Optimisation d'un lanceur spatial

MU5MAI04 : Ingénierie II, Projet Optimisation

GOPINATHAN Prédive
KRSTEVSKA Jovana

Master 2 : Ingénierie Mathématique
Ingénierie mathématique pour l'entreprise

11 mars 2021

Résumé

Dans ce projet, nous avons pour but d'amener un satellite en orbite autour de la Terre. L'objectif est de trouver le lanceur le plus léger possible permettant d'amener un satellite de 2000kg sur une orbite à une altitude de 250km, qui se déplacera avec une vitesse cible à atteindre.

Afin de faire cela, nous avons mis en place plusieurs routines en *Matlab*.

Dans une première partie du travail qui nous a été demandé, nous avons implémenté l'algorithme **SQP** afin de pouvoir résoudre les deux problèmes d'optimisation rencontrés : **le problème d'étagement** et **le problème de trajectoire**. Autrement dit, trouver des bonnes masses d'ergols pour les 3 étapes de la trajectoire et aussi trouver les bons paramètres pour que cette trajectoire soit correcte. Nous avons ensuite testé et validé ce logiciel grâce aux deux cas tests fournis : **MHW4D** et **Ariane1**.

Ensuite, nous avons résolu analytiquement ce problème d'étagement, et validé l'équation obtenue avec les valeurs fournies auparavant.

Dans la deuxième partie, nous avons mis en place une routine de simulation de trajectoires, qui nous a permis de pouvoir simuler plusieurs trajectoires de différents paramètres, afin de pouvoir trouver la trajectoire optimale en branchant cette routine à l'algorithme SQP dans la dernière partie du projet.

En combinant ces étapes, nous avons réussi à trouver une configuration et une trajectoire optimales, et confirmer la validité de ces résultats grâce à la résolution analytique.

Table des matières

1 Algorithme SQP

L'algorithme **SQP** pour **Sequential Quadratic Programming**, en anglais, est un algorithme de résolution d'un problème d'optimisation non linéaire. On cherche à résoudre un problème d'optimisation sous contraintes du type :

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{sous} \quad c(x) = 0$$

Notre programme **SQP** prend comme arguments :

x_init	initialisation des valeurs de x
lambda_init	initialisation des valeurs de λ
problem	fonction qui nous donne $f(x)$ et $c(x)$
choix	choix de l'algorithme utilisé dans Quasi-Newton : **BFGS** ou **SR1**
bornes	les bornes pour chaque coordonnée de x
max_iter	nombre maximal d'itérations
max_eval	nombre maximal d'évaluations de f et c (nombre d'appels à **problem**)
rho	pénalisation pour les contraintes
eps	tolérance pour les critères d'arrêt
max_rho	pénalisation maximale autorisée

Initialisation

On initialise x et λ à (x_0, λ_0) en tout premier.

Approximation du gradient par différences finies

Ensuite nous calculons le gradient par la méthode des différences finies décentrées. Nous obtenons une bonne approximation du gradient de la fonction à minimiser et de la jacobienne des contraintes.

Approximation du hessien par Quasi-Newton

Le hessien du lagrangien $Q = \nabla^2_{xx} L(x_k, \lambda_k)$ est très coûteux à calculer. On le remplace donc par une approximation H_k construite à partir du dernier déplacement, soit par la formule **BFGS**, soit par la formule **SR1**.

Modification de la hessienne afin de la rendre définie positive

Pour s'assurer que la hessienne est définie positive, nous allons la modifier, de sorte que chaque valeur propre soit strictement positive, et ainsi la rendant définie positive :

$$H_{nouvelle} = H + \tau I$$

où I est la matrice identité de même taille que la hessienne, avec $\tau > 0$ suffisamment grand.

Solution problème quadratique

Nous devons ensuite calculer la solution du problème quadratique local, qui est une approximation du "vrai" problème. On le résout en utilisant la méthode **KKT**. Mais comme ce n'est qu'une approximation, le déplacement calculé ne produit pas forcément une amélioration. Pour gérer ceci, notre prochaine étape est la globalisation.

Globalisation : Recherche linéaire

La partie la plus délicate de cet algorithme est la globalisation. C'est ici que nous vérifions si la solution obtenue par la section précédente est acceptée ou non.
L'évaluation de $\nabla_{xx}L$ étant très coûteuse, nous utilisons une fonction mérite $F(x)$ qui va mesurer une amélioration et a la formule suivante :

$$F(x) = f(x) + \rho\|c(x)\|_1$$

La solution QP (la solution du problème quadratique ci-dessus) est acceptée seulement si elle fait décroître la fonction mérite :

$$F(x_{k+1}) < F(x_k)$$

Si cela n'est pas le cas, on l'utilise comme direction de recherche pour construire un déplacement acceptable, en utilisant **la règle d'Armijo**.
Concrètement, la direction de la recherche linéaire d est en fait d_{QP}, un coefficient de Lagrange obtenu dans la partie précédente. Cette dernière doit être une direction de descente pour la fonction mérite, c'est-à-dire la dérivée directionnelle de la fonction mérite dans la direction de d doit être négative : $F_d^{'}(x_k) < 0$. Ceci est vérifié si le hessien est défini-positif, et si la pénalisation ρ est assez grande pour que les contraintes soient suffisamment respectées. Si cela n'est pas le cas on a 2 issues, classées par priorité :

1. On réinitialise la hessienne à l'identité
2. On augmente ρ (jusqu'à atteindre sa valeur maximale autorisée ρ_{max}, donnée en argument)

On itère ces étapes jusqu'à ce qu'un des critères d'arrêt ne soit satisfait :

- nombre maximal d'itérations atteint
- nombre maximal d'évaluations atteint
- déplacement insuffisant ($\|x_{k+1} - x_k\| < \varepsilon$)
- amélioration insuffisante ($|f(x_{k+1}) - f(x_k)| < \varepsilon$)

En sortie on obtient le nombre d'itérations effectuées, le nombre d'appels vers **problem**, et la valeur de x_k, $f(x_k)$, $c(x_k)$, λ_k, ρ et $\|\nabla L(x_k, \lambda_k)\|$ pour chaque itération k.

2 Tests numériques

Nous avons créé deux routines : *test_MHW4D* et *test_ariane1* pour tester notre algorithme sur ces deux cas tests proposés dans les instructions.

2.1 Cas test 1 : MHW4D

Pour ce problème on a les arguments suivants pour l'algorithme SQP :

x_init	[-1 ; 2 ; 1 ; -2 ; -2]
lambda_init	[0 ; 0 ; 0]
problem	MH4WD
choix	BFGS, puis SR1
bornes	[-2, 0 ; 1, 3 ; 0, 2 ; -3, 0 ; -3, -1]
max_iter	50
max_eval	1e4
rho	6
eps	1e-6
max_rho	2^{20}

