Projet Optimisation

Yakine RACHEDI, Jean-Louis DU, Jessica SADER

Table des matières

1 Description du projet

Dans le cadre de l'UE *Projet Optimisation*, nous avons réalisé un projet visant à optimiser la trajectoire d'un lanceur spatial afin de déterminer le lanceur le plus léger capable de placer un satellite de masse donnée sur une orbite cible.

Le projet s'est déroulé en trois étapes principales :

- **Optimisation par méthode SQP (Sequential Quadratic Programming)** : Nous avons développé et validé un algorithme d'optimisation SQP sur des cas tests fournis, notamment un problème d'étagement inspiré du lanceur Ariane. Cette méthode permet de résoudre des problèmes d'optimisation non linéaire.
- **Simulation de trajectoire** : Nous avons implémenté un simulateur simplifié de la trajectoire du lanceur, en supposant une trajectoire plane, des forces élémentaires (propulsion, gravité, etc.) et une commande paramétrique par phase de vol.
- **Dimensionnement du lanceur** : Nous avons itéré entre l'optimisation d'étagement et l'optimisation de trajectoire afin de dimensionner le lanceur optimal pour une mission donnée (charge utile et orbite visée).

Les algorithmes et les simulations ont été développés et implémentés en MATLAB, permettant d'obtenir une solution globale pour l'optimisation du lanceur. Ce projet a ainsi permis de mettre en pratique les méthodes vues en cours d'optimisation sur un problème concret de lancement spatial. **Données du lanceur**

Charge utile (kg)	Altitude (km)
1000	250

2 Description de l'algorithme SQP et validation sur des testes

2.1 Problème d'optimisation

Nous considérons un problème d'optimisation de la forme suivante :

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{avec la contrainte} \quad c(x) = 0,$$

où :

— $f : \mathbb{R}^n \to \mathbb{R}$ est la fonction (non linéaire) objectif à minimiser,

— $c : \mathbb{R}^n \to \mathbb{R}^m$ représente m contraintes d'égalités,

— $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ est le vecteur des variables d'optimisation.

Les contraintes $c(x) = 0$ sont données sous la forme :

$$c(x) = \begin{bmatrix} c_1(x) \\ c_2(x) \\ \vdots \\ c_m(x) \end{bmatrix} = 0,$$

où chaque $c_j : \mathbb{R}^n \to \mathbb{R}$ est une contrainte d'égalité.

La fonction lagrangienne correspondant à ce problème s'exprime sous la forme suivante :

$$\mathcal{L}(x, \lambda) = f(x) + \sum_{j=1}^{m} \lambda_j c_j(x),$$

où :

— $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ est la fonction lagrangienne,

— λ_j est le multiplicateur de Lagrange associé à la contrainte $c_j(x) = 0$.

Les λ_j représentent les multiplicateurs de Lagrange liés aux contraintes.

Les conditions KKT du premier ordre s'écrivent :

$$\nabla L(x, \lambda) = \begin{pmatrix} \nabla f(x) + \nabla c(x) \cdot \lambda \\ c(x) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

La résolution de ce système d'équations est effectuée à l'aide de la méthode **SQP**. À chaque itération, cette méthode résout un sous-problème quadratique obtenu en approximant le problème initial par un développement de Taylor au second ordre.

2.2 Algorithme de la méthode SQP

Étape 1 : Initialisation L'optimisation commence à partir d'un point initial $(x_k, \lambda_k) = (x_0, 0)$. Nous définissons également :

— Algorithme utilisé : `BFGS` ou `SR1`, bornes (si présentes), nombre d'appels de la fonction (`nfonc`), et initialisation de la hessienne approximée.

— Gradients $\nabla f(x_0)$ et $\nabla g(x_0)$.

— Paramètres initiaux : pas des différences finies $h = 10^{-6}$ x_0, tolérance de convergence : 10^{-2}, pénalisation $\rho = 10^{-1}$.

— Indicateur de convergence et indicateur d'échec.

Étape 2 : Mise à jour des gradients À chaque itération, les gradients $\nabla f(x_k)$ et $\nabla c(x_k)$ sont calculés en utilisant une approximation par différences finies, effectuée à l'aide de la fonction `gradient.m`.

Étape 3 : Approximation de la Hessienne par Quasi-Newton Le calcul direct de la Hessienne $\nabla_{xx}L$ est coûteux en termes de ressources computationnelles. Par conséquent, cette matrice est remplacée par une approximation, notée H, obtenue à l'aide des méthodes de quasi-Newton. Parmi les options disponibles, les méthodes BFGS et SR1 sont particulièrement efficaces. Ces approches sont implémentées dans la fonction `hessien.m`, permettant une estimation adaptée de la Hessienne pour chaque itération.

Étape 4 : Modification de la Hessienne L'approximation H obtenue par quasi-Newton n'est pas nécessairement définie positive, ce qui peut compromettre la convergence. Afin de garantir cette propriété, H est ajustée pour produire une matrice définie positive H', proche de H. La méthode BFGS assure intrinsèquement cette propriété, tandis que SR1 peut générer une matrice non définie positive, nécessitant une correction. Cette modification est réalisée à l'aide de la fonction `modif_H.m`.

Étape 5 : Résolution du sous-problème quadratique Une fois la matrice H' obtenue, le sous-problème quadratique est résolu pour déterminer la direction de déplacement $\lambda_k d_{QP}$ au cours de l'itération. Cette étape peut être assimilée à une itération de la méthode de Newton. Cependant, bien que cette méthode fournisse une direction de descente, un contrôle rigoureux est nécessaire pour s'assurer que la solution contribue efficacement à l'avancée vers l'optimum global. Cette résolution est réalisée à l'aide de la fonction `pb_quad.m`.

