

Compte rendu de projet

Optimisation

I Algorithme SQP

I/ Explications sur le logiciel SQP

Arguments des fonctions

Notre logiciel SQP, qui est une fonction Matlab appelée « **algo_SQP.m** », prend en entrée :

- un **cell_array** (sorte de tableau pouvant contenir n'importe quel type d'éléments) **fct_pb = {@f, @c, @proj}** où f est la fonction à minimiser, c les contraintes associées au problème de minimisation et proj est la projection lorsqu'il y a des bornes sur x à respecter
- Le point de départ **x_0**
- **epsilon** la précision pour la condition d'arrêt
- **iter_max** le nombre maximal d'itérations pour la boucle while
- **nfonc_max** le nombre maximal d'appel de fonction
- **h** le pas utilisé pour le calcul du gradient par différences divisées
- **tau** utilisé pour obtenir une matrice définie positive à partir du Hessien du Lagrangien
- **choix** la variable qui dit si l'on choisit la méthode BFGS (choix=1) ou SR1 (choix=2)
- Le **lambda** de départ
- **eps_x** le déplacement minimal
- **rho_0** l'initialisation de la pénalisation de la fonction mérite
- **taille_tab** la taille du tableau tab de sortie

Et qui donne en sortie tab, le tableau contenant les informations demandées sur l'algorithme : les itérations **k**, le nombre d'appel de fonctions **nfonc**, chaque x_k , les valeurs $f(x_k)$ et $c(x_k)$, chaque λ_k ainsi que $\|\nabla L(x_k, \lambda_k)\|$.

A l'intérieur de la fonction algo_SQP.m, plusieurs appels à des fonctions sont effectués :

- « **grad_diff_div.m** » qui sert à calculer le gradient par méthode de différences divisées. Cette fonction prend en entrée : le x courant ; le pas h ; et le cell_array fct_pb = {@f, @c, @proj} et nfonc le nombre d'appel à la fonction f. En sortie, nous avons donc, le grad_f le gradient de f au point x ; jac_c le gradient de c au point x ; f_x la valeur f(x) ; c_x la valeur c(x) et nfonc.
- « **hessien_QN.m** » qui sert à calculer le hessien par la méthode de Quasi-Newton. Cette fonction prend en entrée : d_x le déplacement pour x ; grad_L_prec la valeur du gradient du Lagrangien de l'itération précédente ; grad_Lagrange la valeur du gradient du Lagrangien au point x courant ; H_Lagrange la valeur du Hessien du Lagrangien ; choix qui vaut 1 si on

choisit d'utiliser la méthode BFGS et qui vaut 2 si on choisit la méthode SR1 et glob_ind qui est un indicateur permettant de savoir si à l'issue de la globalisation, on réinitialise H_Lagrange à l'identité ou non. La sortie de cette fonction est la nouvelle valeur du Hessien du Lagrangien.

- « **mat_def_pos.m** » qui sert à rendre le Hessien du Lagrangien défini positif s'il ne l'est pas déjà. En entrée de cette fonction nous avons H_Lagrange le Hessien du Lagrangien ainsi que tau la valeur > 0 suffisamment grande pour avoir des valeurs propres positives. La sortie de cette fonction est la matrice définie positive associée au Hessien du Lagrangien.
- « **sol_pb_q.m** » qui sert à résoudre le problème quadratique. Cette fonction prend en entrée jac_c le gradient des contraintes ; Q la matrice définie positive correspondante au Hessien du Lagrangien ; grad_f le gradient de la fonction à minimiser et -c_x = -c(x). En sortie nous avons d_QP et lambda_QP respectivement les déplacements à appliquer à x et à lambda.
- « **globalisation.m** » qui effectue la globalisation de la fonction. Elle prend en entrée le point courant x ; fct_pb = {@f, @c, @proj} ; grad_f le gradient de la fonction à minimiser ; d_QP le déplacement pour x ; rho la pénalisation ; f_x = f(x) ; c_x = c(x) et glob_ind l'indice qui indique si rho est assez grand ou non, si non, on change la valeur du Hessien ou du rho selon les cas ; rho_0 la valeur initiale de la pénalisation et nfond le nombre d'appel à la fonction f. En sortie de cette fonction nous avons d_x le déplacement pour x ; rho ; glob_ind et nfond. A l'intérieur de globalisation, nous utilisons également :
 - o La fonction « **f_merite.m** » qui calcule la valeur de la fonction mérite. Elle prend en entrée le point x, le cell_array fct_pb ainsi que rho et elle donne en sortie la valeur de la fonction mérite au point x : y = F(x).

A toutes ces fonctions s'ajoutent nos fichiers de tests : « **algo_sqp_test.m** » contenant des tests sur la fonction page 24 des slides notamment ; « **test_global.m** » contenant les tests sur la fonction MHW4D (page 25) et « **test_ariane1.m** » contenant les tests sur la fonction Ariane 1 (page 26).

Tests d'arrêt

Dans notre « algo_SQP.m » nous utilisons une boucle while. Cette boucle while effectue des itérations tant que nous avons les conditions suivantes :

- $(\nabla L(x_k, \lambda_k)) > \epsilon$
- $\|d_x\| \geq \epsilon$
- nfond < nfond_max.
- Iter < iter_max

Ou bien : - $iter < iter_{max}$ et glob_ind $\neq 0$. Où, on rappelle que glob_ind = 0 lorsque la globalisation réussie.