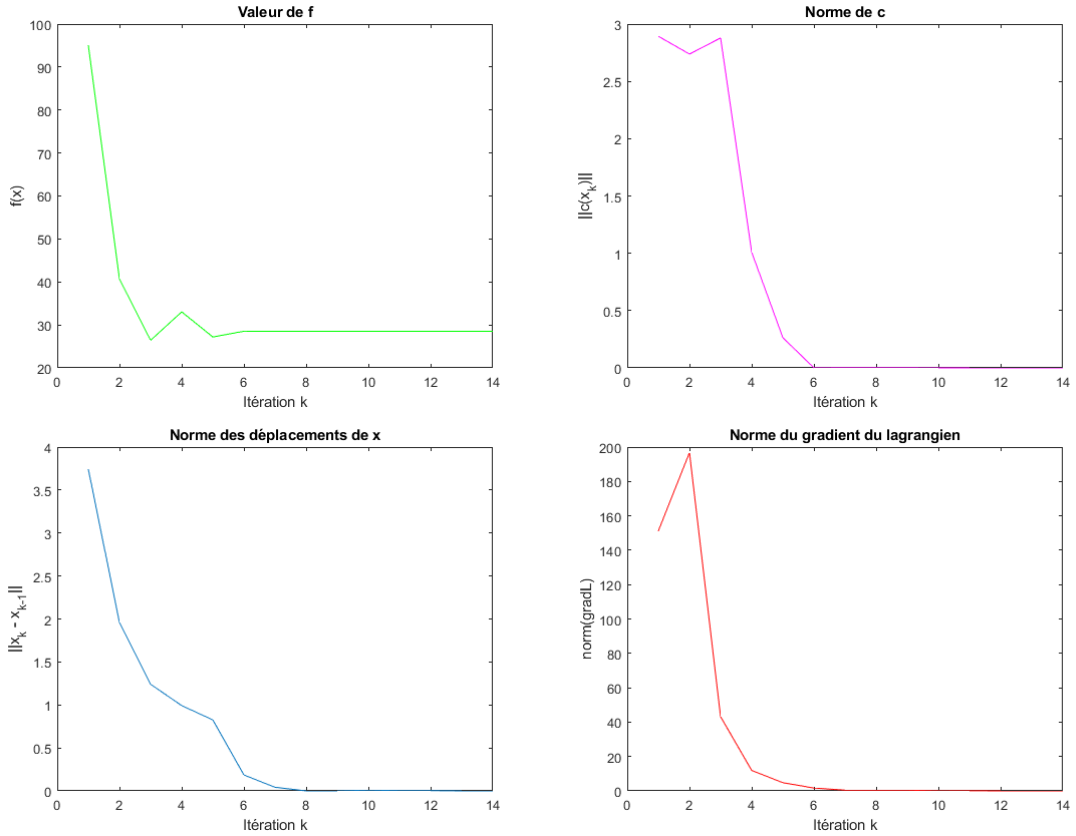
Déroulement de SQP avec la formule BFGS

k	nfonc	x_k	$f(x_k)$	$c(x_k)$	λ_k	$\|\nabla L(x_k, \lambda_k)\|$
1	14	(-1.0000,2.0000,1.0000,-2.000,-2.000)	95.0000	(-2.2426,-1.8284,0)	(0,0,0)	151.2283
2	26	(-0.8613,1.6427,1.6803,-0.2249,-2.2774)	40.6578	(-1.5820 ,-2.2342,-0.0385)	(-12.1902,51.1953,-8.8761)	196.7853
3	34	(-1.2914,1.6736,1.8075,0,-1.1431)	26.4197	(-1.4660,-2.4218,-0.5239)	(-5.5247,12.6289,-9.7787)	43.1501
4	42	(-1.6349,2.5463,1.5000,0,-1.2447)	33.0325	(0.8560,-0.5323,0.0349)	(-2.1453,-0.2109,-9.9957)	11.8867
5	50	(-1.0855,2.4582,1.1810,-0.3369,-1.6416)	27.1970	(0.1095,-0.1018,-0.2180)	(-2.5242,1.9344,-9.8129)	4.7872
⋮	⋮	⋮	⋮	⋮	⋮	⋮
11	84	(-1.2361,2.4623,1.1898,-0.2182,-1.6180)	28.5088	(0.0000,-0.0000,-0.0000)	(-2.5122,0.1240,-8.8966)	0.0847
12	92	(-1.2369,2.4620,1.1908,-0.2157,-1.6169)	28.5087	(0.0000,-0.0000,-0.0000)	(2.5124,0.1248,-8.8962)	0.0005
13	100	(-1.2369,2.4620,1.1908,-0.2157,-1.6169)	28.5087	(0.0000,-0.0000,-0.0000)	(-2.5124,0.1247,-8.8962)	0.0001
14	106	(-1.2369,2.4620,1.1908,-0.2157,-1.6169)	28.5087	(0.0000,-0.0000,-0.0000)	(-2.5124,0.1247,-8.8962)	0.0000

Notre algorithme converge après seulement **14 itérations**, et **106 évaluations du problème**. Le gradient du lagrangien descend assez vite. Voici la solution finale qu'on a obtenue, ainsi que la valeur finale de la fonction f, la valeur finale des contraintes et une comparaison des performances avec la solution donnée dans le polycopié du projet, stockée dans **sol_poly**.

```
sol = [-1.236939; 2.462045; 1.190762; -0.215703; -1.616895]
f(sol) = 28.508725
c(sol) = [0.000000; -0.000000; -0.000000]
sol_poly = [-1.236600; 2.461600; 1.191100; -0.214400; -1.616500]
f(sol_poly) = 28.496892
c(sol_poly) = [-0.001047; 0.000054; -0.001036]

sol - sol_poly =  [-0.000339; 0.000445; -0.000338; -0.001303; -0.000395]
f - f_poly = 0.011832
c - c_poly = [0.001047; -0.000054; 0.001036]
```

Voici quelques graphiques qui nous permettront de mieux évaluer cet algorithme. Nous avons affiché, pour chaque itération, la valeur de la fonction à minimiser, la valeur de la contrainte, la norme du lagrangien et la norme du déplacement de x.