Étape 6 : Recherche linéaire avec la condition d'Armijo Le déplacement calculé à partir du sous-problème quadratique constitue une approximation locale et ne garantit pas toujours une amélioration immédiate de la fonction objectif globale. Pour remédier à cela, une méthode de globalisation est appliquée. La condition d'Armijo, implémentée dans la fonction `globalisation.m`, est utilisée pour déterminer un pas optimal dans la direction calculée. Cette approche assure la validité de la direction de descente et garantit une convergence progressive vers la solution optimale.

Gestion des échecs dans l'algorithme SQP Dans l'algorithme SQP, un échec est identifié lorsque **l'indicateur = 0**. Cela signifie que la procédure de globalisation n'a pas permis de trouver une nouvelle solution acceptable (x_k) pour l'étape actuelle. La gestion des échecs repose sur les étapes suivantes :

- **Lors du premier échec :** La matrice hessienne est réinitialisée à une matrice identité dans la fonction `hessien.m`, afin de simplifier le calcul de la direction de descente.
- **Lors du deuxième échec consécutif :** La pénalité ρ est augmentée d'un facteur 10, renforçant ainsi la priorité donnée à la faisabilité des contraintes dans le problème quadratique.
- **Lorsqu'une solution acceptable est trouvée (*indicateur* $\neq 0$) :** La variable *echec* est réinitialisée à zéro, et l'algorithme reprend son processus normal.

Cette stratégie permet à l'algorithme d'ajuster ses paramètres dynamiquement pour gérer efficacement les situations de non-convergence tout en maintenant un équilibre entre faisabilité et optimalité.

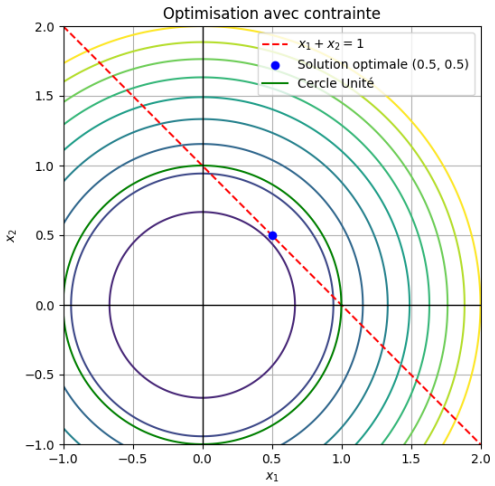
2.3 Validation de la méthode SQP

2.3.1 Test 1

$$\min_{x_1,x_2} x_1^2 + x_2^2$$

sous la contrainte :

$$c(x_1, x_2) = x_1 + x_2 - 1 = 0.$$



Test1

Itération	1
nfonc	16
$\mathbf{x_k}$	$(0.500000, 0.500000, 1.000000, 1.000000, 1.000000)$
$f(\mathbf{x_k})$	0.500000
$c(\mathbf{x_k})$	0.000000
λ_k	-1.100000
$\|L(\mathbf{x_k}, \lambda_k)\|$	0.000000

Résultats du test 1

Observations : Dans le cas du test 1, la solution recherchée est atteinte exactement après une seule itération.

2.3.2 Test MHW4D

$$\min_{x_1,x_2,x_3,x_4,x_5} f(x_1, x_2, x_3, x_4, x_5) = (x_1 - 1)^2 + (x_1 - x_2)^2 + (x_2 - x_3)^3 + (x_3 - x_4)^4 + (x_4 - x_5)^4$$

sous

$$\begin{cases} c_1(x_1, x_2, x_3, x_4, x_5) = x_1 + x_2^2 + x_3^2 - 3\sqrt{2} - 2 = 0, \\ c_2(x_1, x_2, x_3, x_4, x_5) = x_2 - x_3^2 + x_4 - 2\sqrt{2} + 2 = 0, \\ c_3(x_1, x_2, x_3, x_4, x_5) = x_1 x_5 - 2 = 0 \end{cases}$$

Bornes :

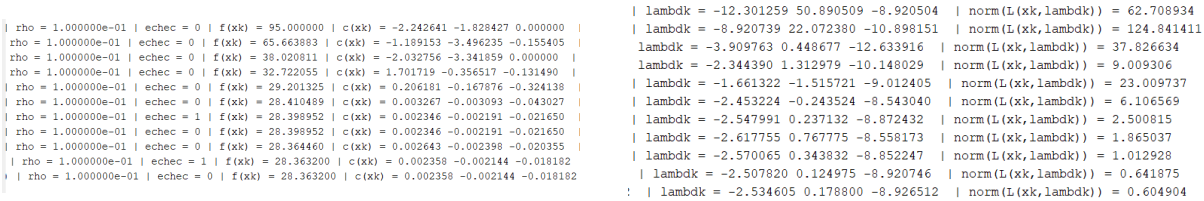
$$\begin{aligned} -2 < x_1 < 0, \\ 1 < x_2 < 3, \\ 0 < x_3 < 2, \\ -3 < x_4 < 0, \\ -3 < x_5 < -1. \end{aligned}$$

Initialisation :

$$(x_1, x_2, x_3, x_4, x_5) = (-1; 2; 1; -2; -2) \quad \rightarrow \quad f = 95$$

Solution :

$$(x_1, x_2, x_3, x_4, x_5) = (-1.2366; 2.4616; 1.1911; -0.2144; -1.6165) \quad \rightarrow \quad f = 28.4974$$




```
Iter = 0 | nfonc = 11 | xk = (-1.000000 2.000000 1.000000 -2.000000 -2.000000 ) |
Iter = 1 | nfonc = 20 | xk = (-0.721248 1.332192 2.000000 0.000000 -2.557504 ) |
Iter = 2 | nfonc = 28 | xk = (-2.000000 1.486568 2.000000 0.000000 -1.000000 ) |
Iter = 3 | nfonc = 36 | xk = (-1.487183 2.686448 1.488132 0.000000 -1.256408 ) |
Iter = 4 | nfonc = 44 | xk = (-0.917889 2.490733 1.078406 -0.667224 -1.825777 ) |
Iter = 5 | nfonc = 53 | xk = (-1.178444 2.477568 1.134024 -0.366225 -1.660641 ) |
Iter = 6 | nfonc = 71 | xk = (-1.196711 2.469308 1.159402 -0.298859 -1.653155 ) |
Iter = 7 | nfonc = 76 | xk = (-1.196711 2.469308 1.159402 -0.298859 -1.653155 ) |
Iter = 8 | nfonc = 87 | xk = (-1.190611 2.459331 1.177958 -0.245717 -1.662714 ) |
Iter = 9 | nfonc = 105 | xk = (-1.207694 2.459498 1.184721 -0.229652 -1.640993 )
Iter = 10 | nfonc = 112 | xk = (-1.207694 2.459498 1.184721 -0.229652 -1.640993 )
xk_final = (-1.210537 2.460118 1.184618 -0.230481 -1.637365 )
```