Dans « globalisation.m » nous utilisons également une boucle while pour la règle d'Armijo. Cette boucle while s'arrête si au moins une des conditions suivantes est respectée :

- $|F(x + s \times d_{QP})| \leq F(x) + 0.1 \times s \times F'(x)$ où F est la fonction mérite et $s \times d_{QP}$ le pas de Newton et x appartient au domaine défini par la fonction projection du problème,
- $iter \geq iter_{max}$.

Traitement des échecs de globalisation

Si la globalisation échoue, c'est-à-dire si $F(x) \geq 0$, nous définissons le pas dx comme étant nul. De plus, si glob_ind vaut 0, alors il passe à 1 ce qui correspond au fait que nous réinitialisons le Hessien à l'identité lors de la prochaine itération. En revanche, si glob_ind vaut 1, il passe à 2 ce qui correspond à multiplier la pénalisation rho par 2. Nous avons de plus programmer une condition sur la valeur de rho, si elle devient trop grande, c'est-à-dire si la globalisation ne réussit jamais, l'algorithme s'arrête et renvoie une erreur.

II/ Test MHW4D

Méthode BFGS

Pour la méthode BFGS, l'algorithme trouve une solution en 12 itérations.

itération	nfond	x_k						f(x_k)	c(x_k)			lambda_k			grad_Lagrange
0	0	-1,0000	2,0000	1,0000	-2,0000	-2,0000	95,0000	-2,2426	-1,8284	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
1	12	-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	
2	25	-0,9688	1,6504	1,7121	-0,0908	-1,9938	34,4171	-1,5562	-2,2003	-0,0684	-5,5247	12,6289	-9,7787	46,1620	
3	33	-0,9611	2,1896	1,4173	-0,5394	-2,0451	34,0320	-0,4005	-1,1870	-0,0346	-2,0242	0,1030	-9,4105	28,7051	
4	44	-1,1726	2,1912	1,4983	-0,2373	-1,5972	28,8624	-0,3689	-1,1194	-0,1271	3,9242	-21,5250	-4,9209	101,2271	
5	52	-1,2918	2,3632	1,3481	-0,2998	-1,4890	29,0319	-0,1323	-0,5823	-0,0765	-1,8199	-2,1056	-8,5023	20,9350	
6	60	-1,3156	2,4354	1,2549	-0,3320	-1,4912	29,2242	-0,0523	-0,2998	-0,0382	-1,9923	-2,3887	-9,2759	18,6322	
7	67	-1,3056	2,4837	1,1780	-0,2735	-1,5316	28,8442	0,0082	-0,0059	-0,0004	-2,3230	-0,4672	-9,1237	6,8025	
8	74	-1,2545	2,4623	1,1980	-0,1991	-1,5918	28,4901	0,0009	-0,0004	-0,0031	-2,4584	0,2083	-8,9832	0,6515	
9	85	-1,2481	2,4619	1,1958	-0,2037	-1,6011	28,4982	0,0004	-0,0002	-0,0016	-2,4913	0,1213	-8,9421	0,2989	
10	94	-1,2453	2,4620	1,1946	-0,2066	-1,6050	28,5003	0,0003	-0,0002	-0,0012	-2,5101	0,1250	-8,9034	0,2314	
11	103	-1,2369	2,4620	1,1908	-0,2157	-1,6169	28,5079	0,0000	0,0000	-0,0001	-2,5121	0,1246	-8,8967	0,0017	
12	110	-1,2369	2,4620	1,1908	-0,2157	-1,6169	28,5087	0,0000	0,0000	0,0000	-2,5123	0,1247	-8,8963	0,0004	

Méthode SR1

Pour la méthode SR1, l'algorithme trouve la même solution en 19 itérations.

itération	nfond	x_k						f(x_k)	c(x_k)			lambda_k			grad_Lagrange
0	0	-1,0000	2,0000	1,0000	-2,0000	-2,0000	95,0000	-2,2426	-1,8284	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
1	12	-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	
2	25	-0,9597	1,6511	1,7087	-0,1030	-2,0178	34,8742	-1,5564	-2,2001	-0,0634	-6,1072	15,8857	-9,7082	58,4737	
3	33	-0,9540	2,1899	1,4141	-0,5486	-2,0629	34,2676	-0,4011	-1,1868	-0,0320	-1,4717	-2,0771	-9,3136	36,4787	
4	40	-1,0155	2,5562	1,0104	-0,8699	-1,9634	34,4431	0,2972	-0,1630	-0,0061	0,3786	-11,8575	-7,7085	55,6534	
5	47	-1,0686	2,5037	1,0226	-0,6297	-1,8668	30,0853	0,0029	-0,0001	-0,0051	-4,9502	7,1572	-8,4453	14,3315	
6	54	-1,1442	2,4648	1,1534	-0,3233	-1,7395	28,6552	0,0186	-0,0171	-0,0096	-2,7614	1,8122	-8,2864	3,4629	
7	61	-1,1859	2,4607	1,1720	-0,2590	-1,6846	28,5401	0,0004	-0,0003	-0,0023	-2,4420	0,2460	-8,6551	1,5623	
8	68	-1,2184	2,4619	1,1832	-0,2336	-1,6403	28,5025	0,0001	-0,0001	-0,0014	-2,5473	0,3953	-8,8724	0,6547	
9	78	-1,2306	2,4623	1,1877	-0,2233	-1,6251	28,5079	0,0000	0,0000	-0,0002	-2,5226	0,1622	-8,8839	0,1229	
10	86	-1,2322	2,4622	1,1884	-0,2215	-1,6230	28,5083	0,0000	0,0000	-0,0001	-2,5178	0,1451	-8,8881	0,0980	
11	95	-1,2340	2,4622	1,1892	-0,2195	-1,6208	28,5089	0,0000	0,0000	0,0000	-2,5178	0,1457	-8,8888	0,0581	
12	102	-1,2350	2,4621	1,1898	-0,2182	-1,6194	28,5088	0,0000	0,0000	0,0000	-2,5159	0,1385	-8,8909	0,0375	
13	109	-1,2357	2,4621	1,1901	-0,2173	-1,6185	28,5088	0,0000	0,0000	0,0000	-2,5147	0,1337	-8,8927	0,0243	
14	116	-1,2361	2,4621	1,1903	-0,2168	-1,6180	28,5087	0,0000	0,0000	0,0000	-2,5139	0,1307	-8,8939	0,0160	
15	123	-1,2364	2,4621	1,1905	-0,2164	-1,6176	28,5087	0,0000	0,0000	0,0000	-2,5134	0,1286	-8,8947	0,0105	
16	130	-1,2366	2,4621	1,1906	-0,2162	-1,6173	28,5087	0,0000	0,0000	0,0000	-2,5130	0,1273	-8,8952	0,0069	
17	137	-1,2367	2,4621	1,1906	-0,2160	-1,6172	28,5087	0,0000	0,0000	0,0000	-2,5128	0,1264	-8,8956	0,0045	
18	144	-1,2368	2,4621	1,1907	-0,2159	-1,6171	28,5087	0,0000	0,0000	0,0000	-2,5127	0,1258	-8,8958	0,0029	
19	151	-1,2368	2,4621	1,1907	-0,2158	-1,6170	28,5087	0,0000	0,0000	0,0000	-2,5126	0,1254	-8,8959	0,0019	

