III Mise en orbite de la sonde autour de Saturne

III.1 Présentation

La sonde Cassini-Huygens a été lancée du site de Cap Canaveral (Floride - USA) le 15 Octobre 1997. Une fusée Titan IV-B/Centaur a été nécessaire pour assurer le lancement de cette sonde de 5,6 tonnes. Aucun lanceur n'étant alors capable de donner à une sonde spatiale de cette masse l'énergie suffisante pour rejoindre Saturne par une trajectoire directe, l'énergie manquante est obtenue en utilisant le phénomène d'*assistance gravitationnelle* de Vénus, de la Terre et enfin de Jupiter. L'assistance gravitationnelle consiste à utiliser le champ de gravitation d'une planète afin de fournir une accélération supplémentaire à la sonde.

Le trajet de la sonde vers Saturne s'est donc effectué en plusieurs étapes comme indiqué sur la **figure 6**.

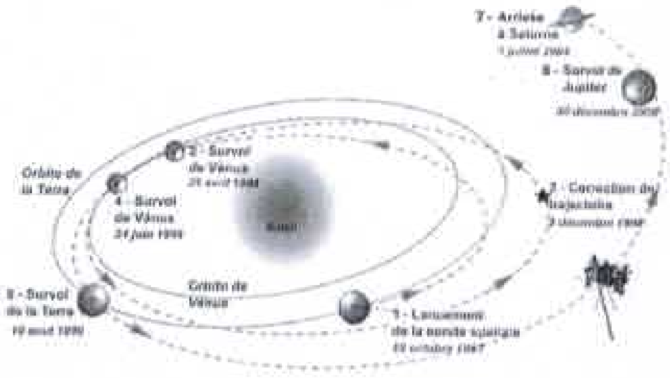

Figure 6 – Trajet de la sonde depuis la Terre vers Saturne

Objectif

L'objectif de cette partie est de simuler le mouvement de la sonde lors de la phase d'assistance gravitationnelle de Vénus afin de déterminer sa trajectoire ainsi que le gain de vitesse obtenu.

III.2 Modélisation du mouvement de la sonde lors de la phase d'assistance gravitationnelle

Paramétrage

Le mouvement de la sonde est étudié dans le référentiel héliocentrique auquel est associé le repère $(O,\vec{x},\vec{y})$.

La sonde est repérée par la position de son centre d'inertie G auquel est associé le vecteur $\vec{r}(t)=\overrightarrow{OG}$ et l'angle $\theta(t)$.

La planète Vénus est repérée par la position de son centre d'inertie G_v auquel est associé le vecteur $\vec{r}_v(t)=\overrightarrow{OG_v}$ et l'angle $\theta_v(t)$.

Hypothèses

- Seuls le Soleil et Vénus ont une influence gravitationnelle significative sur la trajectoire de la sonde.
- Le propulseur de la sonde n'étant pas utilisé lors du survol de Vénus, sa poussée n'est pas prise en compte.
- Les trajectoires de la sonde et de Vénus sont considérées comme coplanaires (plan $O,\vec{x},\vec{y}$). Aussi, les vecteurs position s'expriment en coordonnées cartésiennes :

$$\begin{cases} \vec{r}(t) = x(t)\,\vec{x} + y(t)\,\vec{y} \\ \vec{r}_v(t) = x_v(t)\,\vec{x} + y_v(t)\,\vec{y} \end{cases}$$

- L'orbite de Vénus autour du Soleil, considérée comme quasiment circulaire, est décrite par les équations :

$$\begin{cases} x_v(t) = r_v\cos(\Omega t+\Phi) \\ y_v(t) = r_v\sin(\Omega t+\Phi) \end{cases}$$

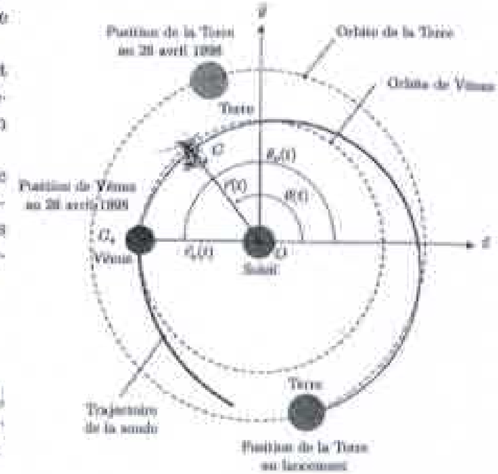

Figure 7 – Paramétrage du mouvement

avec $r_v = 1{,}082\,09\times10^{11}$ m, $\Omega=\dot{\theta}_v(t)=3{,}236\,39\times10^{-7}\,\mathrm{rad\cdot s^{-1}}$ et Φ la position angulaire initiale de Vénus.