Résultats des itérations de la méthode SQP pour le problème M4HWD

Observations : La solution recherchée est atteinte après 10 itérations, avec une précision limitée uniquement par les erreurs d'arrondi. La pénalisation ρ reste constante à 0,1 tout au long du processus, et la hessienne est réinitialisée à une matrice identité à deux reprises.

Le nombre d'itérations est principalement contrôlé par la perturbation des valeurs propres :

— Avec une perturbation de $0,0001$, la solution optimale est atteinte en 10 itérations.

— Avec une perturbation de $0,01$, 333 itérations sont nécessaires pour converger, toujours à quelques erreurs d'arrondi près.

2.3.3 Test : Ariane1

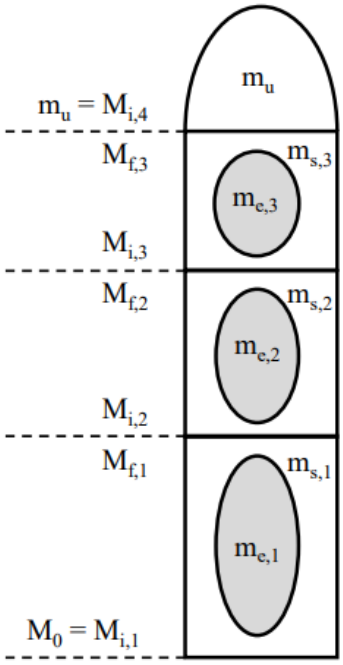
$$\min_{m_{e_1}, m_{e_2}, m_{e_3}} M_0 \quad \text{sous} \quad \Delta V = \Delta V_{\text{requis}}$$

— **Calcul de la masse totale M_0 en partant de la charge utile $M_{i,4} = m_u$:**

$$\begin{cases} m_{sj} = k_j m_{ej} \\ M_{f,j} = M_{i,j+1} + m_{sj} \\ M_{i,j} = M_{f,j} + m_{ej} \end{cases}$$

— **Calcul du ΔV :**

$$\Delta V = \sum_{j=1,3} \Delta V_j = \sum_{j=1,3} v_{ej} \ln \frac{M_{i,j}}{M_{f,j}}$$



$$\begin{aligned} k_1 &= 0.1101 & v_{e_1} &= 2647.2\,\mathrm{m/s} \\ k_2 &= 0.1532 & v_{e_2} &= 2922.4\,\mathrm{m/s} \\ k_3 &= 0.2154 & v_{e_3} &= 4344.3\,\mathrm{m/s} \\ \Delta V_{\text{requis}} &= 11527\,\mathrm{m/s} & m_u &= 1700\,\mathrm{kg} \end{aligned}$$