III/ Test Ariane 1

Méthode BFGS

Pour la méthode BFGS, l'algorithme trouve une solution en 30 itérations.

itération	nfond	x_k			f(x_k)	c(x_k)	lambda_k	grad_Lagrange
0	0	250000,00	50000,00	10000,00	349039,00	876,32	0,00	0,00
1	6	200103,04	11392,06	13450,72	253319,72	-126,95	18222234,11	862201,39
2	16	200449,25	13667,20	12071,23	254651,11	-4,53	-56717,02	2481,17
3	21	200462,72	13752,33	12013,84	254694,48	-0,01	-187,25	7,95
4	26	200461,69	13751,73	12012,28	254690,74	0,00	-9,62	1,96
5	34	196075,04	10562,62	5963,19	238791,38	-176,03	30,28	4,13
6	44	196261,79	10787,11	6131,62	239462,27	-154,63	-33,96	1,22
7	50	196797,76	12629,88	5410,48	241305,87	-91,66	-31,22	1,19
8	57	196996,28	12900,40	5558,11	242017,64	-69,32	-37,90	1,39
9	64	197638,88	13924,68	5889,09	244314,46	-4,23	-38,11	1,10
10	69	197681,09	13962,01	5943,79	244470,85	-0,05	-38,07	1,07
11	74	197679,49	13949,85	5956,80	244470,86	0,00	-37,93	1,06
12	79	197649,82	13850,71	6057,45	244445,92	-0,15	-37,90	1,03
13	84	197590,95	13748,37	6177,07	244407,94	-0,20	-37,94	1,01
14	89	197404,95	13578,01	6406,54	244283,92	-0,68	-38,01	1,01
15	94	196939,05	13382,42	6779,35	243994,27	-1,54	-38,26	1,06
16	99	195728,19	13225,06	7430,50	243260,03	-3,89	-38,90	1,26
17	104	192826,73	13404,62	8520,13	241570,54	-8,87	-40,71	1,64
18	109	186688,36	14634,36	10125,69	238125,86	-18,00	-45,18	2,18
19	114	176508,08	17859,62	11740,27	232506,47	-27,23	-52,91	2,75
20	119	161236,30	23898,27	12531,07	223478,16	-37,03	-59,91	3,12
21	124	138007,86	34390,00	11286,30	208278,45	-67,71	-65,59	2,71
22	129	127283,69	42484,08	7237,72	200787,00	-125,87	-82,24	2,17
23	134	132723,24	36049,03	9637,52	202321,24	-79,98	-97,31	1,97
24	141	141604,23	33818,79	8573,34	208314,72	-8,06	-102,45	0,93
25	146	142205,35	34174,19	8270,60	209023,92	-0,83	-178,48	0,75
26	151	144195,82	32162,56	8084,01	208686,95	-1,15	-110,36	0,27
27	156	145576,62	31023,79	7933,09	208723,11	-0,49	-117,28	0,03
28	161	145395,67	31236,02	7941,18	208776,83	-0,02	-117,15	0,01
29	166	145399,10	31239,03	7936,63	208778,57	0,00	-117,14	0,00
30	171	145400,83	31238,56	7935,51	208778,59	0,00	-117,14	0,00

