Université Laval

Travail présenté à :
Antoine Allard
Louis Archambault
Daniel Côté

PHY-3500 : Physique numérique

Analyse numérique d'un système orbital Terre-Mars

Vincent Chouinard, 111 161 797
Justin Hamel, 111 239 203

6 mai 2020

1 Introduction

Un satellite naturel est par définition un objet céleste en orbite autour d'une planète ou d'un astre plus grand. Celui-ci se forme de trois manières [1] ; par accrétion, par capture ou par collision. Ces trois causes sont uniques et leurs conditions diffèrent. Par exemple, l'accrétion d'anneaux planétaires en satellites réguliers se fait uniquement lors de la formation d'une planète, alors qu'un satellite par collision implique un choc entre un astéroïde de grande taille et une planète, libérant un panache de matière qui finira par s'agglomérer. Cependant, les types de formation demandant un apport externe (collision, capture) ne peuvent créer un satellite stable s'il est sensiblement du même ordre de grandeur. Or, il n'existe aucun système d'orbite de ce genre dans le système solaire. Mais outre cette limitation, est-ce qu'une structure orbitale comme celle-ci pourrait être stable dans notre système solaire actuel ? Le but de ce projet est de remplacer la Lune par Mars et tenter de trouver une orbite stable pour des conditions frontières rendant ce modèle viable. Avec une orbite stable trouvée à l'aide d'un algorithme, nous étudierons les effets de ce nouveau système sur les marées terrestres et comparerons ces effets avec le modèle Terre-Lune actuel.

2 Algorithme de mécanique céleste

Pour modéliser le mouvement un système à trois corps de mécanique céleste, il est suffisant de résoudre la loi de la gravitation de Newton, dans laquelle la force sur un corps est donnée par :

$$\mathbf{F}_i = m_i \mathbf{a}_i = -G \sum_{j \neq i} m_i m_j \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|^3}$$

$$\dot{\mathbf{v}}_i = -G \sum_{j \neq i} m_j \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|^3}$$

La méthode numérique de saute-mouton [2] sera utilisée pour trouver la position et la vitesse des trois corps à partir de conditions initiales puis pour un instant qui suivra le précédent.

En définissant les constantes du problème dans le système d'unités internationales, il sera possible de résoudre le système pour un temps donné en secondes. Cet intervalle de temps est ensuite séparé dans un nombre de tranches N. C'est le nombre de tranches N qui définira la précision du calcul mais aussi le temps de calcul nécessaire pour résoudre le problème.

Pour s'assurer de la précision du système, il est nécessaire de tenter de recréer un système connu. Le système Terre-Lune-Soleil sera donc recréé à partir de conditions initiales centrées sur la Terre fournies par la NASA [3]. L'algorithme décrit bien la physique d'un système connu. De plus, contrairement aux problèmes à 3 corps classiques dans lesquels les corps passent très proches l'un de l'autre, l'algorithme n'a pas besoin d'une très grande précision pour modéliser précisément le système. En effet, l'apogée et le périgée de la Lune sont exactement aux bons endroits et le calcul a seulement nécessité environ 0.3 secondes.

L'algorithme de mécanique céleste est contenu dans la fonction *mouton_3_corps* dans le fichier *saute_mouton.py*. La fonction prend en argument le temps initial en secondes *t_i* (qui sera normalement 0 dans notre cas), le

1. A. Crida et S. Charnoz. "Formation of Regular Satellites from Ancient Massive Rings in the Solar System". In : *Science* 338.6111 (nov. 2012), p. 1196-1199. doi : 10.1126/science.1226477. url : https://doi.org/10.1126/science.1226477.
2. M. Newman. *Computational Physics*. Createspace Independent Pub, 2012. isbn : 9781480145511. url : https://books.google.ca/books?id=SS6uNAEACAAJ.
3. Jon.D. Giorgini. *JPL Solar System Dynamics*. url : https://ssd.jpl.nasa.gov/.

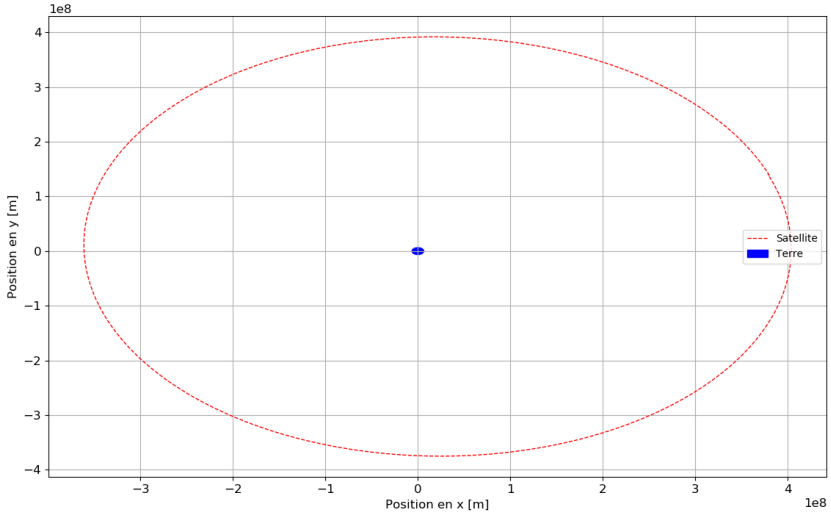


FIGURE 2.1 – Orbite de la Lune pour 27,322 jours calculé avec $N = 2000$ tranches

temps final en secondes *t_f*, le nombre de tranches *N*, les conditions initiales *c_init*, la fonction de l'équation différentielle à résoudre *F*. Les conditions initiales doivent être sous la forme d'un dictionnaire de la forme : [4]

```
{"Corps A": {"x": x_0, "y": y_0, "z": z_0,
             "vx": vx_0, "vy": vy_0, "vz": vz_0},
"Corps B": {{"x": x_0, "y": y_0, "z": z_0,
            "vx": vx_0, "vy": vy_0, "vz": vz_0}},
"Corps C": {{"x": x_0, "y": y_0, "z": z_0,
              "vx": vx_0, "vy": vy_0, "vz": vz_0}}}
```

La fonction F à résoudre doit être de la forme :

```
def F(corps, r_A, r_B, r_C):
    if corps == "A":
        return -G*(m_B*((r_A-r_B)/(np.linalg.norm(r_A-r_B)**3))
                   + m_C*((r_A-r_C)/(np.linalg.norm(r_A-r_C)**3)))

    if corps == "B":
        return -G*(m_A*((r_B-r_A)/(np.linalg.norm(r_B-r_A)**3))
                   + m_C*((r_B-r_C)/(np.linalg.norm(r_B-r_C)**3)))

    if corps == "C":
        return -G*(m_A*((r_C-r_A)/(np.linalg.norm(r_C-r_A)**3))
                   + m_B*((r_C-r_B)/(np.linalg.norm(r_C-r_B)**3)))
```

Où r est le vecteur de position du corps concerné.

4. Dans le cas présent les corps A, B et C seront respectivement la Terre, Mars, et le Soleil.

La fonction *mouton_3_corps* a aussi quelques arguments optionnels qui seront très utiles plus tard. Le premier argument optionnel est l'argument *slice* qui sera égal au nombre de fois que les *array* de positions finales seront coupées de moitié en enlevant un élément sur deux. L'argument sera utile pour afficher ou animer les résultats obtenus à partir de la fonction sans perdre de précision.

L'argument *var* permettra, lors de l'utilisation avec la fonction *perturbations*, de préciser la stabilité de quelle valeur initiale est étudiée.

L'argument *tol_valide* dicte si l'utilisateur tolère les orbites invalides, car ignorer une orbite qualifiée d'invalide pourra permettre d'accélérer les opérations de la fonction de recherche de stabilité.

L'argument *verbose* fera en sorte que la fonction affichera le temps d'exécution à la fin des calculs s'il est vrai.

Puisque la fonction *saute_mouton* sera utilisée par toutes les autres fonctions, il est nécessaire que celle-ci donne le maximum d'informations possible à sa sortie. La fonction donne donc un grand dictionnaire contenant :

- "A" : les positions vectorielles à chaque instant calculé du corps A.
- "B" : les positions vectorielles à chaque instant calculé du corps B.
- "C" : les positions vectorielles à chaque instant calculé du corps C.
- "L" : les positions vectorielles à chaque instant calculé du satellite de la Terre avec la Terre comme origine.
- "S" : les positions vectorielles à chaque instant calculé du Soleil avec la Terre comme origine.
- "P" : la proximité scalaire du satellite de la Terre par rapport à la Terre.
- "t" : les valeurs de temps associées à chaque instant calculé.
- "valide" : un booléen qui dicte si l'orbite est considérée comme valide.
- "c_init" : les conditions initiales utilisées pour les calculs.
- "var" : la stabilité de quelle variable est-ce que l'utilisateur s'intéresse (pertinent seulement lorsqu'utilisé avec la fonction *perturbations*)

Ainsi, à titre de comparaison, la proximité de la Lune par rapport à la Terre est affichée dans la figure 2.2.

3 Algorithme de calcul de stabilité

Puisque le but du projet est de trouver une orbite stable pour Mars autour de la Terre, et qu'il est connu depuis longtemps que l'orbite de la Lune n'est déjà pas complètement stable [5], il sera nécessaire de créer un algorithme qui trouvera une orbite stable selon de nouvelles conditions initiales. Il faudra donc quantifier la stabilité d'une orbite en fonction d'une condition initiale.

3.1 Quantification de la stabilité

Il existe plusieurs manière de quantifier la stabilité d'une orbite. Dans le cas présent, puisque tous les orbites analysées seront des orbites elliptiques, et que dans le cas des marées, une orbite stable sera une orbite

5. G. W. Hill. "Researches in the Lunar Theory". In : *American Journal of Mathematics* 1.1 (1878), p. 5. DOI : 10.2307/2369430. URL : https://doi.org/10.2307/2369430.