Equations de mouvement

Le mouvement de la sonde est défini par l'équation suivante :

$$\frac{\mathrm{d}^2\vec{r}(t)}{\mathrm{d}t^2} = -Gm_s\frac{\vec{r}(t)}{\|\vec{r}(t)\|^3} - Gm_v\frac{\vec{r}(t)-\vec{r}_v(t)}{\|\vec{r}(t)-\vec{r}_v(t)\|^3} \qquad (3)$$

avec G constante universelle de gravitation, m_s masse du Soleil, m_v masse de Vénus ($Gm_S = 1{,}327\,24\times10^{20}\,\mathrm{N\cdot m^2\cdot kg^{-1}}$ et $Gm_V = 3{,}249\,16\times10^{14}\,\mathrm{N\cdot m^2\cdot kg^{-1}}$). On note m la masse de la sonde.

Q17. Indiquer comment a été obtenue cette équation en précisant le théorème utilisé et ce que représentent chacun des termes.

L'équation précédente peut se mettre sous la forme :

$$\frac{\mathrm{d}^2x(t)}{\mathrm{d}t^2} = S_x(x(t),y(t),t) \qquad (4)$$

$$\frac{\mathrm{d}^2y(t)}{\mathrm{d}t^2} = S_y(x(t),y(t),t) \qquad (5)$$

Q18. Donner les expressions des fonctions S_x et S_y. Ecrire une fonction `eval_sm(x,y,t)`, qui prend en argument les coordonnées x et y et l'instant t et qui renvoie les valeurs S_x et S_y associées.

III.3 Résolution numérique

III.3.1 Méthode numérique

La résolution numérique des équations différentielles (4) et (5) repose sur leur discrétisation temporelle et conduit à déterminer à différents instants t_i une approximation de la solution.

On note u_i et v_i, les approximations des composantes du vecteur vitesse $(v_x(t), v_y(t))$ à l'instant t_i avec $v_x(t) = \dfrac{dx(t)}{dt}$ et $v_y(t) = \dfrac{dy(t)}{dt}$. On note $\Delta t = t_{i+1} - t_i$.

Q19. En appliquant la méthode d'Euler explicite, déterminer les deux relations de récurrence reliant u_{i+1} à u_i, S_x et Δt ainsi que v_{i+1} à v_i, S_y et Δt.

On note x_i et y_i, les approximations des composantes du vecteur position $(x(t), y(t))$ à l'instant t_i.

Q20. En appliquant la méthode d'Euler explicite, déterminer les deux relations de récurrence reliant x_{i+1} à x_i, u_i et Δt ainsi que y_{i+1} à y_i, v_i et Δt.

III.3.2 Le programme de résolution

On supposera toutes les constantes du problème définies comme variables globales et donc utilisables directement dans votre programme.

Les conditions initiales du problème seront supposées être :

$$u_0 = v_x(t=0),\ v_0 = v_y(t=0),\ x_0 = x(t=0),\ y_0 = y(t=0).$$

Le programme doit permettre de calculer les quatre vecteurs : $x(t)$, $y(t)$, $u(t)$ et $v(t)$, contenant les différentes valeurs pour les différents instant t_i, qui seront stockés respectivement dans les variables x, y, u et v.

Le programme est constitué de trois étapes : initialisation des variables, résolution et traitement.

On définira également le vecteur *temps* contenant l'ensemble des instants de calcul t_i de longueur n et qui sera stocké dans la variable t.

Q21. Donner les lignes de programme permettant de définir et d'initialiser les quatre variables x, y, u et v ainsi que le vecteur t contenant l'ensemble des instants de calcul t_i de longueur n sachant que la variable deltaT égale à Δt et la variable n sont déjà déclarées.

Q22. Donner les lignes de programme permettant de calculer les vecteurs x, y, u et v en utilisant les relations de récurrence définies précédemment.