Méthode SR1

Pour la méthode SR1, l'algorithme trouve une solution en 119 itérations. Voici un résumé des itérations :

itération	nfonc	x_k			f(x_k)	c(x_k)	lambda_k	grad_Lagrange
0	0	250000,00	50000,00	10000,00	349039,00	876,32	0,00	0,00
1	6	200103,04	11392,06	13450,72	253319,72	-126,95	18222234,11	862201,39
2	16	200304,56	13655,33	12015,16	254408,64	-4,53	-65,48	3,09
3	21	200311,31	13742,06	11958,19	254446,92	-0,01	-12,09	1,96
4	26	200304,93	13737,52	11949,08	254423,52	0,00	-8,50	1,96
5	31	199307,19	13004,62	10545,41	250764,74	-5,77	-10,71	1,89
6	36	198426,18	12669,51	9052,57	247585,87	-7,84	-20,49	1,68
7	41	197801,58	12763,05	7725,94	245387,98	-8,76	-30,73	1,34

itération	nfonc	x_k			f(x_k)	c(x_k)	lambda_k	grad_Lagrange
112	566	148093,87	29034,80	7747,05	208997,70	-0,0026	-108,90	0,13
113	571	147997,70	29107,73	7753,44	208982,82	-0,0024	-109,21	0,12
114	576	147905,01	29178,41	7759,63	208968,94	-0,0022	-109,50	0,12
115	581	147815,67	29246,88	7765,61	208956,00	-0,0021	-109,78	0,11
116	586	147729,55	29313,21	7771,39	208943,92	-0,0019	-110,05	0,11
117	591	147646,55	29377,46	7776,98	208932,67	-0,0018	-110,31	0,11
118	596	147566,53	29439,68	7782,40	208922,17	-0,0017	-110,56	0,10
119	601	147489,41	29499,93	7787,62	208912,39	-0,0016	-110,80	0,10

IV/ Commentaires sur les tests

Choix des paramètres indépendamment des fonctions testées

Nous avons choisi nfonc_max pas trop grand pour que l'algorithme ne tourne pas trop longtemps. Le pas h est choisi à partir des tests que nous avons effectué sur grad_diff_div avec une fonction arbitraire en un point arbitraire.

Nous avons testé $h=10^{-n}$ avec n un entier allant de 1 à 15 pour voir quelle valeur donnait le résultat le plus proche de la dérivée exacte. 10^{-8} donnait le meilleur résultat. C'est donc celui-ci que nous avons conservé ici (en multipliant par l'ordre de grandeur du problème) car il nous donne de bons résultats. Il n'est peut-être pas le meilleur, il faudrait refaire des tests avec chaque fonction pour avoir le meilleur h.

Le choix du tau quant à lui est fait de telle façon que nous n'ayons jamais de matrice singulière donc plus grand qu'une certaine valeur, tout en restant assez petit pour garder en précision. Nous constatons ensuite que grad_Lagrange ne devient pas toujours plus petit qu'epsilon, d'où l'intérêt d'un iter_max dans la boucle while principal de notre algo_SQP.

Nous observons aussi que rho doit être initialisé pas trop grand (par exemple rho = 100 est trop grand) et laisser notre globalisation l'augmenter si besoin pour ne pas perdre en précision. Attention tout de même, si rho est trop petit, par exemple rho = 0.01 alors le nombre d'itération augmente et fait perdre en efficacité à notre algorithme.

Dans la globalisation, le nombre iter_max est petit car la boucle while s'arrêtera assez vite puisque le multiplicateur s diminue très rapidement.

Commentaires sur MHW4D

Nous observons les mêmes résultats avec les deux méthodes BFGS et SR1. Bien que la première soit plus rapide de quelques itérations. La solution f(x) trouvée est très proche du résultat du cours. Le gradient du Lagrangien et les contraintes sont proches de 0 et le lambda se stabilise. On peut considérer que l'objectif est atteint. De plus, tant que nous restons dans les bornes, l'algorithme

semble robuste, en effet, en testant avec plusieurs points situés dans les bornes imposées par les contraintes, l'algorithme converge à chaque fois bien que le nombre d'itérations varie.

Commentaires sur Ariane1

Nous observons que la méthode BFGS donne un résultat satisfaisant en très peu d'itérations. En effet, $f(x)$ est proche de celui attendu dans la consigne. Les contraintes sont très bien respectées et le gradient du Lagrangien proche de 0. Lambda semble se stabiliser au cours des itérations.

Pour SR1, nous obtenons un résultat satisfaisant mais plus d'itérations sont nécessaires. En une centaine d'itérations nous n'atteignons pas encore le point attendu. Le problème se règle toutefois si nous augmentons le nombre d'itérations à 200. Cependant il semble peu utile de le faire ici car les nombres considérés sont grands et que nous ne voulons pas faire trop d'appels de fonction.

Robustesse

Enfin, en ce qui concerne la robustesse, pour MHW4D, comme les bornes sont assez exigeantes, on pourrait penser que l'algorithme converge bien peu importe la valeur de départ tant que celle-ci se trouve dans les bornes de la consigne. C'est le cas. Notre point de départ étant $x_0 = [-1; 2; 1; -2; -2]$. En lui ajoutant ou soustrayant $[0.5 ; 0.5 ; 0.5 ; 0.5 ; 0.5]$, nous restons dans les bornes et nous avons un bon résultat en sortie. Le nombre d'itérations varie cependant selon le point de départ choisi. Pour ariane 1, n'ayant pas pris de bornes exigeantes pour contraintes, nous n'avons pas de problème.

II Solution Analytique

Nous résolvons le problème analytique avec les données suivantes : $\mu=1500 \text{ kg}$ à 250 km d'altitude.

On s'intéresse au problème de maximisation suivant :

$$\max_{m_{e1}, m_{e2}, m_{e3}} J = \frac{m_u}{M_0}$$

sous la condition $V = V_p$ où $V = v_{e1} \ln\left(\frac{M_{i1}}{M_{f1}}\right) + v_{e2} \ln\left(\frac{M_{i2}}{M_{f2}}\right) + v_{e3} \ln\left(\frac{M_{i3}}{M_{f3}}\right)$.

Nous allons reformuler ce problème sous la forme d'un problème de minimisation plus simple.