Solution

$$(m_{e_1}, m_{e_2}, m_{e_3}) = (145349\,\mathrm{kg}; 31215\,\mathrm{kg}; 7933\,\mathrm{kg}) \quad \rightarrow \quad M = 208611\,\mathrm{kg}$$

```
Iter = 0 | nfonc = 16 | xk = (100000.000000 50000.000000 10000.000000 )
Iter = 1 | nfonc = 27 | xk = (100000.000000 50000.000000 10000.000000 )
Iter = 2 | nfonc = 38 | xk = (100000.000000 50000.000000 10000.000000 )
Iter = 3 | nfonc = 49 | xk = (100000.000000 50000.000000 10000.000000 )
Iter = 4 | nfonc = 60 | xk = (100000.000000 50000.000000 10000.000000 )
Iter = 5 | nfonc = 61 | xk = (100000.000000 50000.000000 10000.000000 )
Iter = 6 | nfonc = 67 | xk = (122787.051866 60964.805981 12772.098153 )
Iter = 7 | nfonc = 73 | xk = (126749.149675 62881.006235 12194.340636 )
Iter = 8 | nfonc = 79 | xk = (126933.913687 62968.999448 12168.208348 )
Iter = 9 | nfonc = 85 | xk = (126934.279057 62968.849496 12167.954633 )
Iter = 10 | nfonc = 94 | xk = (126773.754004 61587.040593 11376.386420 )
```

```
| rho = 1.000000e-01 | echec = 1 | f(xk) = 182524.000000 | c(xk) = 356.940079  | lambdk = 3626377.516452  | norm(L(xk,lambdk)) = 50881.271560
| rho = 1 | echec = 2 | f(xk) = 182524.000000 | c(xk) = 356.940079  | lambdk = 3626377.516452  | norm(L(xk,lambdk)) = 50881.271560
| rho = 10 | echec = 3 | f(xk) = 182524.000000 | c(xk) = 356.940079  | lambdk = 3626377.516452  | norm(L(xk,lambdk)) = 50881.271560
| rho = 100 | echec = 4 | f(xk) = 182524.000000 | c(xk) = 356.940079  | lambdk = 3626377.516452  | norm(L(xk,lambdk)) = 50881.271560
| rho = 1000 | echec = 5 | f(xk) = 182524.000000 | c(xk) = 356.940079  | lambdk = 3626377.516452  | norm(L(xk,lambdk)) = 50881.271560
| rho = 1000 | echec = 0 | f(xk) = 182524.000000 | c(xk) = 356.940079  | lambdk = 3626377.516452  | norm(L(xk,lambdk)) = 50881.271560
| rho = 1000 | echec = 0 | f(xk) = 223833.728630 | c(xk) = 58.558059  | lambdk = 78738.817633  | norm(L(xk,lambdk)) = 1439.542541
| rho = 1000 | echec = 0 | f(xk) = 229739.609653 | c(xk) = 2.482742  | lambdk = 762.625749  | norm(L(xk,lambdk)) = 11.367194
| rho = 1000 | echec = 0 | f(xk) = 230014.428174 | c(xk) = 0.005421  | lambdk = 111.202400  | norm(L(xk,lambdk)) = 2.380188
| rho = 1000 | echec = 0 | f(xk) = 230014.352401 | c(xk) = 0.000000  | lambdk = 26.904516  | norm(L(xk,lambdk)) = 1.980980
```

```
Iter = 86 | nfonc = 550 | xk = (145400.681733 31238.703778 7935.509992 )
Iter = 87 | nfonc = 556 | xk = (145400.724559 31238.660814 7935.511641 )
Iter = 88 | nfonc = 562 | xk = (145400.755853 31238.629043 7935.513204 )
Iter = 89 | nfonc = 568 | xk = (145400.778903 31238.605827 7935.514178 )
Iter = 90 | nfonc = 574 | xk = (145400.796778 31238.588304 7935.514478 )
Iter = 91 | nfonc = 580 | xk = (145400.811761 31238.574199 7935.514176 )
Iter = 92 | nfonc = 587 | xk = (145400.825092 31238.562235 7935.513351 )
xk_final = (145400.831089 31238.557114 7935.512733 )
```

```
rho = 1000 | echec = 0 | f(xk) = 208778.588833 | c(xk) = 0.000000  | lambdk = 117.139723  | norm(L(xk,lambdk)) = 0.000023
rho = 1000 | echec = 0 | f(xk) = 208778.588832 | c(xk) = 0.000000  | lambdk = 117.139737  | norm(L(xk,lambdk)) = 0.000014
rho = 1000 | echec = 0 | f(xk) = 208778.588832 | c(xk) = 0.000000  | lambdk = 117.139749  | norm(L(xk,lambdk)) = 0.000007
rho = 1000 | echec = 0 | f(xk) = 208778.588832 | c(xk) = 0.000000  | lambdk = 117.139749  | norm(L(xk,lambdk)) = 0.000005
rho = 1000 | echec = 0 | f(xk) = 208778.588832 | c(xk) = 0.000000  | lambdk = 117.139735  | norm(L(xk,lambdk)) = 0.000006
rho = 1000 | echec = 0 | f(xk) = 208778.588832 | c(xk) = 0.000000  | lambdk = 117.139708  | norm(L(xk,lambdk)) = 0.000007
rho = 1000 | echec = 0 | f(xk) = 208778.588832 | c(xk) = 0.000000  | lambdk = 117.139668  | norm(L(xk,lambdk)) = 0.000007
```

Résultats des itérations de la méthode SQP pour le problème Ariane1

Observations : La solution recherchée est atteinte après 92 itérations, avec une précision limitée uniquement par les erreurs d'arrondi, similaire au test précédent. En diminuant la perturbation, on observe une réduction du nombre d'itérations :

— Initialement, une version générait 3340 itérations.

— Avec des ajustements dans les paramètres et les tolérances, ce nombre a été réduit à 437 itérations.

— Enfin, la dernière version atteint la convergence en seulement 92 itérations, à quelques erreurs d'arrondi près.

3 Deuxième Partie : Simulateur de la Trajectoire du Lanceur

3.1 Problème d'Étagement

L'étagement représente une étape cruciale dans la conception des lanceurs spatiaux. Cela permet de minimiser la masse totale du lanceur en larguant progressivement les structures vides des étages à mesure que les ergols sont consommés. L'objectif principal est d'optimiser et de maximiser l'efficacité en termes de masse tout en assurant que la vitesse de propulsion requise soit atteinte.

Valeurs numériques (Ariane 1)

Etage	Indice constructif k	Vitesse d'éjection v_e (m/s)
EP1	0,10	2600
EP2	0,15	3000
EP3	0,20	4400

Valeurs numériques (Données Ariane1)

Dans ce contexte, le problème d'étagement est formulé comme un problème d'optimisation non linéaire sous contraintes.

On cherche à concevoir le lanceur le plus léger possible capable d'atteindre une vitesse propulsive donnée V_p pour un satellite de masse m_u.

Le problème d'optimisation est formulé comme suit :

$$\max_{m_{e_1}, m_{e_2}, m_{e_3}} J = \frac{m_u}{M_0} \tag{1}$$

sous la contrainte :

$$V_p = \sum_{j=1}^{3} V_{e_j} \ln\left(\frac{M_{ij}}{M_{fj}}\right),$$

où :

— m_{e_i}, $i = 1, 2, 3$, sont les masses d'ergols des trois étages

— m_u est la masse du satellite

— M_0 est la masse totale du lanceur.

L'objectif est donc d'optimiser la performance tout en respectant la contrainte liée à la vitesse propulsive.