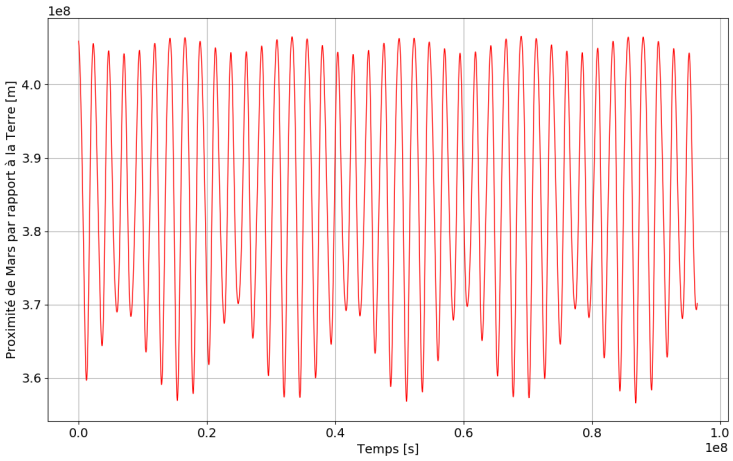


FIGURE 2.2 – Proximité de la Lune par rapport à la Terre pour 3 ans d'orbite

qui gardera ses caractéristiques orbitales. De plus, dans le cas d'une orbite elliptique, c'est principalement l'excentricité qui décrit le mieux les caractéristiques de l'ellipse puisque ce paramètre contient des informations sur le demi-grand axe et sur le petit-grand axe, qui auront le plus grand impact sur les marées.

Une orbite stable sera donc quantifiée par la variation de l'excentricité de l'orbite. En effet, une orbite dont l'écart type de l'excentricité est faible sera qualifié de stable.

Maintenant qu'un critère de stabilité a été défini, il sera aussi pertinent de considérer dans quelle région de l'espace une orbite pourra être pour être considérée comme valide. En effet, il sera important de savoir à quelle distance de la Terre est-ce qu'un satellite ne sera plus sous l'influence gravitationnelle de la Terre mais bien du Soleil, et donc à quelle distance de la Terre est-ce que la Terre perd son satellite ?

L'approximation de la sphère de Hill sera utilisée pour répondre à ce critère. En effet, on peut approximer que le rayon maximale qu'a l'influence gravitationnelle avant de perdre son influence au Soleil peut se calculer de la manière suivante [6] :

$$r_H \approx a(1-e)\left(\frac{m}{3M}\right)^{\frac{1}{3}}$$

Où a, e, m et M sont respectivement le demi-grand axe de la Terre autour du Soleil, l'excentricité de l'orbite de la Terre autour du Soleil, la masse de la Terre et la masse du Soleil. On trouve que pour la Terre, la sphère de Hill a un rayon de :

$$\begin{aligned} r_H &\approx 149.60 \times 10^9 (1 - 0.0167)\left(\frac{5.9724 \times 10^{24}}{3 \cdot 1.989 \times 10^{30}}\right) \\ &= 1.47146 \times 10^9 \text{ [m]} \end{aligned}$$

Une orbite qui atteint cette distance de la Terre sera donc considéré comme invalide aux calculs de stabilité puisque le satellite deviendra plutôt un satellite du Soleil.

De la manière que la sphère de Hill donne une distance maximale à la Terre, il existe la limite de Roche qui délimitera la distance minimale à la Terre pour avoir une orbite valide. En effet, la limite de Roche définit

6. Douglas P. HAMILTON et Joseph A. BURNS. "Orbital stability zones about asteroids". In : *Icarus* 96.1 (mar. 1992), p. 43-64. DOI : 10.1016/0019-1035(92)90005-r. URL : https://doi.org/10.1016/0019-1035(92)90005-r.

une distance minimale entre deux objets avant que l'objet le plus petit objet des deux soit déchiré par les forces de marées du plus gros objet. [7] Elle est définit par :

$$d = R_M \left(2\frac{\rho_M}{\rho_m}\right)^{\frac{1}{3}}$$

Où dans le cas présent, R_M, ρ_M et ρ_m sont respectivement le rayon de la Terre, la densité de la Terre et la densité de Mars. À l'aide de données de la NASA [8], il est possible de trouver la limite de Roche pour la Terre et Mars :

$$\begin{aligned} d &= 6.3781 \times 10^6\,\mathrm{m}\left(2\frac{5514\,\mathrm{kg/m^3}}{3933\,\mathrm{kg/m^3}}\right)^{\frac{1}{3}} \\ &= 8.993920 \times 10^6\,[\mathrm{m}] \end{aligned}$$

une orbite sera donc considérée comme invalide si Mars atteint une distance plus basse que 8.993920×10^6 m puisque Mars aura explosé et les débris auront probablement causés un cataclysme ou si Mars atteint une distance de 1.47146×10^9 m puisque Mars ne serait plus en orbite autour de la Terre.

3.2 Enregistrement des données

Puisque la stabilité d'une orbite en fonction d'une condition initiale sera mesurée, il sera donc nécessaire de calculer plusieurs systèmes pour un seul point de stabilité en fonction d'une condition initiale. Ces calculs nécessiteront donc beaucoup de temps. La décision a donc été prise d'enregistrer des données d'avance et de les charger au moment opportun. Ainsi, les longs calculs peuvent être effectués d'avance, et les données peuvent être analysées n'importe quand. La fonction *sauvegarde* dans le fichier *stabilite.py* enregistre le dictionnaire de données de la fonction *mouton_3_corps* dans des fichiers nommés par des informations pertinentes qu'ils contiennent. La fonction *chargement* extrait les données à l'aide du nom du fichier et retourne le dictionnaire enregistré dans le fichier.

```
1  def sauvegarde(t_i, t_f, N, c_init, F, slice=0, var="x", tol_valide=True, id=0):
2      mouton = mouton_3_corps(t_i, t_f, N, c_init, F, slice, var)
3      nom_fichier = "TMS_{}_jours_{}_N_{}{}".format(round(t_f/(24*3600)), N, var, id)
4      pickle.dump(mouton, open(str(Path.cwd()/"Données"/nom_fichier), "w+b"))
5      return mouton
6
7
8  def chargement(nom_fichier):
9      data = pickle.load(open(str(Path.cwd()/"Données"/str(nom_fichier)), "rb"))
10     return data
```

3.3 Fonction de perturbations des conditions initiales

La fonction *perturbations* est définie pour prendre des conditions initiales de départ (dans le cas présent ce sera les conditions initiales de la Lune) et y appliquer des perturbations. La fonction calculera donc l'orbite

7. G. R. ""Roche's Limit"". In : *Nature* 47.1222 (mar. 1893), p. 509-510. DOI : 10.1038/047509b0. URL : https://doi.org/10.1038/047509b0.

8. https://nssdc.gsfc.nasa.gov/planetary/factsheet/marsfact.html

sur un certain nombre de temps pour différentes conditions initiales autour de la condition initiale de départ. La signification de perturbations n'est pas réellement au sens physique du terme, mais plutôt dans le sens où la fonction applique une "perturbation" à la condition initiale et calcule l'orbite après cette modification légère de l'orbite. De cette manière il sera possible d'obtenir plusieurs orbites à partir desquelles la stabilité en fonction d'une condition initiale sera mesurée.

```
# Applique un nombre de perturbations d'un certain ordre (exemple 10^7 m)
# sur une condition initiales var et enregistre les données
def perturbations(t_i, t_f, N, c_init, F, slice=0, var="x", nombre=10, ordre=1e7):
    c_init["Mars"][var] = c_init["Mars"][var] - (nombre/2)*ordre
    orbites_stables = 0  # on comptera le nombre d'orbites stables trouvés
    for i in range(nombre):
        c_init["Mars"][var] = c_init["Mars"][var] + i*ordre
        data = sauvegarde(t_i, t_f, N, c_init, F, slice, var, tol_valide=False, id=i)

        if data["valide"] is True:
            orbites_stables += 1

        if data["valide"] is False and orbites_stables > 15:
            return i  # retourne le nombre de fichiers créés après au moins 15 orbites stables

    return nombre  # retourne le nombre de fichiers créés si auncune orbite instable
```

3.4 Fonction de calcul de stabilité

Puisqu'il est plus difficile d'obtenir une fonction analytique à partir des données de position d'un système relativement chaotique, il sera plus facile de simplement traiter les données pour en obtenir une valeur de stabilité.

Sachant que les données obtenus par l'algorithme de mécanique céleste vont contenir la proximité de Mars par rapport à la Terre, et sachant que la révolution de Mars autour de la Terre sera plus courte qu'un mois, mais dépendra de la condition initiale perturbée, la proximité de Mars par rapport à Terre sera calculée sur plusieurs mois et sera sectionnée en tranches de 31 jours. De cette manière, il sera facile d'aller cherche le minimum et le maximum de proximité. Ainsi, la valeur de du demi-grand axe a de l'orbite approximative du mois sera :

$$a = \frac{\max - \min}{2}$$

La valeur du demi-petit axe b pourra être trouvé à l'aide d'un peu de trigonométrie. En effet, l'extrémité de b qui touche à l'extérieur de l'orbite est exactement au centre du minimum et du maximum. b se trouve donc à la médiane de l'orbite. Sa longueur sera donc trouvée de la manière suivante :

$$b = \sqrt{\text{médiane}^2 - (a - \min)^2}$$

La figure 3.1 permet de visualiser la manière de trouver la longueur de b.