Posons d'abord $x_j = \frac{M_{ij}}{M_{fj}}$ alors :

$$V = v_{e1} \ln(x_1) + v_{e2} \ln(x_2) + v_{e3} \ln(x_3).$$

D'où $c(x) = v_{e1} \ln(x_1) + v_{e2} \ln(x_2) + v_{e3} \ln(x_3) - V_p$.

Maintenant, on peut écrire

$$J = \frac{m_u}{M_0} = \frac{M_{i4}}{M_{i1}} = \frac{M_{i4}}{M_{i3}} \times \frac{M_{i3}}{M_{f2}} \times \frac{M_{i2}}{M_{f1}}$$

Et aussi

$$\frac{M_{i,j+1}}{M_{i,j}} = \frac{M_{fj} - m_{sj}}{M_{ij}} = \frac{M_{fj} - k_j m_{ej}}{M_{ij}} = \frac{M_{fj} - k_j(M_{ij} - M_{fj})}{M_{ij}} = \frac{1}{x_j} - k_j - \frac{k_j}{x_j} = \frac{1 + k_j}{x_j} - k_j$$

On a donc

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

Or pour l'instant nous avons un problème de maximisation, on le transforme maintenant en problème de minimisation :

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

Ecrivons maintenant les conditions nécessaires d'optimalité d'ordre 1 (conditions KKT). Pour cela, commençons par calculer le lagrangien de notre fonction f ainsi que son gradient.

$$\ell(x, \lambda) = f(x) + \lambda c(x)$$

$$\nabla \ell(x, \lambda) = \nabla f(x) + \lambda \nabla c(x).$$

Posons ensuite $y_j = \frac{1+k_j}{x_j} - k_j$. Alors $f(x_1, x_2, x_3) = -y_1 y_2 y_3$.

Calculons ensuite les dérivées partielles de f et de c .

$$\frac{\partial f}{\partial x_k}(x) = \frac{1+k_k}{x_k^2} \prod_{j \neq k} \left(\frac{1+k_j}{x_j} - k_j \right) = \frac{y_k + k_k}{x_k} \prod_{j \neq k} y_j = \frac{f(y_1, y_2, y_3)}{x_k} + \frac{k_k}{x_k} \prod_{j \neq k} y_j$$

$$\frac{\partial c}{\partial x_k}(x) = \frac{v_{ek}}{x_k}$$

Cela nous donne donc les conditions de KKT :

$$\begin{cases} \frac{f(x^*)}{x_k^*} + \frac{k_k}{x_k^*} \prod_{j \neq k} y_j + \lambda^* \frac{v_{ek}}{x_k^*} = 0 \\ \lambda^* \times (v_{e1} \ln(x_1^*) + v_{e2} \ln(x_2^*) + v_{e3} \ln(x_3^*)) - V_p = 0 \end{cases}$$

Nous allons transformé l'expression de la première condition. On notera $f = f(x^*)$.

$$\begin{aligned} \frac{f}{x_k^*} + \frac{k_k}{x_k^*} \prod_{j \neq k} y_j + \lambda^* \frac{v_{ek}}{x_k^*} = 0 &\Leftrightarrow f - k_k \frac{f}{y_k^*} + \lambda^* \times v_{ek} = 0 \Leftrightarrow f \left(1 - \frac{k_k}{y_k^*} \right) = -\lambda^* \times v_{ek} \\ &\Leftrightarrow \frac{f}{\lambda^*} = v_{ej} \left(1 - x_j^* \frac{k_j}{1+k_j} \right) = v_{ej} (1 - x_j^* \Omega_{j-1}). \end{aligned}$$

Grâce à cette formule, nous pouvons maintenant réduire notre problème à une équation à une inconnue puisqu'on a

$$\begin{aligned} v_{ej} - x_j v_{ej} \Omega_j &= v_{ej-1} - x_{j-1} v_{ej-1} \Omega_{j-1} = v_{ej+1} - x_{j+1} v_{ej+1} \Omega_{j+1} \Leftrightarrow \\ x_j &= \frac{-v_{ej+1} (1 - x_{j+1} \Omega_{j+1}) + v_{ej}}{v_{ej} \Omega_j} \end{aligned}$$

On a ainsi la contrainte qui vaut

$$\begin{aligned} c(x_3) &= -V_p + v_{e1} \ln \left(\frac{-v_{e2} (1 - x_2 \Omega_2) + v_{e2}}{v_{e2} \Omega_2} \right) + v_{e2} \ln \left(\frac{-v_{e3} (1 - x_3 \Omega_3) + v_{e3}}{v_{e3} \Omega_3} \right) + v_{e3} \ln(x_3) \\ c(x_3) &= -V_p + v_{e1} \ln \left(-v_{e2} \left(1 - \frac{-v_{e3} (1 - x_3 \Omega_3) + v_{e3}}{v_{e2}} \right) + v_{e1} \right) + v_{e2} \ln \left(\frac{-v_{e3} (1 - x_3 \Omega_3) + v_{e3}}{v_{e3} \Omega_3} \right) \\ &\quad + v_{e3} \ln(x_3) \end{aligned}$$

Par une résolution de cette équation par la méthode de Newton, nous trouvons $x_3 = 2.9369$. Ceci nous donne ainsi la valeur de $x_2 = 1.9261$ et de $x_1 = 1.4964$. Ceci nous donne de plus, les valeurs de y_1, y_2, y_3 :

$$\begin{cases} y_1 = \frac{1+0.1}{1.4964} - 0.1 = 0.6351 = \frac{M_{i2}}{M_{i1}} \\ y_2 = \frac{1+0.15}{1.9261} - 0.15 = 0.4471 = \frac{M_{i3}}{M_{i2}} \\ y_3 = \frac{1+0.2}{2.9369} - 0.2 = 0.2086 = \frac{M_{i4}}{M_{i3}} \end{cases}$$