3.1.1 Résolution Analytique

Montrons que l'on peut reformuler le problème d'optimisation d'étagements comme suit :

$$\min_{x_1, x_2, x_3} f(x_1, x_2, x_3) \quad \text{sous la contrainte} \quad c(x_1, x_2, x_3) = 0, \tag{2}$$

avec :

$$f(x_1, x_2, x_3) = -\left(\frac{1+k_1}{x_1} - k_1\right)\left(\frac{1+k_2}{x_2} - k_2\right)\left(\frac{1+k_3}{x_3} - k_3\right),$$

$$c(x_1, x_2, x_3) = v_{e1}\ln x_1 + v_{e2}\ln x_2 + v_{e3}\ln x_3 - V_p$$

En posant, pour chaque étage $j = 1, 2, 3$, $x_j = \frac{M_{ij}}{M_{fj}}$, on peut écrire :

$$J = \frac{m_u}{M_0} = \frac{M_{i4}}{M_{i1}} = \frac{M_{i4}}{M_{i3}} \cdot \frac{M_{i3}}{M_{i2}} \cdot \frac{M_{i2}}{M_{i1}}.$$

Avec $M_{fj} = M_{ij} - m_{ej}$ et $m_{sj} = k_j m_{ej}$, on a, d'après la formule de séparation des étages :

$$M_{i(j+1)} = M_{fj} - m_{sj} = M_{fj} \cdot \frac{1}{M_{ij}} - m_{sj} \cdot \frac{1}{M_{ij}}.$$

Ainsi :

$$\frac{M_{i(j+1)}}{M_{ij}} = \frac{1}{x_j} - k_j\left(1 - \frac{1}{x_j}\right) = \frac{1}{x_j}(1 + k_j) - k_j.$$

D'où l'expression pour J devient :

$$J = \left(\frac{1}{x_1}(1+k_1) - k_1\right) \cdot \left(\frac{1}{x_2}(1+k_2) - k_2\right) \cdot \left(\frac{1}{x_3}(1+k_3) - k_3\right).$$

Le problème initial étant de maximiser J, on peut reformuler le problème en minimisant $f(x_1, x_2, x_3) = -J$. Ainsi, nous obtenons le problème d'optimisation désiré.

Conditions KKT (Karush-Kuhn-Tucker)

On pose :

$$f(x_1, x_2, x_3) = \left(\frac{1}{x_1}(1+k_1) - k_1\right) \cdot \left(\frac{1}{x_2}(1+k_2) - k_2\right) \cdot \left(\frac{1}{x_3}(1+k_3) - k_3\right) = -y_1 y_2 y_3$$

$$c(x_1, x_2, x_3) = v_{e1}\ln(x_1) + v_{e2}\ln(x_2) + v_{e3}\ln(x_3) - V_p$$

Si x^* est un minimum local de f, alors il existe un multiplicateur de Lagrange $\lambda \in \mathbb{R}$ tel que :

$$\nabla L(x^*, \lambda) = 0$$

où :

$$L(x^*, \lambda) = f(x^*) + \lambda c(x^*)$$

et :

$$\nabla L(x^*, \lambda) = \nabla f(x^*) + \lambda \nabla c(x^*)$$

Les dérivées partielles de f sont données par :

$$\frac{\partial f}{\partial x_1^*} = y_2 y_3 \frac{1+k_1}{(x_1^*)^2},$$
$$\frac{\partial f}{\partial x_2^*} = y_1 y_3 \frac{1+k_2}{(x_2^*)^2},$$
$$\frac{\partial f}{\partial x_3^*} = y_1 y_2 \frac{1+k_3}{(x_3^*)^2}.$$

Les dérivées partielles de c pour $j \in \{1, 2, 3\}$ sont :

$$\frac{\partial c(x_1^*, x_2^*, x_3^*)}{\partial x_j^*} = \frac{v_{e_j}}{x_j^*}$$

Le gradient $\nabla L(x^*, \lambda)$ s'écrit alors :

$$\nabla L(x^*, \lambda) = \left(\frac{\partial f}{\partial x_j^*} + \lambda \frac{\partial c}{\partial x_j^*} \right)_{j \in \{1,2,3\}} = 0 \quad \text{et} \quad c(x_1^*, x_2^*, x_3^*) = 0$$

On obtient donc :

$$\nabla L(x^*, \lambda) = \begin{pmatrix} y_2 y_3 \frac{1+k_1}{(x_1^*)^2} + \lambda \frac{v_{e1}}{x_1^*} \\ y_1 y_3 \frac{1+k_2}{(x_2^*)^2} + \lambda \frac{v_{e2}}{x_2^*} \\ y_1 y_2 \frac{1+k_3}{(x_3^*)^2} + \lambda \frac{v_{e3}}{x_3^*} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

Montrons que $v_{e_j}(1 - \Omega_j x_j^*) =$ cte, où $\Omega_j = \frac{k_j}{1+k_j}$:

Pour $j = 1$, on a :

$$\begin{aligned} v_{e_1}(1 - \Omega_1 x_1^*) &= v_{e_1} \left(1 - \frac{k_1 x_1^*}{1 + k_1} \right) \\ &= v_{e_1} \left(\frac{(1 + k_1) - k_1 x_1^*}{1 + k_1} \right) \\ &= \frac{v_{e_1} y_1 x_1^*}{x_1^*(1 + k_1)} \\ &= \frac{v_{e_1} y_1}{1 + k_1} \end{aligned}$$

car

$$y_1 = \frac{1 + k_1}{x_1^*} - k_1 = \frac{(1 + k_1) - k_1 x_1^*}{x_1^*}$$

Ainsi, on a :

$$\frac{\partial f}{\partial x_1^*} = \left(\frac{1 + k_2}{x_2^*} - k_2 \right) \left(\frac{1 + k_3}{x_3^*} - k_3 \right) \left(\frac{1 + k_1}{(x_1^*)^2} \right)$$

En simplifiant :

$$\frac{\partial f}{\partial x_1^*} = \left(-\frac{1 + k_1}{(x_1^*)^2} \right) \cdot \frac{f(x_1^*, x_2^*, x_3^*)}{1 + k_1} = \frac{(1 + k_1) f(x_1^*, x_2^*, x_3^*)}{(x_1^*)^2 y_1}$$

car

$$y_1 = \frac{1 + k_1}{x_1^*} - k_1$$

D'après les conditions KKT, on obtient :

$$-\frac{(1 + k_1)}{(x_1^*)^2 y_1} f(x_1^*, x_2^*, x_3^*) + \lambda \frac{v_{e_1}}{x_1^*} = 0$$