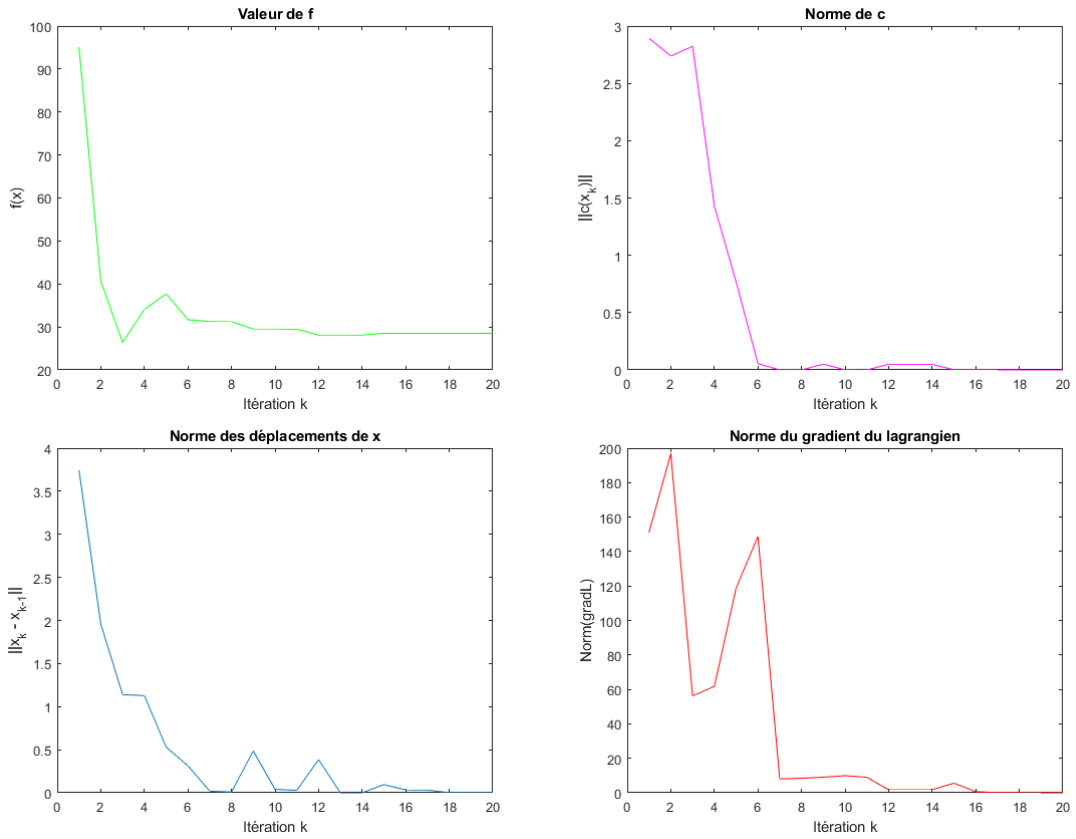
Déroulement de SQP avec la formule SR1

k	nfonc	x_k	$f(x_k)$	$c(x_k)$	λ_k	$\|\nabla L(x_k, \lambda_k)\|$
1	14	(-1.0000,2.0000,1.0000,-2.000,-2.000)	95.0000	(-2.2426,-1.8284,0)	(0,0,0)	151.2283
2	26	(-0.8613,1.6427,1.6803,-0.2249,-2.2774)	40.6578	(-1.5820 ,-2.2342,-0.0385)	(-12.1902,51.1953,-8.8761)	196.7853
3	35	(-1.2550,1.6765,1.7939,0,-1.2391)	26.3902	(-1.4691,-2.3699,-0.4448)	(-6.1072,15.8857,-9.7082)	56.2358
4	44	(-0.8909,2.1877,1.4193,-0.6701,-1.7759)	34.0600	(-0.3329,-1.3252,-0.4179)	(-0.7201,-5.7911,-9.3274)	61.7659
5	52	(-0.8689,2.3936,1.1529,-0.9696,-2.0543)	37.7243	(-0.0531,-0.7336,-0.2151)	(3.2899,-26.5737,-7.2905)	118.8906
⋮	⋮	⋮	⋮	⋮	⋮	⋮
16	130	(1.2272,2.4650,1.1810,-0.2426,-1.6293)	28.5137	(0.0010,-0.0008,-0.0006)	(2.5542,0.2747,-8.8615)	0.6413
17	138	(-1.2365,2.4622,1.1903,-0.2171,-1.6174)	28.5080	(0.0001,-0.0001,-0.0001)	(-2.5017,0.1107,-8.9297)	0.1399
18	146	(-1.2367,2.4621,1.1906,-0.2162,-1.6173)	28.5087	(0.0000,-0.0000,-0.0000)	(-2.5130,0.1277,-8.8948)	0.0073
19	154	(-1.2369,2.4620,1.1907,-0.2158,-1.6170)	28.5087	(0.0000,-0.0000,-0.0000)	(-2.5125,0.1250,-8.8961)	0.0018
20	160	(-1.2369,2.4620,1.1908,-0.2157,-1.6169)	28.5087	(0.0000,-0.0000,-0.0000)	(-2.5124,0.1247,-8.8962)	0.0003

L'algorithme converge en **20 itérations**, et **160 évaluations du problème**, légèrement moins performant que celui avec **BFGS**. Nous observons une bonne descente du gradient du lagrangien ici également. Voici la solution finale qu'on a obtenue, ainsi que la valeur finale de la fonction f, la valeur finale des contraintes et une comparaison des performances avec la solution donnée dans le polycopié du projet, stockée dans **sol_poly**.

```
sol = [-1.236930; 2.462044; 1.190759; -0.215709; -1.616907]
f(sol) = 28.508725
c(sol) = [0.000000; -0.000000; -0.000000]
sol_poly = [-1.236600; 2.461600; 1.191100; -0.214400; -1.616500]
f(sol_poly) = 28.496892
c(sol_poly) = [-0.001047; 0.000054; -0.001036]

sol - sol_poly =  [-0.000330; 0.000444; -0.000341; -0.001309; -0.000407]
f - f_poly = 0.011832
c - c poly = [0.001047; -0.000054; 0.001036]
```

Ensuite, voici les mêmes graphiques pour cette formule également.



En remarquant les pics, nous observons qu'avec cette formule on a une pire stabilité dans les déplacements de x et dans la norme du gradient. Cependant, l'algorithme converge bien et la solution est d'une précision satisfaisante.

2.2 Cas test 2 : Ariane1

Pour ce problème, les arguments d'entrée sont :

x_init	[130000 ; 30000 ; 9000]
lambda_init	0
problem	ariane1
choix	BFGS, puis SR1
bornes	[100000, 150000 ; 20000, 50000 ; 5000, 10000]
max_iter	1e3
max_eval	1e7
rho	8
eps	1e-4
max_rho	2^{20}

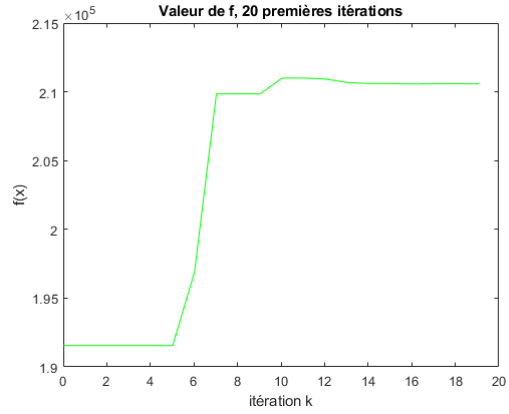
Déroulement de SQP avec la formule BFGS

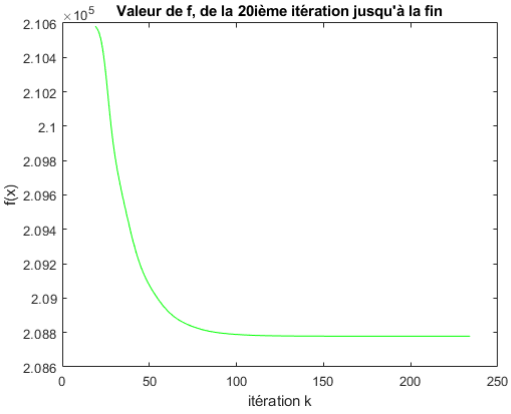
k	nfonc	$x_k \times 10^5$	$f(x_k) \times 10^5$	$c(x_k)$	$\lambda_k \times 10^5$	$\|\nabla L(x_k, \lambda_k)\|$
1	6	(1.3000,0.3000,0.0900)	1.9155	-165.8463	0	2.009823
2	7	(1.3000,0.3000,0.0900)	1.9155	-165.8463	-5.8623	2.009823
3	8	(1.3000,0.3000,0.0900)	1.9155	-165.8463	-5.8623	2.009823
4	9	(1.3000,0.3000,0.0900)	1.9155	-165.8463	-5.8623	2.009823
5	10	(1.3000,0.3000,0.0900)	1.9155	-165.8463	-5.8623	2.009823
⋮	⋮	⋮	⋮	⋮	⋮	⋮
230	1347	(1.4539,0.3124,0.0794)	2.0878	-0.0000	-0.0012	0.000661
231	1353	((1.4539,0.3124,0.0794)	2.0878	-0.0000	-0.0012	0.000629
232	1359	(1.4539,0.3124,0.0794)	2.0878	-0.0000	-0.0012	0.000599
233	1365	(1.4539,0.3124,0.0794)	2.0878	-0.0000	-0.0012	0.000562
234	1369	(1.4539,0.3124,0.0794)	2.0878	-0.0000	-0.0012	0.000524

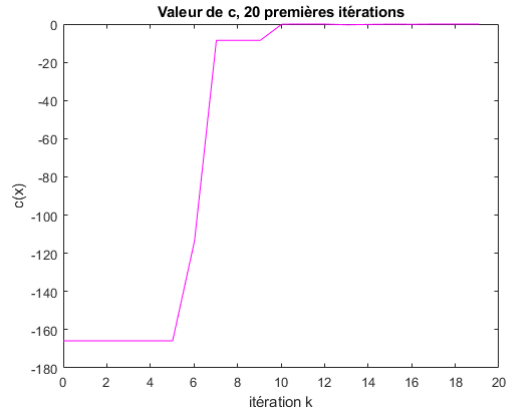
L'algorithme converge en **234 itérations**, et **1369 évaluations du problème**, nous observons une convergence plus lente que celle pour **MHW4D**. Voici la solution finale qu'on a obtenue, ainsi que la valeur finale de la fonction f, la valeur finale des contraintes et une comparaison des performances avec la solution donnée dans le polycopié du projet, stockée dans **sol_ariane**.