L'excentricité d'un mois d'orbite se trouve ainsi facilement :

$$e = \sqrt{1 - \frac{a^2}{b^2}}$$

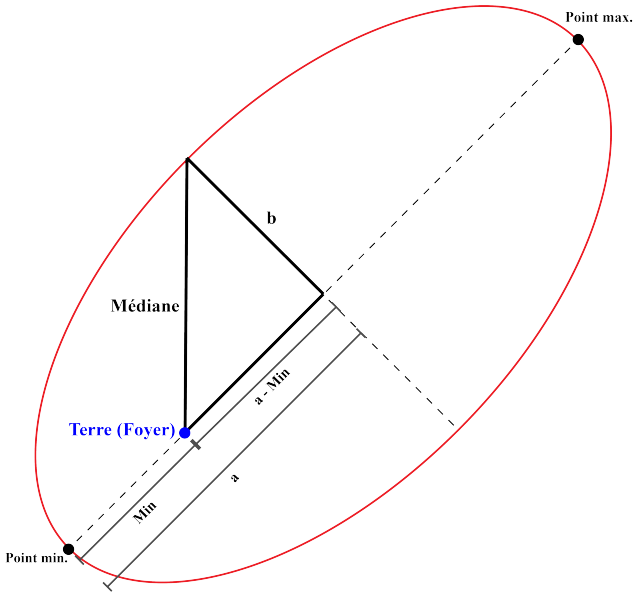


FIGURE 3.1 – Visualation de la méthode pour trouver la longueur de b

Ainsi, en récoltant les positions des corps pour plusieurs mois, la stabilité est quantifiée à partir de l'inverse l'écart type de l'excentricité de chaque mois. Donc, une orbite dont l'excentricité ne change pas beaucoup au fil des mois, aura un écart type petit et un pic de stabilité sera visible et facilement identifiable.

La figure 3.2 montre un exemple de données qui seraient utilisées pour trouver la stabilité. Il est possible de remarquer que les minimums et les maximums changent. À vue d'oeil, il est impossible de distinguer les valeurs de b.

La fonction *stabilite* dans le fichier *stabilite.py* trouve la stabilité de plusieurs orbites préalablement enregistrés par la fonction *perturbations*. La fonction prend donc en argument le temps en jours, le nombre de tranches N, le nom des fichiers, la variable d'intérêt, et le nombre de fichiers à analyser. Ce sont toutes des informations qui ont été enregistrées dans le nom des fichiers par la fonction *sauvegarde*.

À l'aide de la fonction de perturbations, il sera possible de calculer la stabilité des orbites. La figure 3.3 montre la stabilité d'un ensemble d'orbites selon leur conditions initiales en x calculés sur 2 ans. Il est possible de remarquer qu'il semble y exister un maximum de stabilité qui n'est pas placé à la valeur initiale de la Lune. Il est donc possible d'améliorer la stabilité de l'orbite.

3.5 Fonction de recherche d'orbites stables

Si la condition initiale en x la plus stable est choisie, il est possible de s'intéresser à quelle condition initiale en y serait la plus stable. En prenant le maximum de stabilité en y, l'équilibre est légèrement dérangé, et la condition initiale en x n'est plus la plus stable. Pour trouver l'ensemble de conditions initiales en x et y le plus stable, il faudra refaire plusieurs fois les étapes suivantes : [9]

— Trouver la condition initiale en x la plus stable

— Trouver la condition en y la plus stable à partir de la valeur en x trouvée l'étape précédente

9. Il est à noter que l'impact de la condition initiale en z sera ignoré puisque l'inclinaison de l'orbite de la Lune n'est pas si prononcée, et que l'opération qui est déjà très longue serait ainsi allongée.

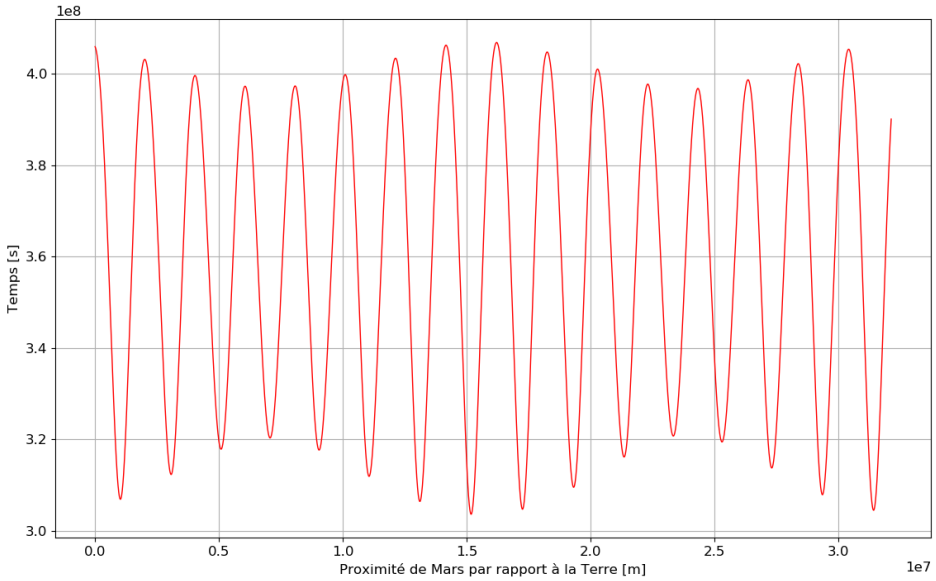


FIGURE 3.2 – Proximité de Mars par rapport à la Terre pendant 1 an à partir des conditions initiales de la Lune le 23 avril 2020

Puisqu'on tend vers la stabilité, l'orbite sera de plus en plus stable. Mais clairement l'opération sera longue et se fera mal à la main. Un algorithme sera donc implémenté pour trouver les conditions initiales optimales à partir des fonctions présentées plus haut.

L'algorithme de recherche de stabilité est basé sur les fonctions *perturbations* et *stabilite*. En effet, la fonction à l'aide du boucle *while* qui arrête lorsqu'une précision désirée est atteinte. La précision est définie par l'écart entre la nouvelle valeur de condition initiale stable trouvée et la précédente. À chaque étape de la boucle, la fonction *perturbations* est appelée pour la variable x puis appelée de nouveau pour la variable y. La fonction *stabilite* est alors appelée séparément pour chacune des variables pour trouver le maximum de stabilité dans les fichiers créés par *perturbations*. De cette on tend vers un maximum de stabilité.

Par contre, l'opération est assez longue et donc il est important de chercher des moyens d'accélérer la procédure. Puisque chaque fichier créé par *perturbations* est un point de stabilité pour la fonction *stabilité*, il est nécessaire de créer plusieurs fichiers pour avoir une précision adéquate pour définir les nombreux pics (voir figure 3.3). C'est donc la fonction *perturbations* qui ralentit le processus puisqu'elle appelle plusieurs la fonctions *mouton_3_corps* qui demeure fondamentalement lente. De plus, à la base, *perturbations* calcule et enregistre des fichiers qui seront complètement ignorés par la fonction *stabilite* puisque ces fichiers représentent des orbites qualifiées d'invalides.

Des conditions sont donc présentes dans la fonction *perturbations* pour arrêter les calculs après avoir recueilli un nombre suffisant d'orbites stables, puisqu'après l'atteinte du premier orbite instable, toutes les autres orbites sont normalement instables dû à la nature du système.

De plus, lorsque l'argument *tol_valide* est faux, la fonction *mouton_3_corps* arrête les calculs dès que la position de l'astre n'est plus dans les valeurs considérées valides qui sont définies par la limite de Roche et la sphère de Hill.

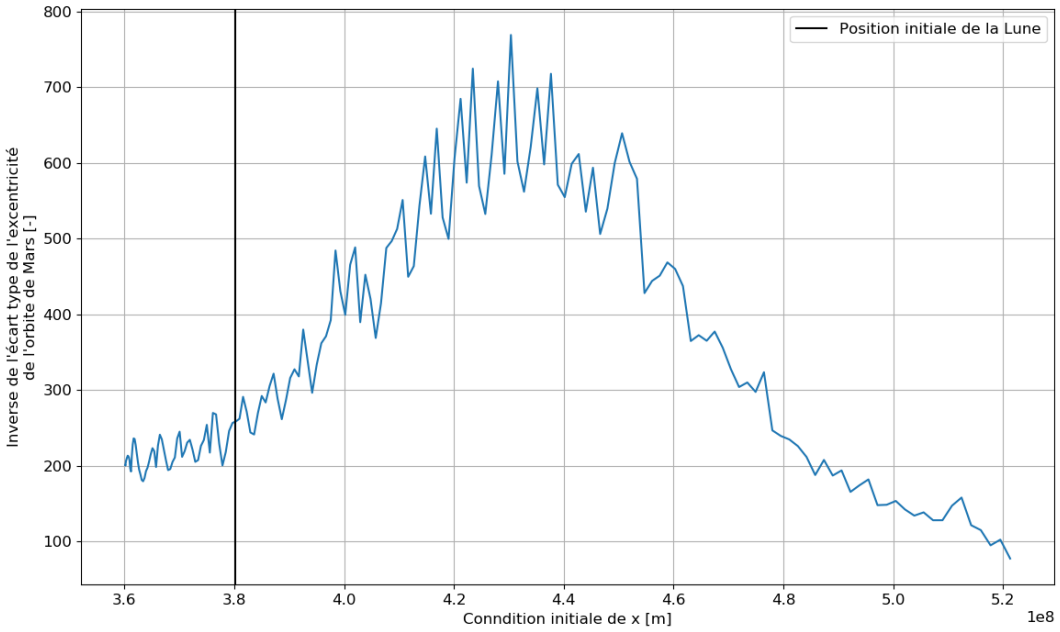
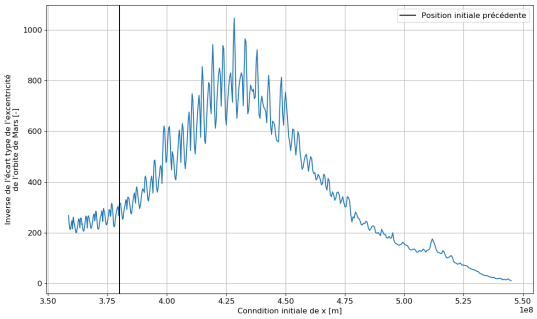
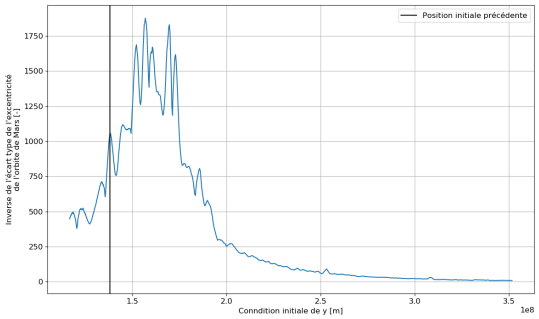