Cela implique :

$$\lambda v_{e_1} x_1^* y_1 = \frac{(x_1^*)^2 y_1}{1 + k_1} f(x_1^*, x_2^*, x_3^*),$$

et donc :

$$\lambda v_{e_1}(1 - \Omega_1 x_1^*) = \frac{f(x_1^*, x_2^*, x_3^*)}{1 + k_1}$$

Finalement :

$$v_{e_1}(1 - \Omega_1 x_1^*) = \frac{\text{cte}}{\lambda}$$

Sous les conditions KKT, en procédant de la même manière pour les indices $j = 2, 3$, on obtient :

$$v_{e_1}(1 - \Omega_1 x_1^*) = v_{e_2}(1 - \Omega_2 x_2^*) = v_{e_3}(1 - \Omega_3 x_3^*) = \text{cte}$$

Le problème se ramène à résoudre une équation à une inconnue

En partant de l'égalité précédente on voit qu'on a une expression de *cte* en fonction de x_3. De plus on peut exprimer x_2 et x_1 en fonction de *cte* et donc de x_3 par ce qu'on a évoqué précédemment. On obtient

$$x_1 = -\left(\frac{v_{e3}}{v_{e1}}\left(1-\Omega_3 x_3\right)-1\right)/\Omega_1$$
$$x_2 = -\left(\frac{v_{e3}}{v_{e2}}\left(1-\Omega_3 x_3\right)-1\right)/\Omega_2$$

La contrainte devient :

$$\begin{aligned} c(x_3) = & v_{e1}\ln\left(-\left(\frac{v_{e3}}{v_{e1}}\left(1-\Omega_3 x_3\right)-1\right)/\Omega_1\right) \\ & + v_{e2}\ln\left(-\left(\frac{v_{e3}}{v_{e2}}\left(1-\Omega_3 x_3\right)-1\right)/\Omega_2\right) \\ & + v_{e3}\ln x_3 - V_p. \end{aligned}$$

Résolution du problème par un algorithme de Newton

Pour se faire on a besoin du gradient de c.

$$\begin{aligned} \frac{dc}{dx_3}(x_3) = & \frac{-v_{e3}\Omega_3}{\Omega_1\left(\frac{v_{e3}}{v_{e1}}\left(1-\Omega_3 x_3\right)-1\right)/\Omega_1} \\ & + \frac{-v_{e3}\Omega_3}{\Omega_2\left(\frac{v_{e3}}{v_{e2}}\left(1-\Omega_3 x_3\right)-1\right)/\Omega_2} \\ & + \frac{v_{e3}}{x_3} \end{aligned}$$

En appliquant une méthode de Newton on obtient x3 dont on déduit x1 et x2. Puis en utilisant $x_j = \frac{M_{i,j}}{M_{e,j}}$ on trouve les masses d'ergols optimales et le poids total du lanceur. On peut retrouver le code dans M_Newton.

3.1.2 Résolution Numérique : application de l'algorithme SQP

L'algorithme SQP (Sequential Quadratic Programming) est un mécanisme puissant pour résoudre les problèmes d'optimisation non linéaire sous contraintes. Pour le problème d'étagement, les étapes suivantes sont mises en œuvre :

1. Initialisation : Les variables sont initialisées avec une estimation raisonnable des masses d'ergols.

2. Résolution du sous-problème quadratique : À chaque itération, un problème quadratique local est résolu pour minimiser une approximation quadratique du lagrangien, tout en tenant compte d'une version linéaire des contraintes.

3. Mise à jour des variables : Les solutions obtenues du sous-problème quadratique sont utilisées pour modifier et ajuster les valeurs des masses d'ergols et recalculer les contraintes.

4. Tests de convergence : L'algorithme s'arrête lorsque les conditions d'optimalité sont satisfaites à une précision donnée, vérifiant notamment que les gradients du lagrangien sont proches de zéro.

5. Globalisation : Une stratégie de recherche linéaire est appliquée pour assurer la solidité de l'algorithme et garantir une amélioration à chaque étape.

Cette méthode numérique fournit une validation pratique des solutions obtenues analytiquement et permet d'explorer des configurations plus complexes, en tenant compte des contraintes réelles du système.

On retrouve bien la même solution que la solution analytique trouvé précédemment.

3.2 Problème de Trajectoire

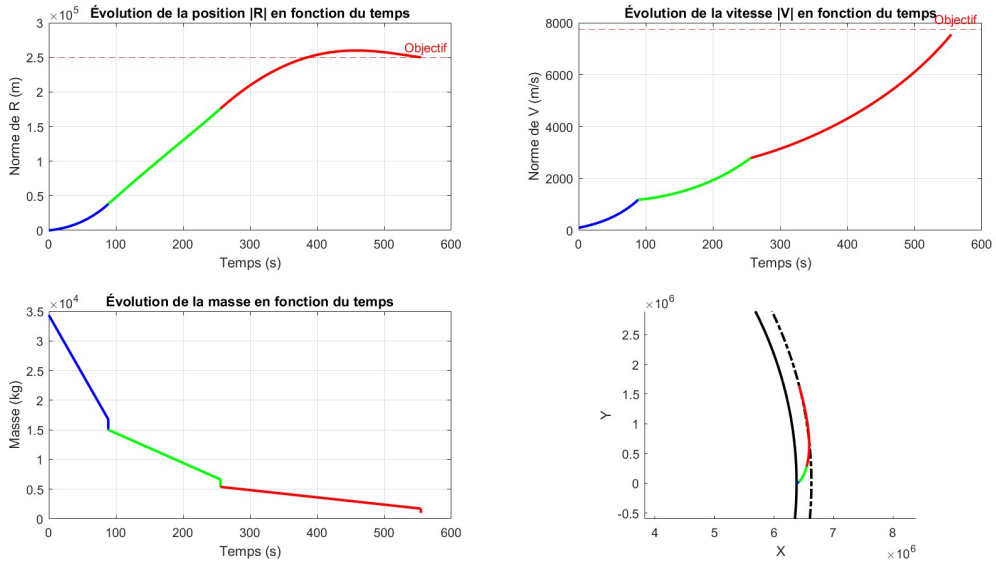
Après avoir déterminé les masses d'ergols dans chaque étage de la fusée, on veut maintenant optimiser sa trajectoire. Le problème qu'on a est le suivant :

$$\max_{\theta_0,\theta_1,\theta_2,\theta_3} V(t_f) \text{ sous } R(t_f) = Rc \text{ et } \vec{R}(t_f) \cdot \vec{V}(t_f) = 0$$