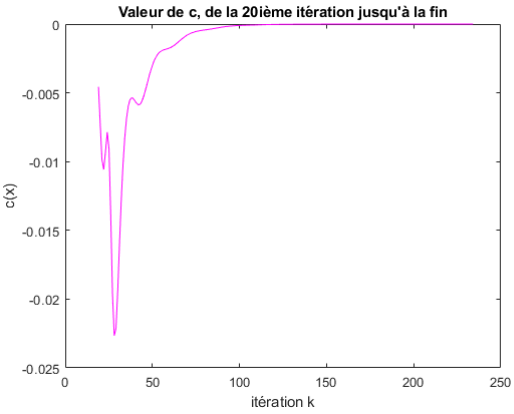
```
sol = [145394.950101; 31243.500773; 7936.194491]
f(sol) = 208778.589983
c(sol) = -0.000000

sol_ariane = [145349.000000; 31215.000000; 7933.000000]
f(sol_ariane) = 208690.831100
c(sol_ariane) = -0.749430

f - f_ariane = 87.758883
c - c_ariane = 0.749430
sol - sol_ariane = [45.950101; 28.500773; 3.194491]
```

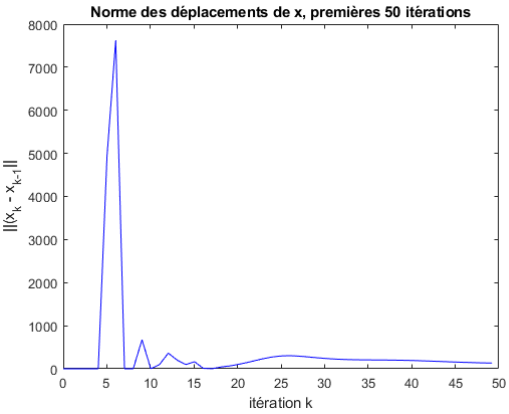
Pour une meilleure visibilité de ces résultats, nous avons affiché les valeurs caractéristiques d'abord pour les 20 premières itérations (50 pour le déplacement de x), et ensuite pour toutes les autres itérations jusqu'à la fin.

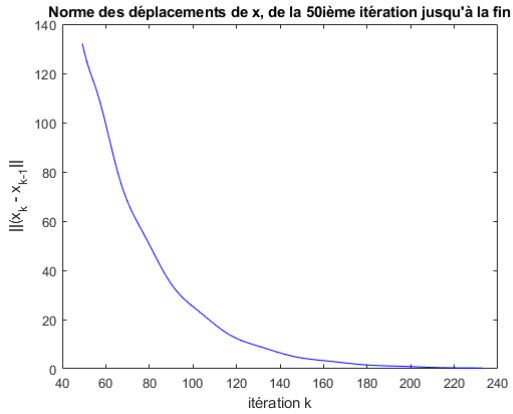


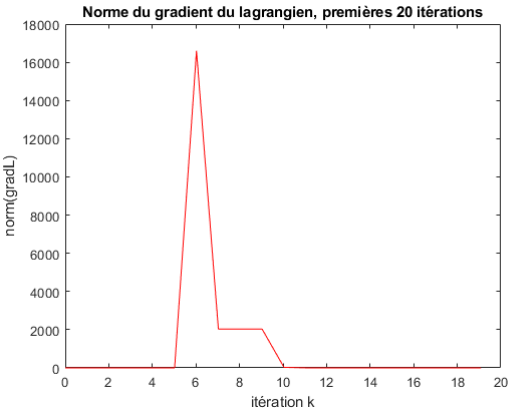


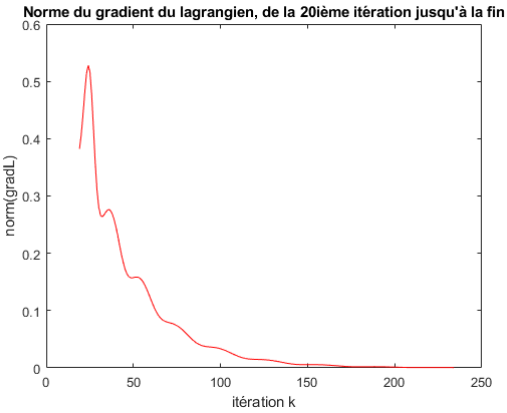












On remarque qu'il y a une descente assez rapide dans les 20 premières itérations, et ensuite on converge plus lentement vers la bonne solution.

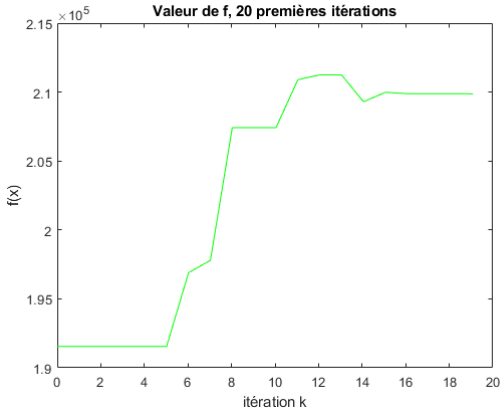
Déroulement de SQP avec la formule SR1

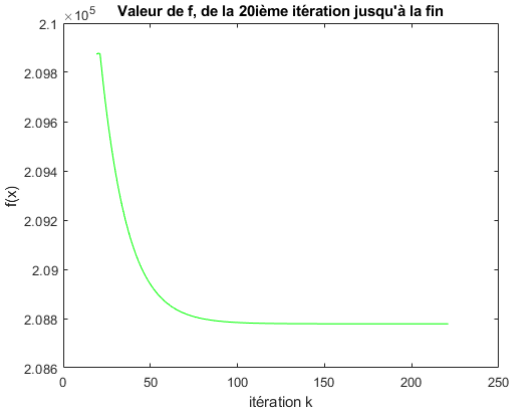
k	nfonc	$x_k \times 10^5$	$f(x_k) \times 10^5$	$c(x_k)$	$\lambda_k \times 10^6$	$\|\nabla L(x_k, \lambda_k)\|$
1	6	(1.3000,0.3000,0.0900)	1.9155	-165.8463	0	2.009823
2	7	(1.3000,0.3000,0.0900)	1.9155	-165.8463	-0.5862	2.009823
3	8	(1.3000,0.3000,0.0900)	1.9155	-165.8463	-0.5862	2.009823
4	9	(1.3000,0.3000,0.0900)	1.9155	-165.8463	-0.5862	2.009823
5	10	(1.3000,0.3000,0.0900)	1.9155	-165.8463	-0.5862	2.009823
⋮	⋮	⋮	⋮	⋮	⋮	⋮
217	1273	(1.4539,0.3124,0.0794)	2.0878	-0.0000	-0.0001	0.000360
218	1279	((1.4539,0.3124,0.0794)	2.0878	-0.0000	-0.0001	0.000348
219	1285	(1.4539,0.3124,0.0794)	2.0878	-0.0000	-0.0001	0.000338
220	1291	(1.4539,0.3124,0.0794)	2.0878	-0.0000	-0.0001	0.000326
221	1295	(1.4539,0.3124,0.0794)	2.0878	-0.0000	-0.0001	0.000314