FIGURE 3.3 – Inverse de l'écart type de l'excentricité de Mars pour deux ans d'orbite autour de la Terre où les conditions initiales en x sont séparées de 10^3 m

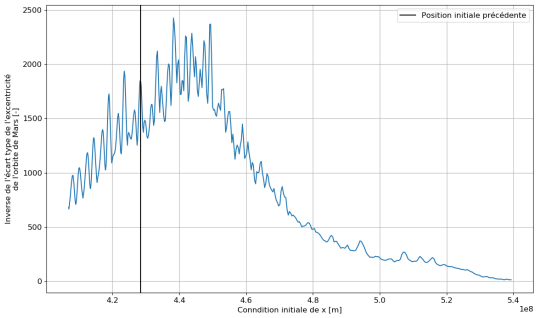
Les figures 3.4 et 3.5 montrent la progression de l'algorithme pour des orbites de 3 ans à partir des conditions initiales de la Lune. Il est possible de remarquer que les formes des pics changent dans les premières itérations, mais deviennent de plus en plus stables alors que l'algorithme tend vers une valeur. Il est par contre difficile d'atteindre la précision demandée. En effet, dans ce cas ci, la précision demandée était de 10^4 et les points étaient séparés de 10^3. La plus grande précision obtenue était de 2.8×10^5, ce qui était très satisfaisant pour l'ordre du système étudié. En utilisant les nouvelles conditions initiales trouvées à l'aide de l'algorithme, la nouvelle orbite de Mars est trouvée. Les figures 3.6 et 3.7 permettent de comparer l'orbite de Mars à partir des conditions initiales de la Lune et l'orbite de Mars à partir des conditions initiales optimales trouvées avec l'algorithme. Il est évident que l'orbite est plus stable selon les critères utilisés, puisque celle-ci est moins élargie et garde sa forme de manière plus constante et restreinte. Il est aussi possible de remarquer que l'orbite est aussi beaucoup plus circulaire en observant les valeurs des extrémités de l'orbite. La proximité de Mars par rapport à la Terre varie aussi dans un domaine plus restreint dans le cas des conditions initiales trouvées avec l'algorithme. Par contre, il est possible d'observer au moins deux battements dans les valeurs de la proximité, alors que dans le cas avec les conditions initiales de la Lune, seulement un battement était clairement remarquable. Il reste à voir quel impact ce changement dans l'orbite aura-t-il sur les marées sur Terre.



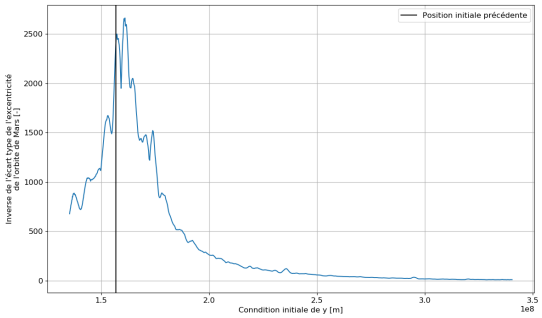
(a) Itération 1 en x



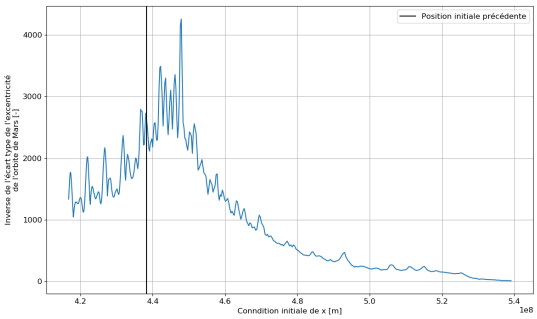
(b) Itération 1 en y



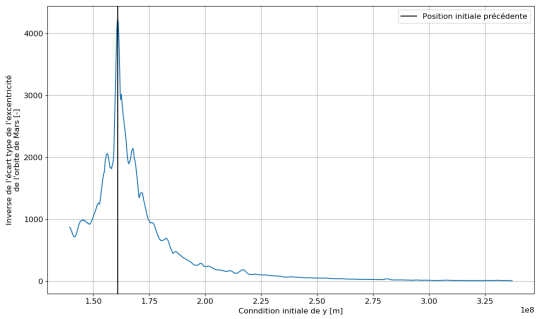
(c) Itération 2 en x



(d) Itération 2 en y

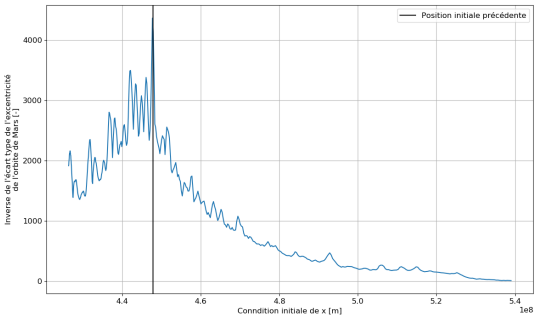


(e) Itération 3 en x

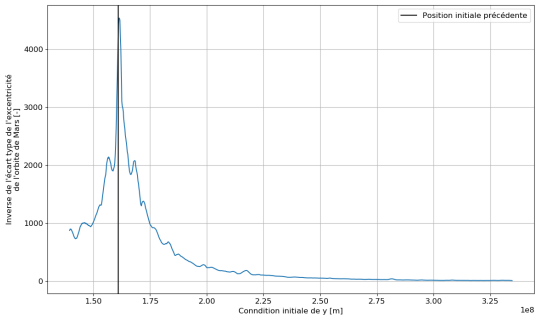


(f) Itération 3 en y

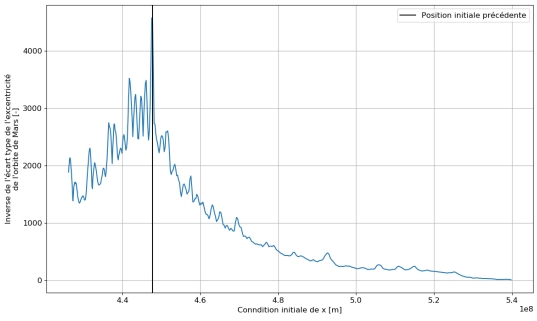
FIGURE 3.4 – Affichage des 3 premières itérations des résultats utilisés par l'algorithme de recherche de stabilité. La barre noire représente la valeur précédente