Physiquement, le problème est de maximiser la vitesse en ajustant les différents angles d'inclinaison de la fusée (qui est supposée constante le long d'un étage). C'est un problème sous contraintes, en effet on veut que la fusée arrive à la hauteur Rc qui est l'altitude à laquelle on veut que la fusée soit en orbite, et on veut qu'une fois sur cette orbite la vitesse soit orthogonale à la position pour lui permettre de rester sur l'orbite. De plus, on veut atteindre la vitesse Vc pour compenser l'attraction de la Terre tout en utilisant le moins d'ergols possible, d'où la maximisation.

Pour commencer on va chercher à simuler une trajectoire pour θ fixé. Pour un étage i fixé, on fait l'hypothèse que plusieurs paramètres sont fixés tels que la direction de poussée, la vitesse d'éjection ou encore l'angle θ_i. Les équations du mouvement dans le repère choisi sont des équations aux dérivées partielles qui nécessitent une intégration numérique. Pour ce faire on a donc créé une fonction integ modélisant les EDP du problème et on va donner cette fonction à l'intégrateur ode45 de matlab. L'intégrateur ode45 permet d'intégrer une fonction d'un temps initial à un temps final. On va bien faire attention à mettre les bonnes bornes temporelles. Pour calculer le temps final on calcule me/q_i avec q_i le débit massique pour l'étage i.

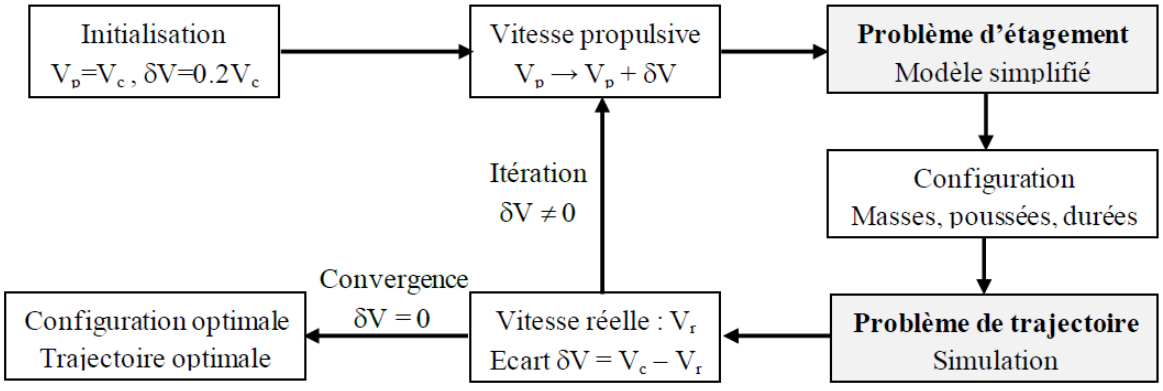
Ainsi, en répétant cet intégrateur pour chacun des étages, on se retrouve avec l'altitude au cours du temps, la vitesse au cours du temps, la distribution de masse au cours du temps et la trajectoire de notre lanceur.
On retrouve bien les bonnes distributions de masses à chaque étage, à l'issue du largage du 3ème étage il reste effectivement que la masse utile $m_u = 1000$ kg. On veut aussi vérifier que les vitesses et positions obtenues sont cohérentes. On voit que ces valeurs-là sont très dépendantes du θ choisi, selon les angles la fusée peut même s'enfoncer dans la Terre. On aurait aimé initialiser θ au hasard pour ensuite le donner à SQP pour qu'il nous trouve l'angle optimal, mais malheureusement cela ne fonctionne pas. Il faudrait avoir une idée plus précise de ces angles. Ainsi en tâtonnant un peu aux niveaux des angles on trouve une configuration de départ assez convenable.



Trajectoire du lanceur pour $\theta = \frac{\pi}{180} * [5; 1.2; 10; 5]$

Malgré le fait que le θ choisi est très bon la SQP ne donne pas du tout le résultat voulu. Même en ajustant les paramètres pour la SQP, on ne trouve pas d'état qui fonctionne pour tous les angles.
Le problème vient du fait que la trajectoire est très dépendante des angles θ_0 et θ_1. Ainsi on va les fixer pour n'optimiser que les angles θ_2 et θ_3. On fixe $\theta_0 = \frac{5\pi}{180}$ et $\theta_1 = \frac{1.2\pi}{180}$ car on a vu que ça permettait d'avoir une trajectoire correcte. Plus tard on tentera d'effectuer un balayage pour ces deux angles sur une plage de valeur qui fonctionne. En faisant cela la SQP fonctionne correctement.

Maintenant qu'on sait lancer une trajectoire et "l'optimiser", il faut calculer la différence entre la vitesse réelle Vr obtenue et la vitesse objectif Vc, $\delta V = Vc - Vr$. On retourne dans la partie étagement en ajustant la vitesse propulsive. L'idée est d'itérer jusqu'à ce que δV soit nulle. En effet, si $\delta V < 0$ cela veut dire que le lanceur est sur-performant, on arrive sur l'orbite désirée avec trop de vitesse. Si $\delta V > 0$ cela veut dire que le lanceur est sous-performant, on n'arrive pas sur l'orbite désirée avec assez de vitesse.