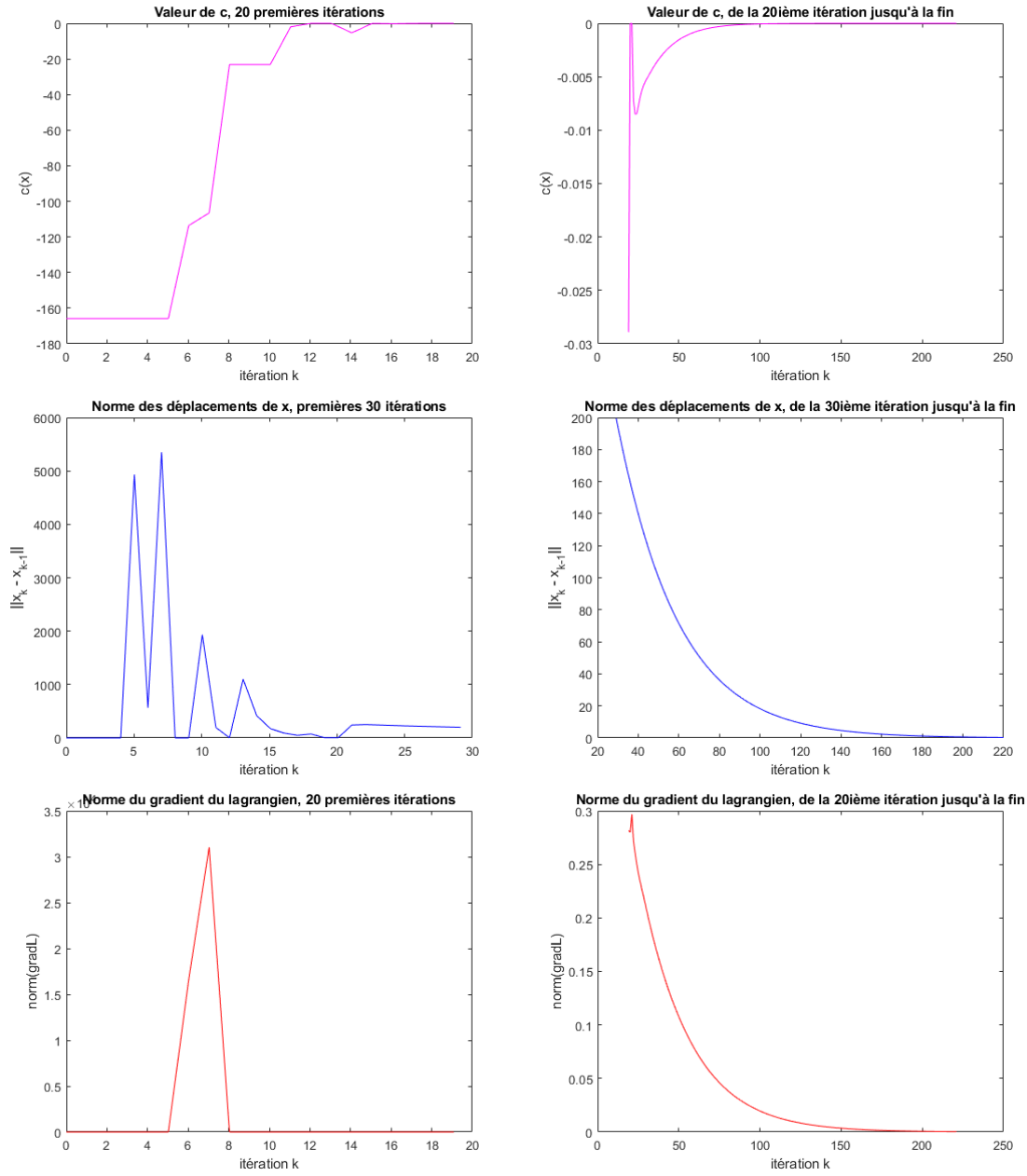
L'algorithme converge en **221 itérations**, et **1295 évaluations du problème**, nous observons une convergence légèrement plus rapide que l'algorithme avec **BFGS**. Voici la solution finale qu'on a obtenue, ainsi que la valeur finale de la fonction f, la valeur finale des contraintes et une comparaison des performances avec la solution donnée dans le polycopié du projet, stockée dans **sol_ariane**.

```
sol = [145394.392361; 31244.230662; 7936.011534]
f(sol) = 208778.590177
c(sol) = -0.000000

sol_ariane = [145349.000000; 31215.000000; 7933.000000]
f(sol_ariane) = 208690.831100
c(sol_ariane) = -0.749430

f - f_ariane = 87.759077
c - c_ariane = 0.749430
sol - sol_ariane = [45.392361; 29.230662; 3.011534]
```

Pour une meilleure visibilité de ces résultats, nous avons affiché les valeurs caractéristiques d'abord pour les 20 premières itérations (30 pour le déplacement de x), et ensuite pour toutes les autres itérations jusqu'à la fin.







Nous observons des fortes variations dans les premières 20 itérations de l'algorithme, mais ensuite on a une descente plus lisse que celle avec **BFGS**.

En conclusion, notre algorithme fonctionne correctement et on peut l'utiliser pour notre mission par la suite.

3 Étagement

3.1 Formulation du problème d'étagement

Afin de réduire la masse totale du lanceur, on répartit les ergols dans 3 étages propulsifs, qui fonctionnent successivement. Lorsque tous les ergols d'un étage ont été brûlés, l'étage vide (qu'on appelle masse sèche par la suite) est largué, ce qui permet d'alléger le lanceur avant d'allumer l'étage suivant.
Le problème d'étagement détermine la configuration pour une vitesse propulsive V_p donnée et une masse du satellite m_u donnée, en utilisant un modèle simplifié du lanceur.

$$\max_{m_{e1},m_{e2},m_{e3}} J = \frac{m_u}{M_0} \quad \text{sous} \quad V = V_p \tag{PE}$$

Le rapport J est le critère de performance du lanceur (généralement de l'ordre de 1%).
La vitesse propulsive V est la somme des contributions des 3 étages :

$$V = v_{e1} \log \frac{M_{i1}}{M_{f1}} + v_{e2} \log \frac{M_{i2}}{M_{f2}} + v_{e3} \log \frac{M_{i3}}{M_{f3}}$$

Si la combustion était instantanée, la vitesse en fin de vol serait cette vitesse propulsive. Mais comme cela n'est pas le cas, il y aura des pertes de vitesse au cours de la trajectoire dues aux autres forces (gravité, traînée) et au non-alignement de la poussée avec la vitesse. Il faudrait donc dimensionner le lanceur sur une vitesse propulsive supérieure à celle à atteindre sur l'orbite. D'où la nécessité de simuler les trajectoires, en itérant les configurations, pour pouvoir bien ajuster la vitesse propulsive et trouver une configuration optimale.

***Remarque** : Ariane1 était en effet, un problème d'étagement, de vitesse propulsive donnée : $V =$ 11527 m/s, et de masse de satellite : $m_u = 1700kg$.*

3.2 Résolution analytique du problème d'étagement

Étape 1 : Reformulation du problème (PE)

Nous allons reformuler le problème (PE) de la façon suivante. Posons :

$$x_j = \frac{M_{i,j}}{M_{f,j}} \tag{*}$$

Réécriture de la contrainte :

$$c(x_1, x_2, x_3) = v_{e_1} \times \ln(x_1) + v_{e_2} \times \ln(x_2) + v_{e_3} \times \ln(x_3) - V_p \tag{1}$$

Comme $m_u = M_{i,4}$ et $M_0 = M_{i,1}$, on réécrit le critère de performance :

$$J = \frac{m_u}{M_0} = \frac{M_{i,4}}{M_{i,1}} = \frac{M_{i,4}}{M_{i,3}} \times \frac{M_{i,3}}{M_{i,2}} \times \frac{M_{i,2}}{M_{i,1}}$$

On a de plus $M_{f,j} = M_{i,j} - m_{e,j}$, $M_{i,j+1} = M_{f,j} - m_{s,j}$ et $m_{s,j} = k_j \times m_{e,j}$, donc on obtient :

$$\frac{M_{i,j+1}}{M_{i,j}} = \frac{M_{f,j} - m_{s,j}}{M_{i,j}} = \frac{M_{f,j} - k_j \times m_{e,j}}{M_{i,j}} \tag{2}$$

En utilisant (*) dans (2) on obtient :

$$\frac{M_{i,j+1}}{M_{i,j}} = \frac{1}{x_j} - \frac{k_j \times m_{e,j}}{M_{i,j}}$$

et l'on a aussi $m_{ej} = M_{i,j} - M_{f,j}$ et donc :

$$\frac{M_{i,j+1}}{M_{i,j}} = \frac{1}{x_j} - k_j \frac{M_{i,j} - M_{f,j}}{M_{i,j}} = \frac{1}{x_j} - k_j(1 - \frac{1}{x_j}) = \frac{1+k_j}{x_j} - k_j \tag{3}$$

Le critère de performance se réécrit donc :

$$J = (\frac{1+k_1}{x_1} - k_1)(\frac{1+k_2}{x_2} - k_2)(\frac{1+k_3}{x_3} - k_3) \tag{4}$$

On a bien $\max f = \min -f$, le résultat voulu.