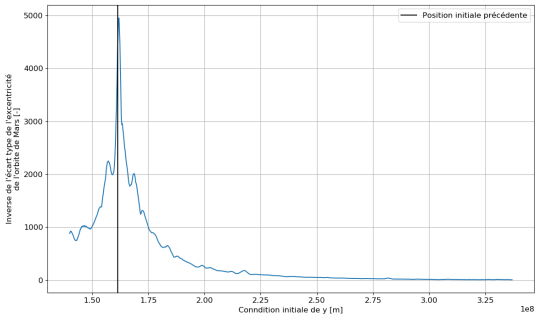
(a) Itération 4 en x



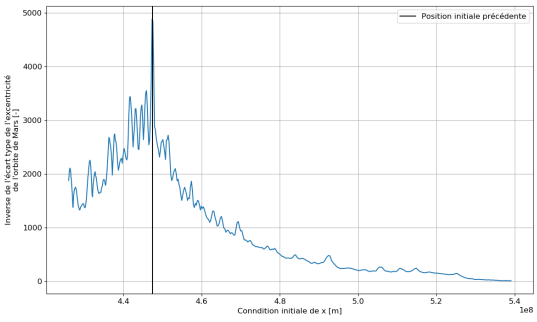
(b) Itération 4 en y



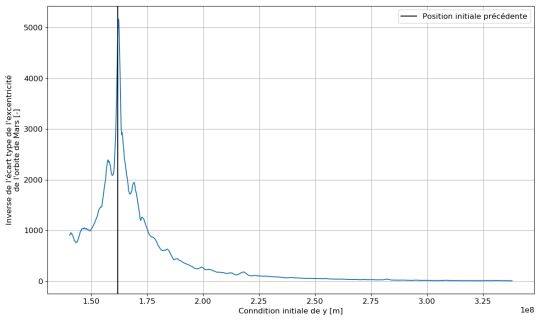
(c) Itération 5 en x



(d) Itération 5 en y

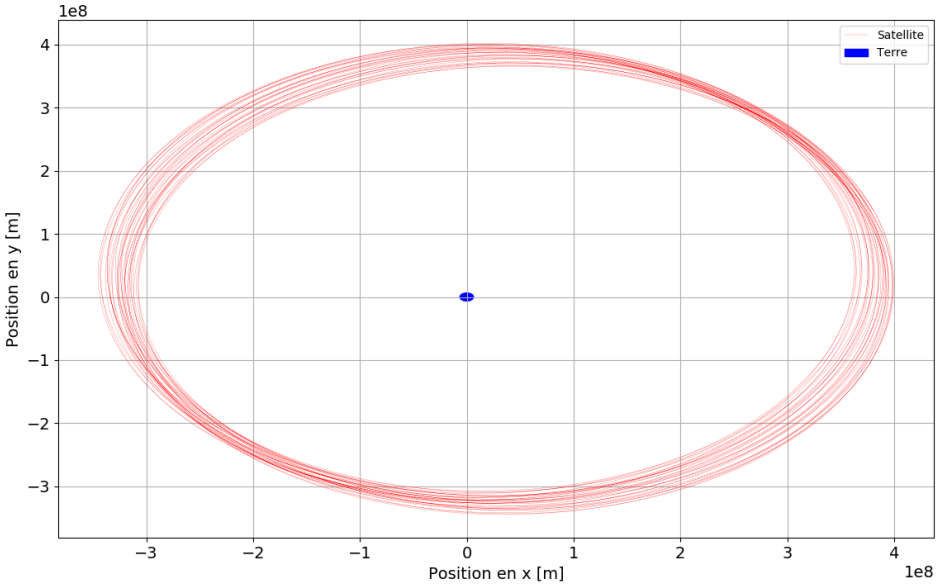


(e) Itération 6 en x

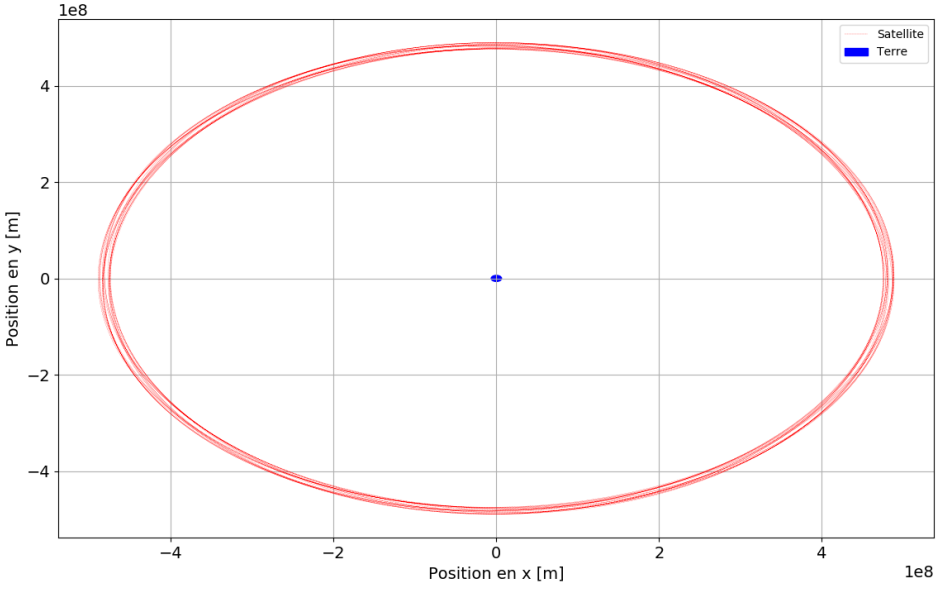


(f) Itération 6 en y

FIGURE 3.5 – Affichage des 3 dernières itérations des résultats utilisés par l'algorithme de recherche de stabilité. La barre noire représente la valeur précédente

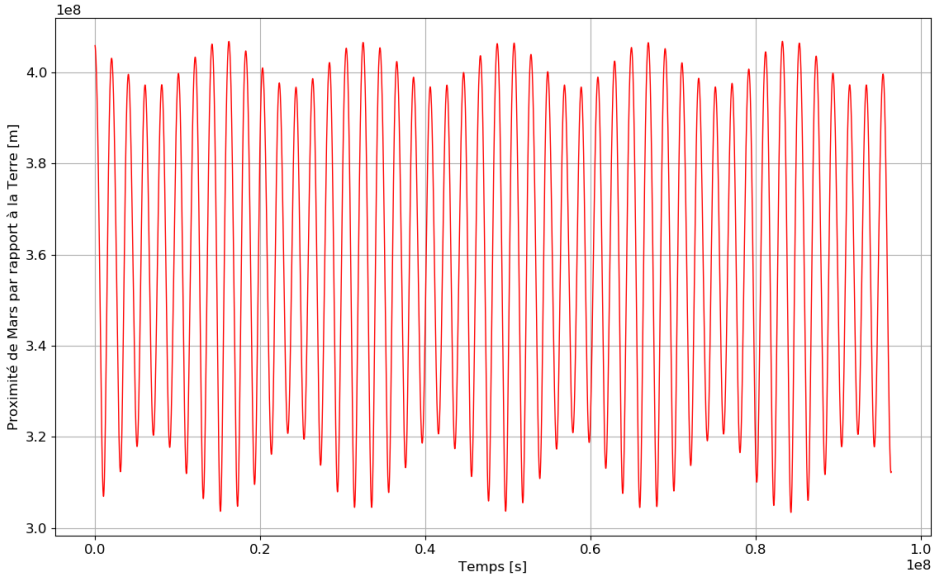


(a) Orbite de Mars avec les conditions initiales de la Lune

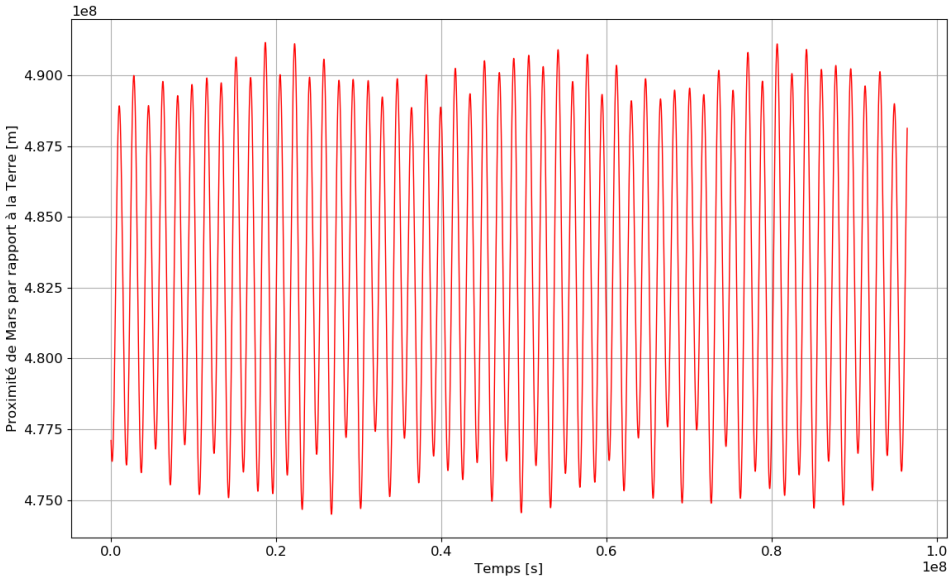


(b) Orbite de Mars avec les nouvelles conditions initiales optimales

FIGURE 3.6 – Orbites de Mars pour deux ensembles de coordonnées de conditions initiales différentes pour 2 ans d'orbite.



(a) Proximité de Mars par rapport à la Terre avec les conditions initiales de la Lune



(b) Proximité de Mars par rapport à la Terre avec les nouvelles conditions initiales optimales

FIGURE 3.7 – Proximité de Mars par rapport à la Terre pour deux ensembles de coordonnées différentes pour 3 ans d'orbite.

4 L'étude des marées d'un système Terre-Mars

La dernière partie du projet consiste à prendre notre nouveau modèle numérique et vérifier les effets concrets qu'il aurait sur notre vie courante. La force de gravitation entre les deux corps de notre système est essentiellement notre point de départ pour comparer nos systèmes. vérifions l'apport gravitationnel de Mars sur la terre :

$$F_{gmars} = \frac{Gm_1m_2}{d^2}$$

Dans le fichier fonctions_marees.py, une équation simple trouve la distance entre un corps et l'origine (dans notre cas Mars et la Terre respectivement) avec ses composantes en (x,y,z). Nous obtenons à l'aide de nos conditions initiales stables de notre système TMS_stable une distance $d = 477105544.62$ [m]. Nous pouvons maintenant trouver l'accélération gravitationnelle causée par Mars sur une masse quelconque $[m_2]$ sur terre :

$$F_{gmars} = 0.000188m$$

Nous constatons que l'apport est minime par rapport à notre accélération gravitationnelle terrestre g de 9,8 ms^{-2}. Ceci-dit, sur 1.4×10^{10} mètres cubes d'eau, une force d'une telle amplitude crée des mouvements dans le liquide. Nous étudierons donc les marées causées par notre nouveau modèle en équilibre. Pour ce faire, deux méthodes ont été considérées. La première méthode consiste à résoudre les équations partielles de Laplace pour les marées [10] :

$$\begin{aligned}
\frac{\partial \zeta}{\partial t} + \frac{1}{a\cos(\varphi)}\left[\frac{\partial}{\partial \lambda}(uD) + \frac{\partial}{\partial \varphi}(vD\cos(\varphi))\right] &= 0 \\
\frac{\partial u}{\partial t} - v(2\Omega\sin(\varphi)) + \frac{1}{a\cos(\varphi)}\frac{\partial}{\partial \lambda}(g\zeta + U) &= 0 \\
\frac{\partial v}{\partial t} + u(2\Omega\sin(\varphi)) + \frac{1}{a}\frac{\partial}{\partial \varphi}(g\zeta + U) &= 0
\end{aligned}$$

Cependant, pour résoudre cet ensemble d'équations différentielles, nous avons besoin de conditions frontières. Dans notre cas, nous devions paramétriser mathématiquement les côtes des continents et des îles sur l'entièreté du globe. Nous avons donc opté pour une équation sensiblement plus simple, où nous considérons que la terre est une sphère sans côtes.