Mais on connaît $M_{i4} = m_u = 1500\text{kg}$. Donc on peut retrouver les valeurs des M_{ij} :

$$\begin{cases} M_{i3} = \frac{M_{i4}}{y_3} = \frac{1500}{0.2086} = 7190 \\ M_{i2} = \frac{M_{i3}}{y_2} = 16085 \\ M_{i1} = \frac{M_{i2}}{y_1} = 25327 \end{cases}$$

Et comme on sait que $x_j = \frac{M_{ij}}{M_{fi}}$ alors on peut retrouver la valeur des M_{fi} .

$$\begin{cases} M_{f3} = \frac{M_{i3}}{x_3} = \frac{7190}{2.9369} = 2448 \\ M_{f2} = \frac{M_{i2}}{x_2} = \frac{M_{i2}}{x_2} = \frac{16085}{1.9261} = 8350 \\ M_{f1} = \frac{M_{i1}}{x_1} = \frac{M_{i1}}{x_1} = \frac{25327}{1.4964} = 16925 \end{cases}$$

Enfin, on peut donner les m_{ej} :

$$\begin{cases} m_{e3} = M_{i3} - M_{f3} = 4724 \\ m_{e2} = M_{i2} - M_{f2} = 7734 \\ m_{e1} = M_{i1} - M_{f1} = 8402 \end{cases}$$

Nous obtenons donc $x = [1.4964, 1.9261, 2.9369]$. Ce qui nous donne par suite, les masses d'ergols $me = [8402, 7734, 4724]$. Notre programme « **simulateur_test.m** » quant à lui, nous donne $me = [8401, 7731, 4745]$ ce qui est très proche de la solution analytique (la différence est dû à une erreur d'arrondi). On peut donc valider le résultat.

III Simulateur de trajectoire

Le simulateur est programmé dans le fichier « **simulateur.m** », il calcule la trajectoire de notre lanceur. Ce dernier étant obtenu grâce à l'algorithme SQP. Le test de ce simulateur est réalisé par le fichier « **simulateur_test.m** » présenté dans III 3.

Avant d'aborder le simulateur et son test, nous présentons brièvement les fonctions auxiliaires utilisées.

1 Fonctions et fichiers auxiliaires

- « **fct_u.m** » : fonction qui renvoi le vecteur directeur de la poussée ;
- « **ode45_test.m** » : fichier utilisé brièvement pour tester la fonction Matlab ode45 en calculant une partie de la trajectoire ;
- « **ode_fun.m** » : fonction qui code le système d'équations à résoudre pour obtenir la trajectoire du lanceur, elle est utilisée avec la fonction Matlab ode45 ;
- « **recherche_lanceur.m** » : fichier utilisée brièvement pour déterminer l'initialisation de l'algorithme SQP et ses paramètres, il permet de tester automatiquement l'algorithme sur plusieurs points pour un jeu de paramètres donné ;
- « **tracer.m** » : fonction qui automatise les tracés de la trajectoire et des évolutions de la masse, la vitesse et l'altitude du lanceur.

Descriptif des entrées et sorties des fonctions

Fonctions	Entrées	Sorties
fct_u.m	R : position du lanceur V : vitesse du lanceur theta : angle de la vitesse initiale et incidences des poussées	u : vecteur directeur de la poussée
ode_fun.m.m	t : temps y : fonction inconnue (position, vitesse et masse du lanceur en fonction du temps) T_norm : normes des poussées v_e : vitesses d'éjection des ergols theta : angle de la vitesse initiale et incidences des poussées	dy : gradient de la fonction inconnue
tracer.m	RES : résultats de l'intégration de ode45 (approximation numérique de la fonction inconnue y) sep : temps des séparations des étages du lanceur T_norm : normes des poussées v_e : vitesses d'éjection des ergols theta : angle de la vitesse initiale et incidences des poussées	Pas de variables en sortie, la fonction sert uniquement à afficher les graphiques.

2 Fonction principale du simulateur

La simulation de la trajectoire de notre lanceur consiste à calculer une approximation numérique de la fonction y .

Cette fonction contient 5 composantes :

- y_1 et y_2 : composantes du vecteur position R ,
- y_3 et y_4 : composantes du vecteur vitesse V ,
- y_5 : masse.

Cette approximation numérique est réalisée par la fonction Matlab `ode45` pour résoudre les équations du mouvement appliquées à notre lanceur (cf. p.7 du sujet).

Le lanceur est composé de 3 étages stockant essentiellement du carburant (les ergols) et un étage contenant le satellite à mettre en orbite.

Durant le vol du lanceur, sa masse diminue en brûlant ses ergols mais aussi en se séparant des étages vides (après combustion de tout le carburant d'un étage). Ces séparations permettent d'alléger le lanceur, cependant elles nécessitent de résoudre les équations du mouvement par morceaux.

Le simulateur fonctionne selon les étapes suivantes :

- Initialisation des données du lanceur,
- Boucle for sur les étages contenant des ergols :
 - calcul des données pour l'étage,
 - intégration du morceau de trajectoire correspondant,
 - enregistrement des résultats,
 - mise à jour de la masse (séparation de l'étage vide) ;

Descriptif des entrées et sorties du simulateur.