Étape 2 : Conditions nécessaires d'optimalité d'ordre 1, KKT

En posant $y_j = \frac{1+k_j}{x_j} - k_j$, on obtient une réécriture de f de la façon suivante :

$$f(y_1, y_2, y_3) = -y_1 y_2 y_3 \tag{5}$$

Ce qui nous permet d'écrire le gradient de f de la façon suivante :

$$\partial_{x_j} f = -f \frac{1+k_j}{y_j x_j^2} \tag{6}$$

D'où, on obtient les conditions **KKT** pour le problème (PE) :

$$\begin{cases} \partial^*_{x_j} L(x^*, \lambda^*) & = -f \frac{1+k_j}{y_j((x_j)^*)^2} + \lambda \frac{v_{ej}}{x_j^*} \\ \sum_{j=1}^{3} v_{ej} \log(x_j) - V_p & = 0 \end{cases} \tag{7}$$

En utilisant la première équation du système **KKT** on obtient :

$$\begin{aligned} \lambda \frac{v_{ej}}{x_j} &= f \frac{1+k_j}{y_j x_j^2} \\ \frac{\lambda v_{ej}(1 + k_j - k_j x_j)}{1+k_j} &= f \\ v_{e_j}(1 - \frac{k_j}{1+k_j} x_j) &= \frac{f}{\lambda} \\ v_{e_j}(1 - \Omega_j x_j) &= \frac{f}{\lambda} \end{aligned}$$

$\frac{f}{\lambda}$ étant une constante, on a donc montré que $v_{ej}(1 - \Omega_j x_j) = \text{cte}$, avec $\Omega_j = \frac{k_j}{1+k_j}$.

Étape 3 : Équation d'une inconnue

En utilisant le résultat précédent, particulièrement en utilisant le fait que :

$$v_{e1}(1 - \Omega_1 x_1) = v_{e2}(1 - \Omega_2 x_2) = v_{e3}(1 - \Omega_3 x_3) = \frac{f}{\lambda} \tag{8}$$

on déduit que le problème se ramène à résoudre une équation à une inconnue.

On choisit x_3 comme inconnue et on exprime x_1 et x_2 en fonction de x_3 :

$$x_1 = \frac{v_{e3}\Omega_3}{v_{e1}\Omega_1}x_3 + \frac{v_{e1} - v_{e3}}{v_{e1}\Omega_1}$$
$$x_2 = \frac{v_{e3}\Omega_3}{v_{e2}\Omega_2}x_3 + \frac{v_{e2} - v_{e3}}{v_{e2}\Omega_2}$$

Pour nous simplifier la vie, posons :

$$x_3 = x$$
$$v_{ej}\Omega_j = \eta_j$$
$$v_{e1} - v_{e3} = \delta_{13}$$
$$v_{e2} - v_{e3} = \delta_{23}$$

Ainsi, on peut finalement réécrire la contrainte (1), comme une équation en x de la façon suivante :

$$c(x) = v_{e1}\log(\eta_3 x + \delta_{13}) - v_{e1}\log(\eta_1) + v_{e2}\log(\eta_3 x + \delta_{23}) - v_{e2}\log(\eta_2) + v_{e3}\log(x) - V_p \qquad (9)$$

La dérivée de cette fonction est donnée par :

$$c'(x) = \frac{v_{e1}\eta_3}{\eta_3 x + \delta_{13}} + \frac{v_{e2}\eta_3}{\eta_3 x + \delta_{23}} + \frac{v_{e3}}{x} \qquad (10)$$

3.2.1 Application à Ariane1

Pour ce problème, nous disposons des valeurs numériques suivantes :

étage	**indice constructif** k	**vitesse d'éjection** v_e
1	0.1101	2647.2
2	0.1532	2922.4
3	0.2154	4344.3

Vitesse propulsive : $V_p = 11527$ m/s
Masse du satellite : $m_u = 1700$ kg

En utilisant la **méthode de Newton** pour trouver la solution de l'équation (9) on obtient :

$$x^* = 3.327601$$

Puis, en remontant dans nos raisonnements pour obtenir m_{e1}, m_{e2} et m_{e3} on obtient :

$$m_{e1} = 145400.850019$$
$$m_{e2} = 31238.539508$$
$$m_{e3} = 7935.512148$$

Nous comparons ces résultats aux valeurs données dans le polycopié, ce qui nous permet de nous assurer que notre équation est bonne.

	solution analytique	**solution polycopié**	**écart**
m_{e1}	145400.850019	145349.000000	51.850019
m_{e2}	31238.539508	31215.000000	23.539508
m_{e3}	7935.512148	7933.000000	2.512148

En utilisant les valeurs obtenues pour x_1, x_2 et x_3 on obtient les valeurs de y_1, y_2 et y_3, ce qui nous permet d'obtenir la valeur de f en les injectant dans (5), et on obtient le critère de performance J en utilisant le fait que $\max J = \min f$. Nous obtenons :

$$J = 0.0081 = 0.81\%$$

La valeur est proche de 1%, comme attendu. Calculons le multiplicateur de Lagrange, λ, en utilisant (8). Voici le résultat obtenu.

$$\lambda = -2.0710 \times 10^{-6}$$

Vérifions que les conditions **KKT** d'ordre 1 sont satisfaites :

$$\begin{cases} \nabla_x^* L(x^*, \lambda^*) & = (0.0020, 0.0025, 0.0033) \\ \sum_{j=1}^{3} v_{ej} \log(x_j) - V_p & = 3.6380 \times 10^{-12} \end{cases} \tag{11}$$

Les conditions **KKT** d'optimalité d'ordre 1 sont bien satisfaites.
Comparons cette solution analytique avec celle qu'on a obtenue avec SQP.

	solution analytique	**solution SQP**	**écart**
m_{e1}	145400.850019	145394.392361	6.4577
m_{e2}	31238.539508	31244.230662	-5.6912
m_{e3}	7935.512148	7936.011534	-0.4994

On peut confirmer la validité de notre algorithme SQP, car nous observons un écart très petit entre la solution analytique et la solution obtenue par SQP, vu l'ordre de grandeur des valeurs.

4 Trajectoire

4.1 Formulation du problème de trajectoire

Pour une configuration des masses d'ergols et une vitesse propulsive données, la trajectoire du lanceur est calculée en intégrant numériquement les équations différentielles du mouvement à l'aide de l'intégrateur numérique **ODE45** de Matlab. En raison des discontinuités de masse à la séparation des étages, des changements de poussée et d'angle de commande à chaque étage, il faut réaliser l'intégration par 3 appels successifs à l'intégrateur correspondant aux 3 étages.
Entre chaque appel, la masse sèche de l'étage précédent est larguée, la poussée, le débit et l'angle de commande sont actualisés pour l'étage suivant.

