob += pulp.lpSum(xm) == 1
```

Ou des contraintes d'inégalité (ici utilisant les vecteurs de variables binaires)

```
prob += Torque <= I*(1/motor_data[i]['k'])
- motor_data[i]['i0']/motor_data[i]['k'] +
(1-xm[i])*Tormax
```

On définit aussi la fonction objective (à minimiser dans notre cas):

```
prob += inv_end
```

et on résout à l'aide de CPLEX:

```
status = prob.solve(CPLEX())
```

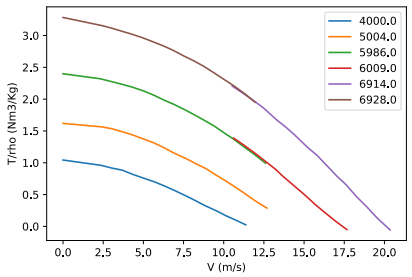
Finalement on vérifie bien que les paramètres optimisés pour le moteur donnent bien le couple demande (ici à moins de 0.1% d'écart).

La fonction prop_optim renvoi: un string contenant le statut de l'optimisation ('Optimal' si l'optimisation à abouti), le courant moteur, la tension de la batterie, le rapport cyclique, la vitesse de rotation (en RPM), l'autonomie (en heures), la masse de la motorisation et trois dictionnaires contenant les caractéristiques du moteur, hélice, et batterie (définis dans le module *fetch_data*).

II.3. Bases de données

La base de donnée moteurs utilisée pour cette première version est un base de donnée de moteurs AXI utilisée dans droneCalc. D'autres bases de donnée sont disponibles sur internet mais elles ne contiennent, pour la plus part que K_v, I_0, r et la masse, ce qui est insuffisant si on veut travailler dans la plage de rendement maximal. En plus de cela, la base de données de droneCalc en continent que I_{Nmax} on a donc approché I_{Nmin} par $0.5 \cdot I_{Nmax}$.

La basse de données hélices doit être traitée car non seulement on doit faire les changements décrits dans la partie I.4 mais on doit aussi assembler des morceaux de courbes. Sur la base de données brute, on utilise une vitesse de rotation proche pour prolonger les courbes.



On assimile alors tout à une seule courbe avec une seule vitesse de rotation.

En plus de cela, les vitesses de rotation de la base de données expérimentale sont trop restrictives on choisit donc de extrapoler ces résultats pour avoir accès à plus de points de fonctionnement

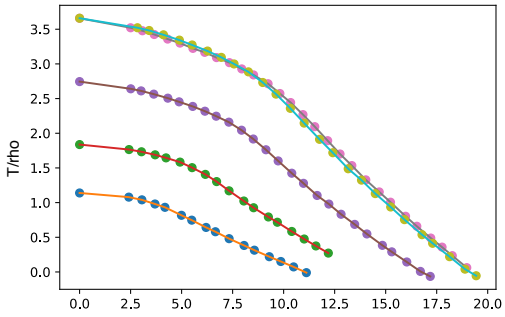
II.3.1 Extrapolation

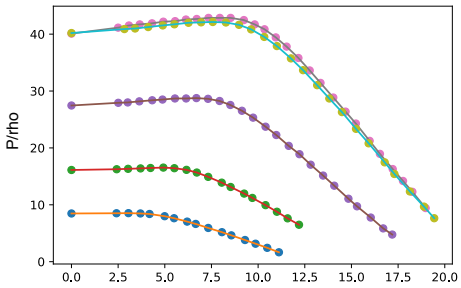
On commence par remarquer que les courbes semblent, dans la plus part des cas garder la même forme mais être "amplifiées" lorsque la

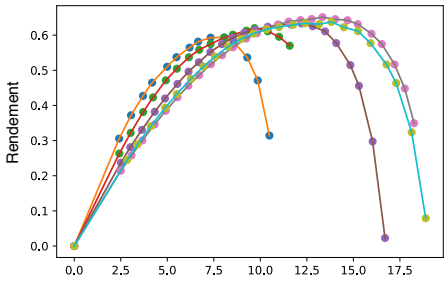
vitesse de rotation augmente. On remarque aussi que sur la plus part due courbes la pente devient constante à partir d'une certaine vitesse. On suit alors la démarche suivante:

1. On trouve les points d'intersection des courbes avec la ligne T/rho=0 (ou P/rho=0) on note $V_{t=0\_RPM_{max}}$ et $V_{p=0\_RPM_{max}}$ ces points pour la valeur la plus grande des RPM disponibles. On note aussi $t_{V=0\_RPM_{max}}$ et $p_{V=0\_RPM_{max}}$ l'intersection des courbes avec la ligne v=0 (trouvé comme décrit dans I.4)
2. On extrapole avec une fonction linéaire avec les points trouvés précédemment $V_{t=0\_RPM_{new}}$ et $V_{p=0\_RPM_{new}}$ pour le nouveau RPM, ainsi que $t_{V=0\_RPM_{new}}$ et $p_{V=0\_RPM_{new}}$ trouvés par extrapolation (I.4).
3. On note h le rapport des hypoténuses ($t_{V=0}$, $V_{t=0}$), i.e.:
$$h = \frac{\sqrt{(V_{p=0\_RPM_{new}})^2 + (t_{V=0\_RPM_{new}})^2}}{\sqrt{(V_{p=0\_RPM_{max}})^2 + (t_{V=0\_RPM_{max}})^2}}$$
4. On multiplie les points de la courbe T/rho du RPM_{max} par h^2 et les points le la courbe P/rho par h^3 et on ajoute une constante pour que on ait bien $t_{V=0\_RPM_{new}}$ et $p_{V=0\_RPM_{new}}$. On obtient alors les points T/ρ_{new},P/ρ_{new}
5. On calcule alors la constante par laquelle il faut multiplier les vitesses des points pour que l'intersection des courbes avec l'ordonnée 0 soient bien $V_{t=0\_RPM_{new}}$ et $V_{p=0\_RPM_{new}}$ et on obtient V_{t_new} et V_{p_new} pour tous les points
6. On recalcule les valeurs des points P/rho aux valeurs V_{t_new} à partir d'une interpolation de s points (V_{p_new},P/ρ_{new}).

On voit ici l'exemple de l'hélice apc29ff 9x5 où l'on a extrapolé une courbe au RPM le plus grand à partir des 3 premières. On obtient des résultats un peu différents mais satisfaisants.







II.4 Liste de meilleurs résultats

L'algorithme utilisée ne fait que trouver le meilleur couplage hélice-moteur. Pour avoir une liste des meilleurs résultats le module *prop_optim_list* suit la démarche suivante:

1. On optimise et on trouver le moteur m_1 et l'hélice h_1.
2. On optimise sans l'hélice h_1
3. On optimise sans le moteur m_1
4. On optimise sans m_1 et h_1
5. On répète jusqu'à avoir le nombre de résultats désirés

III. Ameliorations

Il restent quelques choses à améliorer sur le programme.

D'abord on n'a pas pris en compte le choix de ESC (qui se fait normalement en fonction de son courant optimal), son poids ou son rendement [ref]. Normalement le courant de la batterie et celui du moteur ne sont par le mêmes (celui du moteur est plus élevé) on a, à priori, donc sous estimé l'autonomie.

Il faut aussi prendre en compte le fait que la tension aux bornes de la batterie varie dans le temps et celle qu'on a n'est que la tension moyenne.

Finalement on peut aussi adapter le programme pour fair de l'optimisation multi-point de fonctionnement pour avoir par exemple les performances en montée et en palier, ou pour des drones hybrides.