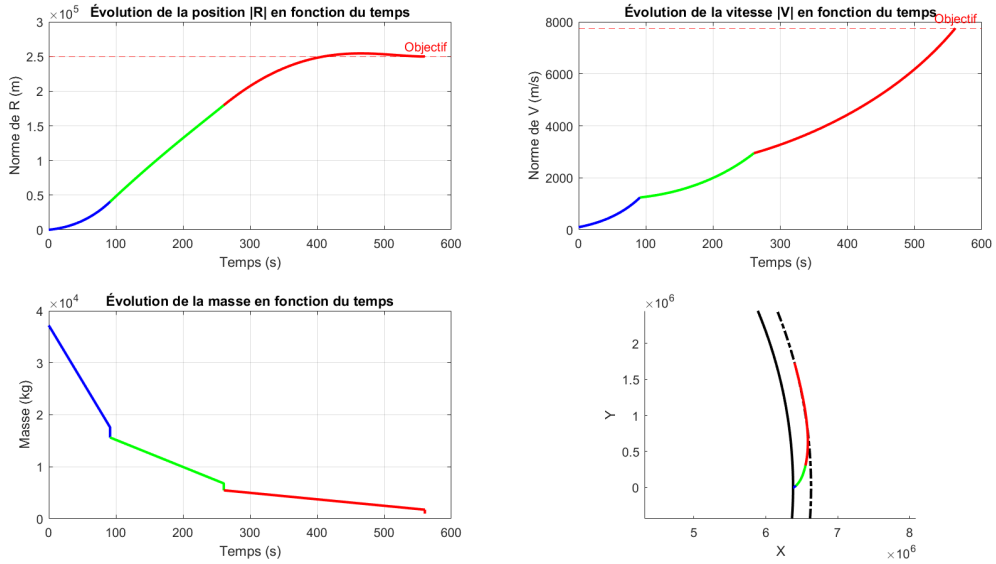
Algorithme final

On voit qu'en seulement quelques itérations on a un δV qui se rapproche de 0.

Itération	1	2	3	4	5
δV	197.42	-34.44	5.62	-0.88	0.15

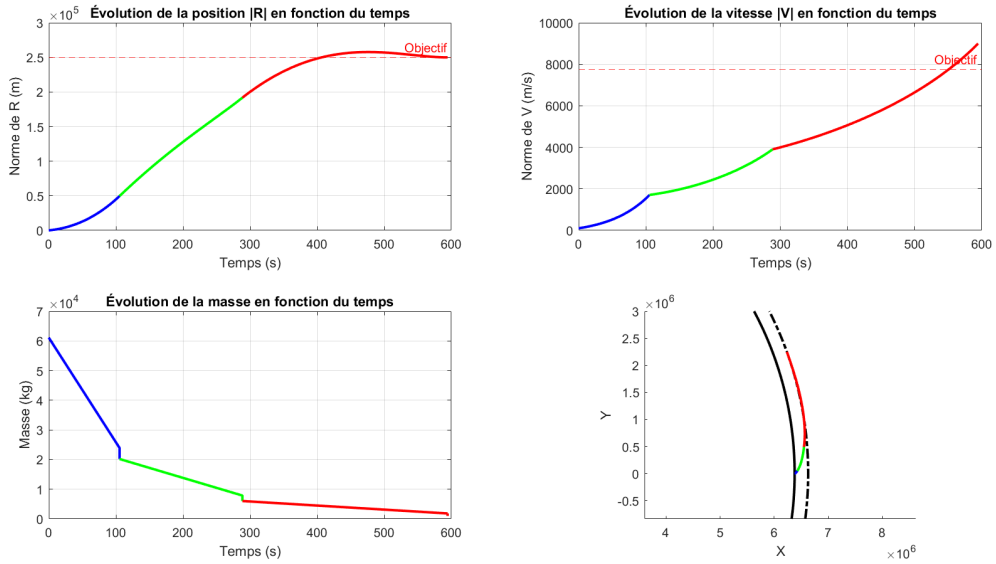
Tableau des itérations et des valeurs de δV

Notre lanceur est ainsi bien optimiser.



Trajectoire finale pour $\theta_0 = \frac{5\pi}{180}$ et $\theta_1 = \frac{1.2\pi}{180}$

On a aussi essayé de faire varier θ_0 et θ_1 en faisant un balayage. Mais on s'est confronté à plusieurs problèmes. Premièrement, il faut trouver des valeurs cohérentes qui permettent d'obtenir de bons résultats. Cela n'est pas facile car tous les angles θ sont très interdépendants. Deuxièmement, faire une boucle sur les θ choisis peut très vite s'avérer très long et coûteux en temps de calcul. On a tout de même réussi à mettre la méthode en place en faisant un balayage assez grossier sur des intervalles petits. On a choisi $\theta_0 \in [0 : 5]$ et $\theta_1 \in [0 : 2]$ avec des pas respectifs de 1 et 0.2.
On retrouve cette trajectoire-ci, on remarque que malgré le fait qu'on fasse la boucle sur δV, on a toujours une vitesse très grande.



Trajectoire finale avec un balayage sur θ_0 et θ_1

4 Conclusion

Au cours de ce projet, on a optimisé la masse et la trajectoire de notre lanceur pour que notre satellite de 1000 kg atteigne 250 km d'altitude. Pour ce faire, on a dû séparer le problème en plusieurs étapes, un problème d'étagement portant sur la répartition des masses d'ergols et un problème de trajectoire portant sur les angles donnant l'inclinaison au cours d'un étage. On a dû boucler plusieurs fois sur ces problèmes afin d'avoir un résultat satisfaisant. Malheureusement, on s'est heurté à des difficultés sur l'optimisation des angles. Cela nous a empêché d'avoir un lanceur parfaitement optimisé. Mais tout de même, avec les simplifications qu'on a faites, on arrive tout de même à des résultats.