On cherche à atteindre l'altitude de l'orbite en maximisant la vitesse fournie au satellite.

$$\max_{\theta_0, \theta_1, \theta_2, \theta_3} V(t_f) \quad \text{sous} \quad \begin{cases} R(t_f) = R_c \\ \overrightarrow{R}(t_f).\overrightarrow{V}(t_f) = 0 \end{cases} \tag{PT}$$

La résolution de ce problème permet de connaître la vitesse maximale V_r atteignable par le lanceur et de vérifier si la configuration du lanceur est adaptée à la mission. Si la vitesse réelle V_r est inférieure à celle à atteindre, alors le lanceur est sous-performant et il faut augmenter sa vitesse propulsive. Si la vitesse réelle V_r est supérieure à la vitesse cible, alors le lanceur est sur-performant et il faut diminuer sa vitesse propulsive. Donc, on itère sur la vitesse propulsive jusqu'à ce que la vitesse atteinte à la fin coïncide avec la vitesse cible V_c.

4.2 Résolution du problème de trajectoire

Pour résoudre ce problème, on va brancher notre optimiseur **SQP** au simulateur de trajectoire nommée **simulation_trajectoire**. C'est-à-dire, pour une configuration des masses du lanceur donnée, on simule une trajectoire, on garde les conditions finales, et puis on lance **SQP** avec le problème (PT) à résoudre, en simulant une trajectoire à chaque itération.

Montrons cette résolution et ce branchement sur notre problème directement.

5 Notre problème : Ariane6

5.1 Données du problème

Comme dit tout au début, notre mission est d'amener un satellite de 2000kg en orbite, à une altitude de 250km. Voici plus précisément les données de notre problème, qu'on a nommé **Ariane6**.

Hauteur cible : $H_c = 250000$ m
Vitesse cible : $V_c = 7754.84120$ m/s
Rayon de l'orbite : $R_c = 6628137$ m
Masse du satellite : $m_u = 2000$ kg

étage	**indice constructif** k	**vitesse d'éjection** v_e	**accélération initiale** α_e
1	0.10	2600	15
2	0.15	3000	10
3	0.20	4400	10

5.2 Logiciel final

Initialisation

Tout d'abord on écrit les constantes utilisées : la constante gravitationnelle μ, le rayon terrestre R_{terre} et le coefficient de traînée du lanceur c_x.
Ces constantes sont suivies par les données du problème, écrites ci-dessus.

On a besoin de prendre une vitesse propulsive légèrement plus grande que la vitesse cible, pour pouvoir ajuster les pertes de vitesse (δV) dans la boucle qui suit. C'est pour cela que nous initialisons la vitesse propulsives et les pertes de la façon suivante, et en début des itérations on prendra $V_p = V_p + \delta V$:

$$V_p = V_c \qquad \delta V = 0.2 V_c$$

On choisit des bornes pour les valeurs des masses d'ergols et pour les angles θ_j :

Bornes pour m_e **:** [8000, 50000 ; 5000, 20000 ; 4000, 10000]
Bornes pour θ **:** [-90, 90 ; -90, 90 ; -90, 90 ; -90, 90]

On initialise le point de départ pour le problème d'étagement, pour l'algorithme SQP par :

m_e **initial :** [15000 ; 13000 ; 5000]

Pour la valeur initiale de θ on avait effectué un balayage, c'est-à-dire, on avait trouvé un bon point de départ de façon expérimentale, en faisant plusieurs simulations de trajectoire. On a trouvé :

$$\theta \textbf{ initial :} \quad [4.5\,;\,2\,;\,6.5\,;\,4]$$

Ensuite, on simule cette trajectoire initiale dans le logiciel final pour nous assurer qu'on a choisi un bon θ initial.

Itérations

Les itérations se font sur la vitesse propulsive, plus précisément nous commençons par :

$$V_p = V_p + \delta V$$

et le but c'est d'itérer jusqu'à ce que δV devienne nulle.

Appel à SQP pour le problème d'étagement

x_init	[15000 ; 13000 ; 5000]
lambda_init	0
problem	ariane6
choix	SR1
max_iter	1e4
max_eval	1e7
rho	2
eps	1e-4
max_rho	2^{12}

Nous avons testé cet appel avec **BFGS** et **SR1**, et nous avons choisi **SR1** car on a observé une meilleure performance sur ce problème. A chaque appel on utilise le même point de départ, idem, car la performance est meilleure ainsi.

Appel à SQP pour le problème de trajectoire

x_init	[4.5 ; 2 ; 6.5 ; 4] puis le θ optimal de l'appel précédent
lambda_init	[0 ; 0]
problem	branchement
choix	BFGS
max_iter	1e3
max_eval	1e7
rho	2
eps	1e-8
max_rho	2^{12}

Nous avons testé cet appel avec **BFGS** et **SR1**, et nous avons choisi **BFGS** car on a observé une meilleure performance sur ce problème. A chaque nouvel appel on part avec le point optimal trouvé dans l'itération précédente, ceci accélère la convergence considérablement.

Ensuite on simule la trajectoire obtenue avec ces paramètres. On garde la position et la vitesse finales. Finalement, on met δV à jour :

$$\delta V = V_c - \|\overrightarrow{V}(t_f)\| = V_c - V_r$$

Critères d'arrêt

L'algorithme s'arrête après avoir obtenue une précision satisfaisante pour δV et le produit scalaire des vecteurs de la position et de la vitesse.

$$\delta V < \varepsilon \qquad \overrightarrow{R}(t_f).\overrightarrow{V}(t_f) < \varepsilon$$

5.3 Résultats : configuration et trajectoire finales

Pour une précision de $\varepsilon = 10^{-3}$ pour δV et pour le produit scalaire $\overrightarrow{R}(t_f).\overrightarrow{V}(t_f)$ qui nous assure qu'on a bien une vitesse horizontale à la fin, le logiciel final converge en seulement **9 itérations**, en **1.784334 seconds**. Nous avons **3008 itérations au total pour le problème d'étagement** (ce qui fait en moyenne **334 itérations pour l'étagement par itération**) et **136 itérations au total pour le problème de trajectoire** (ce qui fait en moyenne **15 itérations pour la trajectoire par itération**).