Entrées	Sorties
theta : angle de la vitesse initiale et incidences des poussées M_0 : masse initiale du lanceur m_s : masses sèches (masses des étages vides) m_e : masses des ergols selon les étages	RES : résultats de l'intégration de <code>ode45</code> (approximation numérique de la fonction inconnue y) sep : temps des séparations des étages du lanceur T_norm : normes des poussées

3 Test du simulateur

La simulation est calculée avec les données de mission suivantes :

- masse utile $\mu = 1500 \text{ kg}$,
- altitude cible = 250 000m

Ces données nous ont amené à coder une version du problème Ariane1 permettant d'adapter μ et la vitesse propulsive V_p . Cette dernière dépend de l'altitude cible.

De plus les données de l'étagement (indices constructifs et vitesses d'éjection) diffèrent un peu de celles de la partie sur l'algorithme SQP.

Les fonctions de cette version adaptée de Ariane1 sont enregistrées dans le dossier `fct_Ariane`.

Dans un premier temps le simulateur calcule un étagement avec les données de notre mission. Nous réutilisons donc les paramètres du test sur Ariane1.

Un ajustement des paramètres h et τ (approximation des gradients et de la hessienne) a été réalisé en utilisant la fonction `recherche_lanceur.m`. Nous avons choisi les valeurs $(h, \tau) = (1e-12, 0.1)$ qui nous permettent d'observer une convergence de l'algorithme SQP (norme du gradient du lagrangien

inférieur à 5) pour 85 points sur 108 points testés.

Avec ces paramètres et $m_0 = [155000 ; 75000 ; 5000]$, nous obtenons un lanceur de 25 tonnes en 43 itérations avec la norme du gradient égale à 0.0021.

Les valeurs de theta sont ajustées manuellement en effectuant plusieurs essais du simulateur. Ce réglage se base sur l'observation des tracés de la trajectoire et de l'altitude en fonction du temps. Nous avons choisi $\theta = [0;0;-7.2; 8]$. Pour faciliter ce travail, les vecteurs poussées ont été ajoutés au graphique de la trajectoire.

IV Résultats des itérations successives étagement-trajectoire

Notre vitesse cible V_c est égale à 7 754,84 m/s, voici les résultats obtenus avec notre algorithme :

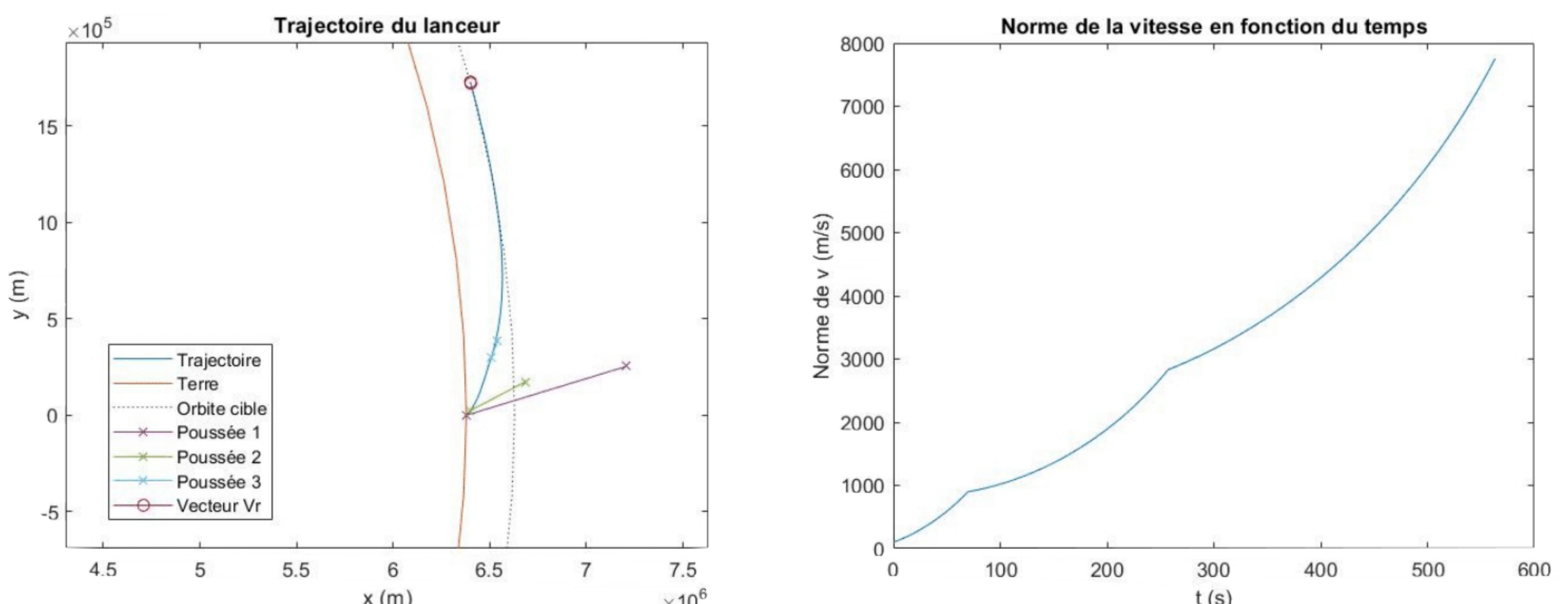
Résultats obtenus avec BFGS :