4.1 L'Equilibrium Tide Equation

Nous utilisons donc *l'Equilibrium Tide Equation* [11] que nous appliquons en tout point sur la Terre :

$$\bar{\zeta_l} = r\frac{m_l}{m_e}\left[C_0(t)\left(\frac{3}{2}\sin^2\phi_P - \frac{1}{2}\right) + C_1(t)\sin 2\phi_P + C_2(t)\cos^2\phi_P\right] \tag{4.1}$$

10. David A. Randall. *The Laplace Tidal Equations and Atmospheric Tides*. Accessed : 11-04-2020.
11. David Pugh et Philip Woodworth. *Sea-Level Science*. Cambridge University Press, 2014. doi : 10 . 1017 / cbo9781139235778. url : https://doi.org/10.1017/cbo9781139235778.

Nous obtenons la hauteur de la marée $\bar{\zeta}_l$ à un point quelconque choisi avec la latitude et la longitude (des transformations sont requises). $C_0(t)$, $C_1(t)$ et $C_2(t)$ sont :

$$C_0(t) = \left(\frac{r}{R_1}\right)^3 \left(\frac{3}{2}\sin^2 d_l - \frac{1}{2}\right) \tag{4.2}$$

$$C_1(t) = \left(\frac{r}{R_l}\right)^3 \left(\frac{3}{4}\sin 2d_l \cos C_P\right) \tag{4.3}$$

$$C_2(t) = \left(\frac{r}{R_l}\right)^3 \left(\frac{3}{4}\cos^2 d_l \cos 2C_P\right) \tag{4.4}$$

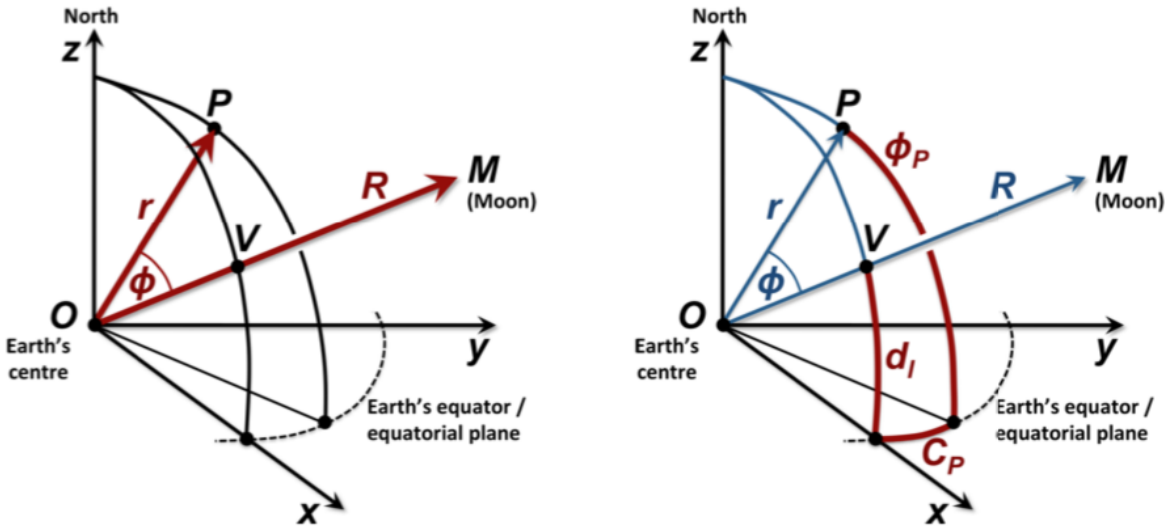


FIGURE 4.1 – Graphiques des variables de *l'Equilibrium Tide Equation*. Figure tirée du *Sea-Level Science*[12]

La figure 4.1 représente en graphiques d'un système Terre-Lune les variables utilisées dans les équations (4.1) à (4.4). L'équation originale est uniquement par rapport à l'angle ϕ (que l'on voit dans la figure gauche ci-haut). Pour appliquer notre système en tout point P, la décomposition de ϕ en trois constantes d_l, C_p et ϕ_P est nécessaire. Le fichier python fonctions_marees.py sert essentiellement à transformer ces variables en coordonnées sphériques. La sorte de coordonnées sphériques utilisée n'est pas celle communément acceptée en physique, mais plutôt celle utilisée en géographie. Ainsi, notre θ est l'angle de la longitude et δ est la latitude. Dans ce fichier python, six équations sont définies. Trois d'entre eux sont pour trouver d_l, C_p et R_l avec les composantes (x,y,z) de notre astre. L'angle ϕ_p est simplement la latitude de notre point P, donc c'est une variable flottante dans l'optique ou nous voulons éventuellement obtenir la hauteur des marées en tout point. Ensuite, trois fonctions servent à trouver nos coefficients C_0, C_1 et C_2. Voici un exemple de l'un d'eux :

```
1 def C_2(R_astre, d_l, C_p):
2     return (r/R_astre)**3.0 * (0.75)*np.cos(d_l)**2.0 * np.cos(2*C_p)
```

Finalement, notre fonction "Equilibrium" prend toutes ces informations et les combine pour obtenir la hauteur de la marée selon la longitude et la latitude choisie. Il est important de noter que la fonction "Equilibrium" fait une somme de hauteurs pour l'effet que l'astre aura sur la marée ajouté à l'effet du Soleil. Il suffit donc de faire deux fois l'équation avec des positions (x,y,z) pour l'astre et (x_s, y_s, z_s) pour le soleil en argument

et retourner la somme résultante.

Revenons sur les limitations de notre système. Le modèle que nous ferons ne prendra pas en compte les côtes des continents et des îles. C'est une limitation majeure. Une marée côtière peut varier grandement [13] . Par exemple, la baie de Fundy conserve le record mondial du plus grand marnage avec une différence entre une basse-mer et haute-mer de 17 mètres. Ainsi, pour que nos comparaisons soient vraisemblables, nous prenons le marnage moyen des océans ouverts. Cette valeur est de $\bar{H} = 0.75$ [m] [14].

4.2 Fonctions dans grid_2D_Equilibrium.py

Nous voulons obtenir un graphique 3D qui suit le développement de la position des astres du système trouvé avec les algorithmes de la mécanique céleste abordés ci-haut. Mais avant de s'y rendre, il nous faut un array de NxN points qui contient le champ scalaire en deux dimensions. Ce champ scalaire sera ensuite placé dans des cellules ouvertes qui forment ensemble une géométrie sphérique. Le fichier grid_2D_Equilibrium contient une fonction qui nous permet de présenter un graphique en deux dimensions du champ scalaire. Cette fonction est accompagnée d'une fonction "lambda" qui transforme la fonction "Equilibrium" de fonctions_marees.py en fonction ne dépendant que de la longitude et latitude :

```
# Retourne une fonction Equilibrium avec seulement theta et delta comme paramètres d'entrée
def fonc_Equilibrium(x_l, y_l, z_l, x_s, y_s, z_s, m_astre):
    return lambda theta, delta: Equilibrium(x_l, y_l, z_l, x_s, y_s, z_s, m_astre, theta, delta)


# Crée une grid NxN de champ scalaire p/r à une fonction donnée f(theta, delta) et la plot en "heatmap"
def grid_fct(N, fct):
    lon = np.linspace(-1.0 * np.pi, 1.0 * np.pi, N)
    lat = np.linspace(-0.5 * np.pi, 0.5 * np.pi, N)
    theta, delta = np.meshgrid(lon,lat)
    G = np.zeros((N,N))

    for i in range(N):
        for j in range(N):
            G[i,j] = fct(theta[i,j], delta[i,j])

    plt.pcolor(theta, delta, G)
    c = plt.colorbar()
    c.ax.set_ylabel("Hauteur de la marée [m]", rotation=270)
    plt.xlabel("Longitude [rad]")
    plt.ylabel("Latitude [rad]")
    plt.show()
```

13. Joris Vanlede, Leen Coen et Maarten Deschamps. "Tidal Prediction in the Sea Scheldt (Belgium) Using a Combination of Harmonic Tidal Prediction and 1D Hydraulic Modeling". In : *Natural Resources* 05.11 (2014), p. 627-633. DOI : 10.4236/nr.2014.511055. URL : https://doi.org/10.4236/nr.2014.511055.

14. Vanlede, Coen et Deschamps, "Tidal Prediction in the Sea Scheldt (Belgium) Using a Combination of Harmonic Tidal Prediction and 1D Hydraulic Modeling".