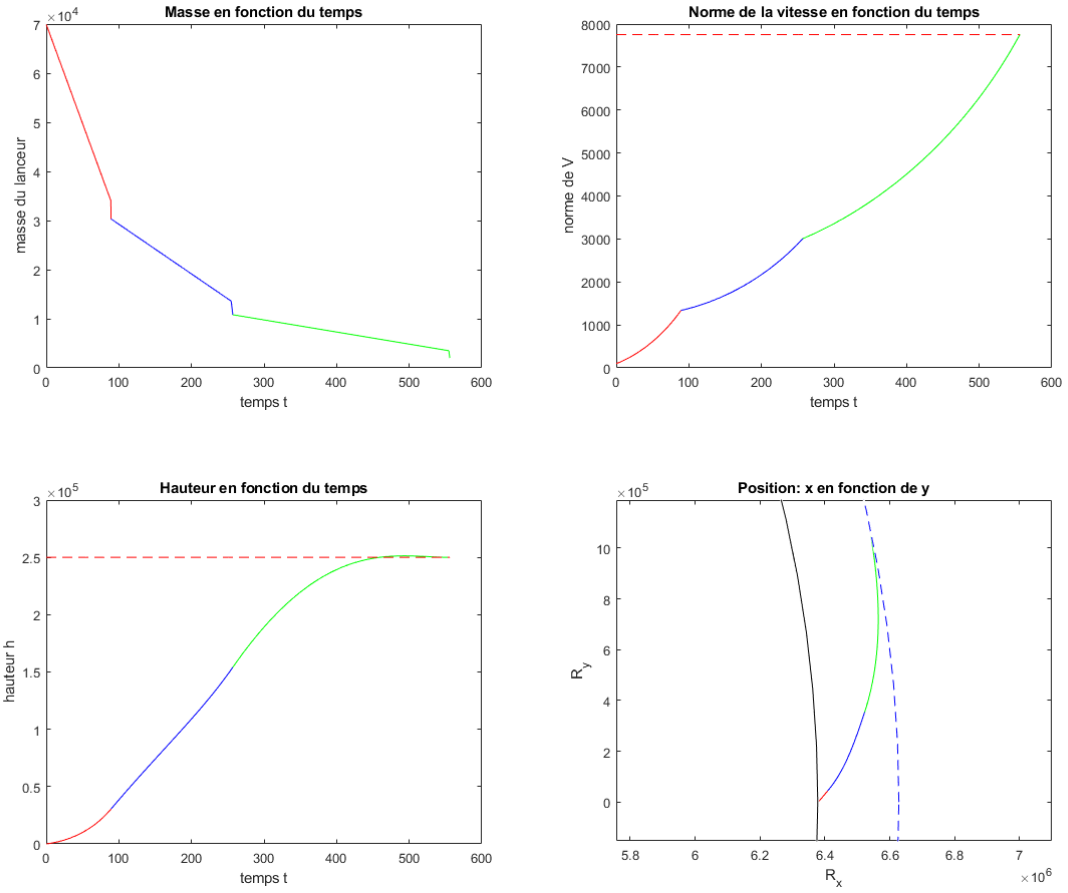
Dans le le tableau suivant, pour chaque itération, voici les valeurs des masses d'ergols, la masse totale du lanceur, la vitesse propulsive, les angles $\theta_0, \theta_1, \theta_2$ et θ_3, la vitesse réelle atteinte V_r et la différence δV entre cette vitesse réelle et la vitesse cible V_c.

k	(m_{e1}, m_{e2}, m_{e3})	M	V_p	$(\theta_0, \theta_1, \theta_2, \theta_3)$	V_r	δV
0	(15000, 13000, 5000)	39450.000000	9305.809440	(4.5, 2, 6.5, 4)	5181.191486	2573.649714
1	(35138.891430, 16758.820773, 7328.839819)	68720.032245	9305.809440	(37.545971, 18.810696,21.015677,4.413905)	7707.095662	47.745538
2	(36259.016867, 17015.191637, 7364.547889)	70289.846403	9353.554977	(37.815014, 18.892652,20.880846,4.257147)	7768.032243	-13.191043
3	(35946.341299, 16943.999298, 7354.734477)	69852.255993	9340.363934	(37.741407, 18.870425,20.918437,4.300938)	7751.214198	3.627001
4	(36031.398531, 16963.918877, 7357.694054)	69972.277957	9343.990936	(37.759794, 18.875686,20.908435,4.289103)	7755.815231	-0.974031
5	(36008.708370, 16958.489402, 7356.818214)	69940.023877	9343.016905	(37.757062, 18.875373,20.911177,4.292192)	7754.586540	0.254659
6	(36014.493481, 16959.954505, 7357.137195)	69948.455143	9343.271564	(37.757054, 18.875113,20.910497,4.291455)	7754.901200	-0.060000
7	(36013.499435, 16959.335748, 7356.985770)	69946.468413	9343.211564	(37.757175, 18.875197,20.910415,4.291604)	7754.838227	0.002972
8	(36013.565535, 16959.355202, 7356.988549)	69946.566830	9343.214536	(37.757258, 18.875226,20.910505,4.291579)	7754.841902	-0.000702
9	(36013.551351, 16959.349749, 7356.987405)	69946.543583	9343.213834	(37.757242, 18.875229,20.910408,4.291596)	7754.841084	0.000116

Dans le tableau suivant, nous montrons le nombre d'itérations exact de **SQP** pour les deux problèmes, d'étagement et de trajectoire, la différence entre le rayon de l'orbite atteinte et le rayon de l'orbite visé, le produit scalaire entre le vecteur de la position finale et le vecteur de la vitesse finale, et aussi δV encore une fois, pour meilleure visibilité de la convergence.

k	nb_iter étagement	nb_iter trajectoire	$\|\overrightarrow{R}(t_f)\| - R_c$	$\overrightarrow{R}(t_f).\overrightarrow{V}(t_f)$	δV
1	366	45	0.000173	3.053892	47.745538
2	319	21	-0.000000	-0.005407	-13.191043
3	330	21	-0.000001	-0.037703	3.627001
4	335	19	0.000160	4.034105	-0.974031
5	330	17	0.000002	0.057920	0.254659
6	335	4	-0.000001	-0.028374	-0.060000
7	331	3	0.000000	0.034998	0.002972
8	331	3	0.000001	0.038015	-0.000702
9	331	3	0.000000	-0.000004	0.000116

Enfin, voici les tracés de la masse du lanceur en fonction du temps, la norme de la vitesse en fonction du temps, la hauteur en fonction du temps et la trajectoire finale montrée par la position en fonction du temps.



Nous avons bien atteint la vitesse horizontale demandée et la hauteur demandée. De plus, la masse à la toute fin correspond bien à la masse du satellite.

5.4 Comparaison de nos résultats avec la solution analytique

Pour la vitesse propulsive obtenue, qui est $V_p = 9343.213834$, on cherche la solution analytique pour les masses d'ergols, et nous la comparons à la solution obtenue par l'algorithme **SQP** dans le logiciel final.

En utilisant les valeurs obtenues par la résolution analytique, on trouve le critère de performance J :

$$J = 0.0286 = 2.81\%$$

Voici le multiplicateur de Lagrange :

$$\lambda = -8.9359 \times 10^{-6}$$

Vérifions que les conditions **KKT** d'ordre 1 sont satisfaites.

$$\begin{cases} \nabla_x^* L(x^*, \lambda^*) & = (0.0058, 0.0061, 0.0065) \\ \sum_{j=1}^{3} v_{ej} \log(x_j) - V_p & = 5.4570 \times 10^{-12} \end{cases} \tag{12}$$

Les conditions **KKT** d'optimalité d'ordre 1 sont bien satisfaites. Et voici, finalement, la comparaison de cette solution analytique avec les résultats de notre logiciel final.

	solution analytique	**solution SQP**	**écart**
m_{e1}	36024.907836	36013.551351	11.356486
m_{e2}	16949.475067	16959.349749	-9.874682
m_{e3}	7356.038625	7356.987405	-0.948780

On observe que les résultats de notre algorithme SQP sont bien proches de celles de la solution analytique, car l'écart est très petit devant l'ordre de grandeur des valeurs.

Nous pouvons confirmer donc, que nos résultats sont corrects et que le logiciel final fonctionne correctement.

6 Conclusion

Nous avons bien réussi notre mission. Voici quelques remarques qu'on s'est faits pendant qu'on travaillait sur ce projet :

- Les algorithmes **SQP**, avec **BFGS** et avec **SR1** ne sont pas très robustes sur tout type de problème, et ils sont très fortement dépendants du point initial et des bornes pour les variables. Lorsqu'on part d'un mauvais point initial, il est possible que l'on n'ait pas de convergence et qu'on finit par augmenter ρ jusqu'à sa valeur maximale et sortir de l'algorithme car x ne se déplace plus à la fin de cette augmentation de ρ.

- Sans qu'on trouve une bonne valeur initiale pour θ "à la main", il est très difficile de faire converger l'algorithme **SQP** et trouver le θ optimal. Nous avons fait approximativement 50 simulations pour trouver ce bon point de départ.

- Pour la solution analytique, on cherche la bonne solution de l'équation (9) en utilisant la **méthode de Newton** et il est très important de choisir un bon point de départ également. Pour chaque problème nous avons utilisé la valeur 4, car la solution était près de 3. Dans chaque cas, si on choisissait un point de départ inférieur à 2, on obtenait des solutions complexes.