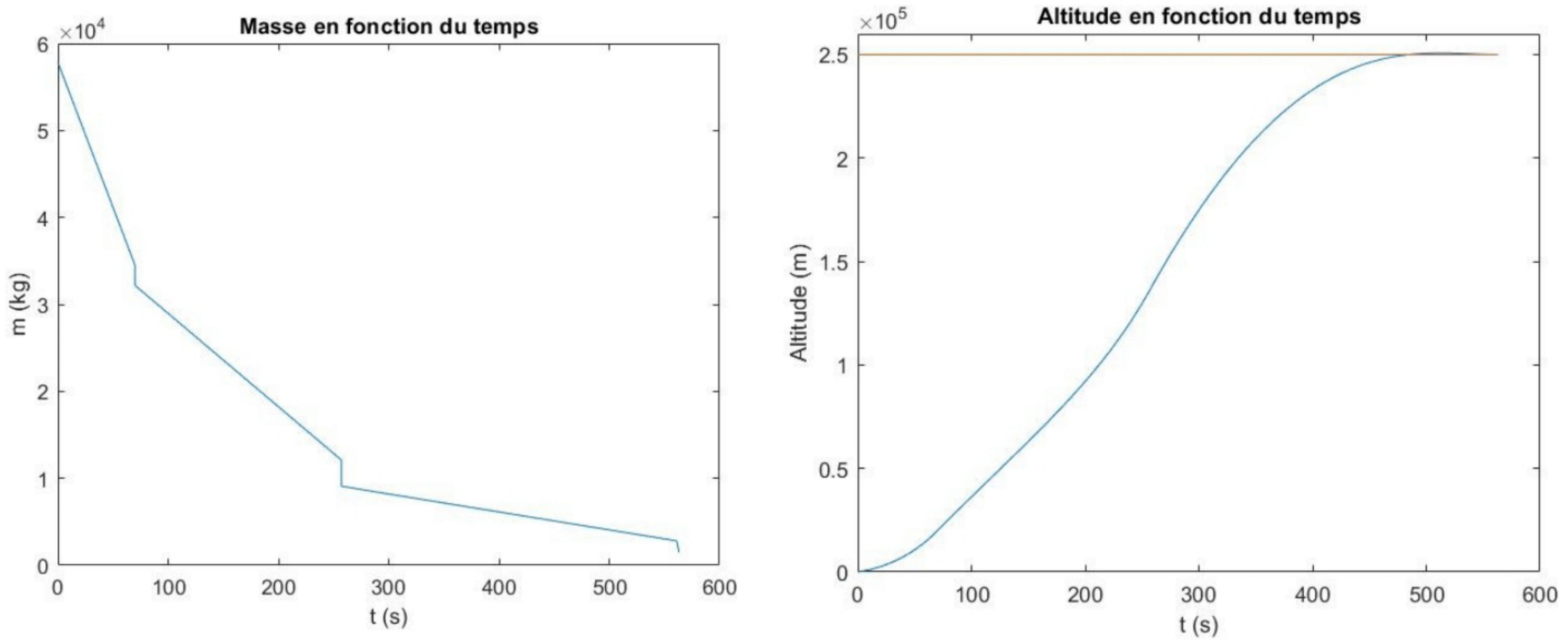
Vitesse propulsive	Masse Totale	Vitesse Réelle	thêta				masse d'ergols		
9305,81	51575,76	7538,06	-0,63	-2,67	17,96	5,73	25526,83	13065,12	5809,47
9522,59	57128,69	7972,58	37,68	18,68	20,19	3,77	30353,14	13442,02	5651,60
9304,85	51518,23	7706,66	36,66	18,35	20,64	4,48	26595,39	12341,28	5475,68
9353,04	52704,95	7754,78	36,75	18,40	20,75	4,33	27055,28	12870,10	5536,27

Résultats obtenus avec SR1 :

Vitesse propulsive	Masse Totale	Vitesse Réelle	thêta				masse d'ergols		
9305,81	52600,81	7206,70	0,12	-0,24	-7,18	8,36	20161,79	19397,30	5513,29
9853,95	69123,90	8072,30	14,37	6,42	19,24	3,43	27433,24	25254,73	7003,66
9536,49	58945,14	7812,04	35,62	18,25	24,22	2,63	23433,70	21576,83	5712,26
9479,29	57194,09	7746,45	35,42	18,16	24,21	2,89	22802,45	20691,94	5679,72
9487,68	57248,53	7737,89	35,13	18,06	24,27	3,23	22827,86	20388,00	5993,08
9504,63	57636,64	7746,57	35,07	18,06	24,26	3,44	23107,37	20192,96	6247,19
9512,90	57837,77	7754,44	35,04	18,04	24,21	3,50	23315,06	20095,29	6318,01

V Tracés de la trajectoire optimale





VI Conclusion

L'algorithme SQP semble suffisamment robuste et donne des résultats proches des solutions proposées dans le cours. De plus, il permet d'avoir de bons résultats sur le simulateur et sur le problème étagement-trajectoire.

La solution analytique est très proche de la solution trouvée avec notre programme de simulation.

En ce qui concerne le problème étagement-trajectoire, on note que la trajectoire du lanceur atteint bien l'orbite cible. L'altitude augmente régulièrement pour atteindre l'altitude cible également.

La masse diminue selon une courbe affine par morceaux car le débit massique de chaque étage est constant (débit d'éjection des ergols). De plus on observe bien les baisses brutales dues aux largages des étages. Le norme de la vitesse augmente rapidement. On remarque l'effet du largage des étages par deux petites irrégularités.

Pour rappel, la position, la vitesse et la masse du lanceur au temps initial représentent l'état du lanceur après la phase de décollage vertical. Autrement dit, la trajectoire obtenue n'inclut pas cette première phase.

En gardant ce détail à l'esprit, la trajectoire semble correcte. On voit qu'il est utile de calculer le thêta optimal car il est très différent du thêta de départ choisi à la main. En effet, l'ajustement manuel ne prenait pas en compte toutes les contraintes lors de la mise en orbite (atteindre l'altitude cible avec une vitesse horizontale).