Ces fonctions servent globalement à tester notre équation d'Equilibrium. Voici le graphique résultant dans le cas où le Soleil et Mars sont à leur position initiale dans notre dictionnaire sys_TMS_stable :

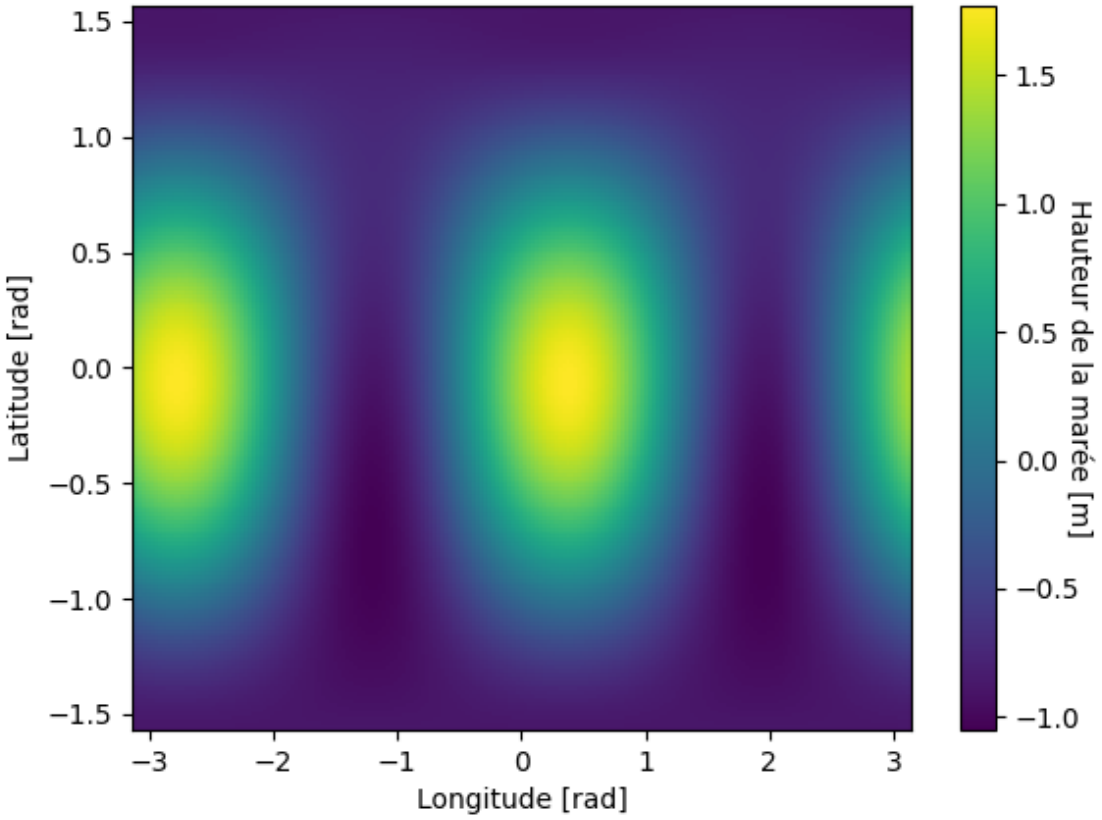


FIGURE 4.2 – Champ scalaire $\bar{\zeta}$

Nous pouvons constater sans surprise que deux maximums sont présents à la figure 4.2. En effet, le temps entre une marée haute et une marée basse est d'environs 6 heures. Ainsi, pour une rotation complète de la terre, deux marées hautes et deux marées basses auront lieu. Ce phénomène est explicable par somme de forces. puisque la surface d'eau la plus proche du satellite est davantage attiré que le centre de la terre, nous obtenons une courbature elliptique. Ce même raisonnement est aussi applicable à la surface d'eau opposée au satellite. Celle-ci est moins attirée par le satellite que le centre de la terre. Ainsi, pour la partie opposée, la terre est attirée davantage vers le satellite que l'eau derrière, donc nous obtenons un effet semblable de courbature. Le résulat crée une ellipse que nous pouvons observer sur la figure 4.2.

La dernière fonction "grid_mouton" de notre fichier grid_2D_Equilibrium fait le même processus que la fonction grid_fct, mais pour toutes les positions retournées de notre fonction mouton_3_corps. la fonction grid_mouton retourne une liste de grid NxN qui pourront être traitées pour former notre graphique 3D qui bouge dans le temps. Une loop à l'intérieur de cette fonction fait le même processus que grid_fct, mais de façon vectorisée. Cela nous évite de faire une triple boucle, mais seulement pouvoir itérer sur le range du nombre de positions que nous avons choisi de traiter avec notre fonction mouton_3_corps.

```python
def grid_mouton(N, equilib, sm, masse_astre):
    lon = np.linspace(-1.0 * np.pi, 1.0 * np.pi, N)
    lat = np.linspace(-0.5 * np.pi, 0.5 * np.pi, N)
    liste = []
    somme_moyenne = 0
    for i in range(len(sm["t"])):
        theta, delta = np.meshgrid(lon, lat)
        theta = theta + (2*np.pi/(24*3600))*sm["t"][i]
        G = np.zeros((N, N))
        G = equilib(sm["L"][i, 0], sm["L"][i, 1], sm["L"][i, 2], sm["S"][i, 0], sm["S"][i, 1], sm["S"][i
        liste.append(G)
        somme_moyenne += G.max() - G.min()
    moyenne = somme_moyenne/len(sm["t"])
    print("Moyenne du marnage: {}".format(moyenne))
    return liste
```

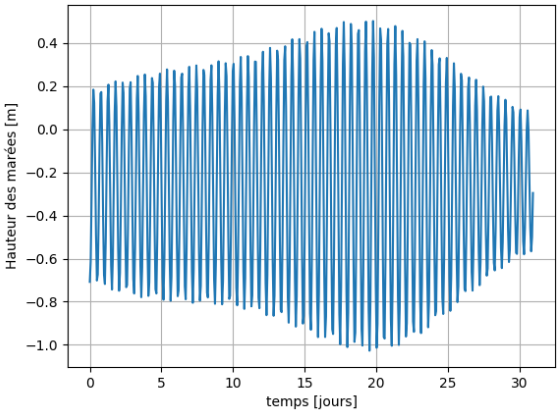
En terme d'efficacité de programme, notre nouvelle fonction plus générale grid_mouton mieux adaptée pour de nombreux calculs. Le fait que nous ne faisons plus de loop sur les coordonnées nous permet de faire le calcul de chaque point en même temps et cela diminue énormément le temps d'exécution de la fonction. Aussi, un petit ajout nous print la moyenne du marnage pour l'entièreté de notre temps d'exécution pour l'astre choisi. Par contre, notre valeur perd de la justesse si nous n'étudions pas environs une période lunaire. Voici les résultats donnés lorsque nous évaluons la moyenne du marnage pour 28 jours :

$$\bar{H}_M = 2.57814749$$
$$\bar{H}_L = 0.67681314$$

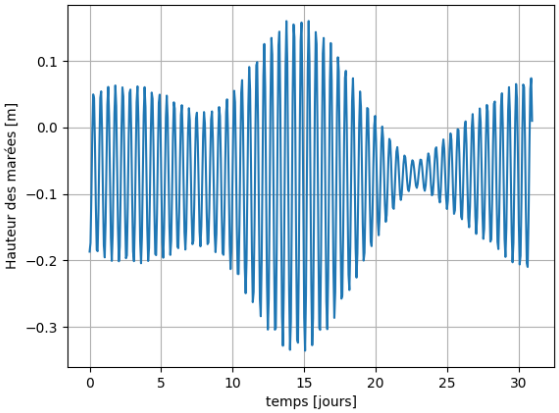
Nous constatons que $\bar{H}_L$ est proche de la valeur théorique énoncée plus haut ($\bar{H} = 0,75$ [m]), à un pourcentage d'écart de 9,758%.

4.3 Fonctions dans Graph_maree(t).py

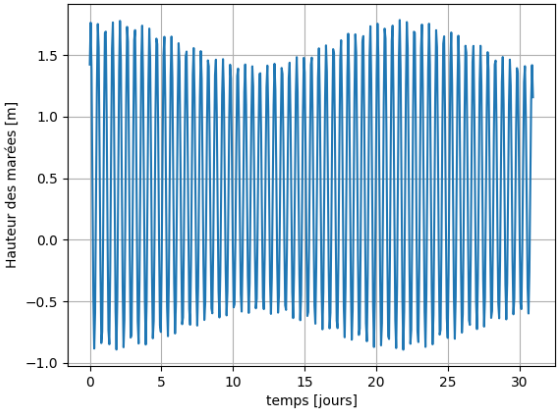
Nous avons maintenant suffisamment de fonctions pour voir la progression des marées en fonction du temps écroulé (et de la position des astres) en un point sur le globe terrestre. La fonction utilisée est graph_maree_1pt prenant en argument la fonction donnant le champ scalaire (Equilibrium dans notre cas), l'argument "sm" qui est notre dictionnaire d'informations retourné par la fonction mouton_3_corps, l'argument "masse_astre" qui dépend de l'astre choisi, et les arguments de longitude et latitude qui viennent fixer un point et fait évoluer son scalaire associé dans le temps. Les figures produites sont :



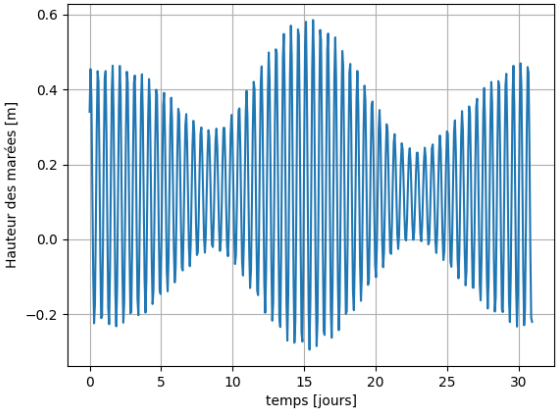
(a) Astre : Mars — Position : Ville de Québec



(b) Astre : Lune — Position : Ville de Québec



(c) Astre : Mars — Position : méridien de Greenwich



(d) Astre : Lune — Position : méridien de Greenwich

FIGURE 4.3 – Affichage des graphiques des hauteurs de marées évoluant dans le temps selon l'astre choisi et la longitude/latitude sur le globe terrestre

Nous constatons que le marnage est beaucoup plus proche d'être constant pour le système avec Mars. Ceci est principalement causé par l'orbite s'approchant davantage d'un cercle que d'une ellipse. Nous voyons aussi pour le système lunaire que pour la figure 4.3 b) n'a pas les deux mêmes minimums (trois corps en quadrature, mortes-eaux). C'est causé par l'orbite lunaire qui passe par l'hémisphère nord dans sa première moitié de rotation et qui finit dans l'hémisphère sud. On voit à la figure d) qu'il n'y a presqu'aucune différence entre les mortes-eaux, car la lune passe très près de l'équateur dans les deux cas. Il est important de noter que notre différence entre la moyenne de marnage de Mars et de la Lune n'est pas proportionnelle avec la différence entre les deux masses de nos satellites. La masse de la lune est environs dix fois moins grande que la masse de Mars, mais pourtant le marnage est seulement quatre fois plus grand. Ceci est essentiellement causé par l'orbite circulaire de Mars, mais surtout car la distance nette entre Mars et la Terre est plus grande que notre distance Terre-Lune quand nous comparons les deux orbites stables. Ainsi, les marées observées avec Mars ne sont pas aussi grandioses que ce dont nous nous attendions.

4.4 Fonctions de 3D_gif.py graphique bougeant dans le temps

Voici les fonctions utilisées pour obtenir notre graphique 3D bougeant dans le temps (Nous utilisons pyvista) :

```
# Crée une grille de cellules prêtes à acueillir une valeur scalaire
def _cell_bounds(points, bound_position=0.1):
    assert points.ndim == 1, "Only 1D points are allowed"
    diffs = np.diff(points)
    delta = diffs[0] * bound_position
    bounds = np.concatenate([[points[0] - delta], points + delta])
    return bounds


def points(lon, lat):
    R = RADIUS
    return np.array(
        [1.01 * R * np.cos(lat) * np.cos(lon), 1.01 * R * np.cos(lat) * np.sin(lon), 1.01 * R * np.sin(l


# Crée des points sur la sphère, étiquettés par leur "label". Ils resteront toujours sur notre sphère, e
label = ["Québec", "Paris", "Tokyo", "Brasilia"]
Quebec = points(-1.2428137, 0.8170563)
Paris = points(0.0409942, 0.85265268)
Tokyo = points(2.440659, 0.622259)
Brasilia = points(-0.8365314, -0.2754080)

Points_array = np.vstack((Quebec, Paris, Tokyo, Brasilia))

# Fonction plotant notre graphique 3Dgif.
def plot_3Dgif(grid_scalaire, grid, clim, nomgif, nom_scalaire):
    p = pv.Plotter()
    p.add_mesh(pv.Sphere(radius=RADIUS), color="white", style="surface")
    p.show_bounds()
    p.add_point_labels(Points_array, label)
    p.add_mesh(grid_scalaire, clim=clim, opacity=0.9, cmap="cividis")

    print('Orienter la vue, et peser "q" pour fermer la fenêtre et produire le gif')

    # on choisit notre angle de caméra
    p.show(auto_close=False)

    # ouvre le gif
    p.open_gif(nomgif)

    # Actualise chaque frame par rapport à la grid suivante dans notre liste d'array de NxN points scal
    for i in range(len(a)):
```

```
43          grid_scalaire.cell_arrays[nom_scalaire] = np.array(grid[i]).swapaxes(-2, -1).ravel("C")
44          p.write_frame()
45
46  # Close movie and delete object
47      p.close()
```

Nous produisons ainsi un gif d'un système 3D qui bouge dans le temps selon une orbite stable calculée à l'aide d'un algorithme fait sur mesure, qui utilise à son tour un algorithme saute-mouton.

Les graphiques 3D sont à voir dans le fichier 3D_gif.py. Il ne suffit que d'exécuter le code, choisir un angle de caméra et peser sur "q" pour observer le gif en action.

5 Conclusion

En somme, nous avons utilisé saute-mouton pour créer un algorithme de mécanique céleste qui trouvait la position et la vitesse d'un système de trois astres pour chaque instant. Nous avons ensuite créé un algorithme qui comparait et calculait une orbite stable pour un système Terre-Mars-Soleil. Nous avons ensuite évalué sur 40000 points sur terre la hauteur des marées avec l'Equilibrium Tide Equation et nous avons fini par faire un graphique en trois dimensions qui change en fonction du temps pour représenter le système total et comparer avec notre modèle d'origine Terre-Lune-Soleil. Nous concluons qu'un modèle Terre-Mars-Soleil pourrait être stable, mais la distance Mars-Terre devrait être supérieure à la distance actuelle entre la Terre et son satellite actuel. Aussi, nos marées seraient environs quatre fois plus fortes, mais les marées de vives-eaux et de mortes-eaux se ressembleraient davantage. Les moyennes de marnage trouvées numériquement et les valeurs théoriques sont semblables à un pourcentage d'écart d'environs 10%.

Références

[1] A. Crida et S. Charnoz. "Formation of Regular Satellites from Ancient Massive Rings in the Solar System". In : *Science* 338.6111 (nov. 2012), p. 1196-1199. DOI : 10.1126/science.1226477. URL : https://doi.org/10.1126/science.1226477.

[2] M. Newman. *Computational Physics*. Createspace Independent Pub, 2012. ISBN : 9781480145511. URL : https://books.google.ca/books?id=SS6uNAEACAAJ.

[3] Jon.D. Giorgini. *JPL Solar System Dynamics*. URL : https://ssd.jpl.nasa.gov/.

[5] G. W. Hill. "Researches in the Lunar Theory". In : *American Journal of Mathematics* 1.1 (1878), p. 5. DOI : 10.2307/2369430. URL : https://doi.org/10.2307/2369430.

[6] Douglas P. Hamilton et Joseph A. Burns. "Orbital stability zones about asteroids". In : *Icarus* 96.1 (mar. 1992), p. 43-64. DOI : 10.1016/0019-1035(92)90005-r. URL : https://doi.org/10.1016/0019-1035(92)90005-r.

[7] G. R. ""Roche's Limit"". In : *Nature* 47.1222 (mar. 1893), p. 509-510. DOI : 10.1038/047509b0. URL : https://doi.org/10.1038/047509b0.

[10] David A. Randall. *The Laplace Tidal Equations and Atmospheric Tides*. Accessed : 11-04-2020.

[12] David Pugh et Philip Woodworth. *Sea-Level Science*. Cambridge University Press, 2014. DOI : 10.1017/cbo9781139235778. URL : https://doi.org/10.1017/cbo9781139235778.

[13] Joris Vanlede, Leen Coen et Maarten Deschamps. "Tidal Prediction in the Sea Scheldt (Belgium) Using a Combination of Harmonic Tidal Prediction and 1D Hydraulic Modeling". In : *Natural Resources* 05.11 (2014), p. 627-633. DOI : 10.4236/nr.2014.511055. URL : https://doi.org/10.4236/nr.2014.511055.

A Fonction *stabilite* du fichier *stabilite.py*

```
def stabilite(temps, N, nom_fichiers, var="x", nombre_fichiers=10):
    stabilite = np.zeros(nombre_fichiers)
    c_init = []
    for n in range(nombre_fichiers):
        mouton = chargement(nom_fichiers+str(n))
        mois = int(31*N//temps)
        liste_mois = np.split(mouton["P"], np.arange(0, N-1, mois))
        liste_mois.pop(0)
        excentricites = np.zeros(len(liste_mois))
        for m in range(len(liste_mois)):
            a = (np.amax(liste_mois[m])-np.amin(liste_mois[m]))/2
            b = np.sqrt(np.median(liste_mois[m])**2 - (np.amin(liste_mois[m])-a)**2)
            excentricites[m] = np.sqrt(1-(a**2/b**2))
        if mouton["valide"] is False:
            stabilite[n] = np.nan
            c_init.append(mouton["c_init"]["Mars"][var])
        else:
            stabilite[n] = 1/np.std(excentricites)
            c_init.append(mouton["c_init"]["Mars"][var])
    return stabilite, c_init
```

B Fonction *recherche_stab* du fichier *stabilite.py*

```
def recherche_stab(t_i, t_f, N, c_init, F, slice=0, nombre=10, ordre=1e7):
    temps = round((t_f-t_i)/(24*3600))
    nombre_fichiers = nombre
    nombre_x, nombre_y = nombre, nombre
    delta_x, delta_y = np.inf, np.inf
    while delta_x >= 1e4 or delta_y >= 1e4:

        #stabilité en x
        var = "x"
        nom_fichiers = "TMS_{}_jours_{}_N_{}".format(temps, N, var)
        c_init_x = c_init["Mars"]["x"]
        nombre_fichiers_x = perturbations(t_i, t_f, N, c_init, F, slice, var="x", nombre=nombre_x, ordre=ordre)
        stab_data_x = stabilite(temps, N, nom_fichiers, var="x", nombre_fichiers=nombre_fichiers_x)
        stab_x = list(stab_data_x[0])
        cond_x = list(stab_data_x[1])
        index_x = stab_x.index(max(stab_x))  # on trouve la position du maximum dans la liste
        c_init["Mars"]["x"] = cond_x[index_x]  # on trouve la condition initiale correspondant au max

        delta_x = np.abs(c_init_x - c_init["Mars"]["x"])
        c_init_x = c_init["Mars"]["x"]
        print("-------\n", "delta_x: ", delta_x, " max: ", cond_x[index_x], "\n-------")

        # stabilité en y
        var = "y"
        nom_fichiers = "TMS_{}_jours_{}_N_{}".format(temps, N, var)
        c_init_y = c_init["Mars"]["y"]
        nombre_fichiers_y = perturbations(t_i, t_f, N, c_init, F, slice, var="y", nombre=nombre_y, ordre=ordre)
        stab_data_y = stabilite(temps, N, nom_fichiers, var="y", nombre_fichiers=nombre_fichiers_y)
        stab_y = list(stab_data_y[0])
        cond_y = list(stab_data_y[1])
        index_y = stab_y.index(max(stab_y))  # on trouve la position du maximum dans la liste
        c_init["Mars"]["y"] = cond_y[index_y]  # on trouve la condition initiale correspondant au max

        print(nombre_y)
        delta_y = np.abs(c_init_y - c_init["Mars"]["y"])
        c_init_y = c_init["Mars"]["y"]
        print("-------\n", "delta_y: ", delta_y, " max: ", cond_y[index_y], "\n-------")

    print(c_init, nom_fichiers, nombre_fichiers_y)
    pickle.dump(c_init, open(str(Path.cwd()/"Données"/"c_init"/nom_fichiers), "w+b"))
    return c_init
```