Q23. Donner un ordre de grandeur de la quantité de mémoire nécessaire pour réaliser cette simulation numérique pour une durée de simulation de 30 jours et pour un pas de temps de 1 seconde sachant que les variables x, y, u et v contiennent des flottants codés sur 8 octets. Conclure sur la faisabilité de la simulation par cette méthode.

III.4 Evolution de la vitesse

Q24. Afin de quantifier le gain de vitesse obtenu par l'assistance gravitationnelle étudiée, écrire une fonction vitesse_sonde dont vous préciserez les arguments et les valeurs retournées, permettant d'afficher l'évolution de la norme de la vitesse $\vec{v}(t)$ de la sonde en fonction du temps t. Les vitesses seront exprimées en $km \cdot s^{-1}$ et les axes seront légendés. On pourra utiliser les informations données en annexe pour l'utilisation des commandes de tracés.

L'étude de la propagation des épidémies joue un rôle important dans les politiques de santé publique. Les modèles mathématiques ont permis de comprendre pourquoi il a été possible d'éradiquer la variole à la fin des années 1970 et pourquoi il est plus difficile d'éradiquer d'autres maladies comme la poliomyélite ou la rougeole. Ils ont également permis d'expliquer l'apparition d'épidémies de grippe tous les hivers. Aujourd'hui, des modèles de plus en plus complexes et puissants sont développés pour prédire la propagation d'épidémies à l'échelle planétaire telles que le SRAS, le virus H5N1 ou le virus Ebola. Ces prédictions sont utilisées par les organisations internationales pour établir des stratégies de prévention et d'intervention.

Le travail sur ces modèles mathématiques s'articule autour de trois thèmes principaux : traitement de bases de données, simulation numérique (par plusieurs types de méthodes), identification des paramètres intervenant dans les modèles à partir de données expérimentales. Ces trois thèmes sont abordés dans le sujet. *Les parties sont indépendantes.*

Dans tout le problème, on peut utiliser une fonction traitée précédemment. On suppose que les bibliothèques numpy et random ont été importées par :

```
import numpy as np
import random as rd
```

Partie I. Tri et bases de données

Dans le but ultérieur de réaliser des études statistiques, on souhaite se doter d'une fonction tri. On se donne la fonction tri suivante, écrite en Python :

```
def tri(L):
    n = len(L)
    for i in range(1, n):
        j = i
        x = L[i]
        while 0 < j and x < L[j-1]:
            L[j] = L[j-1]
            j = j-1
        L[j] = x
```

❑ **Q1** – Lors de l'appel tri(L) lorsque L est la liste [5, 2, 3, 1, 4], donner le contenu de la liste L à la fin de chaque itération de la boucle for.

❑ **Q2** – Soit L une liste non vide d'entiers ou de flottants. Montrer que « la liste L[0:i+1] (avec la convention Python) est triée par ordre croissant à l'issue de l'itération i » est un invariant de boucle. En déduire que tri(L) trie la liste L.

❑ **Q3** – Évaluer la complexité dans le meilleur et dans le pire des cas de l'appel tri(L) en fonction du nombre n d'éléments de L. Citer un algorithme de tri plus efficace dans le pire des cas. Quelle en est la complexité dans le meilleur et dans le pire des cas ?

On souhaite, partant d'une liste constituée de couples (chaîne, entier), trier la liste par ordre croissant de l'entier associé suivant le fonctionnement suivant :

```
>>> L = [['Bresil', 76], ['Kenya', 26017], ['Ouganda', 8431]]
>>> tri_chaine(L)
>>> L
[['Bresil', 76], ['Ouganda', 8431], ['Kenya', 26017]]
```

❑ Q4 - Écrire en Python une fonction, tri_chaine réalisant cette opération.

Partie II. Modèle à compartiments

On s'intéresse ici à une première méthode de simulation numérique.

Les modèles compartimentaux sont des modèles déterministes où la population est divisée en plusieurs catégories selon leurs caractéristiques et leur état par rapport à la maladie. On considère dans cette partie un modèle à quatre compartiments disjoints : sains (S, "susceptible"), infectés (I, "infected"), rétablis (R, "recovered", ils sont immunisés) et décédés (D, "dead"). Le changement d'état des individus est gouverné par un système d'équations différentielles obtenues en supposant que le nombre d'individus nouvellement infectés (c'est-à-dire le nombre de ceux qui quittent le compartiment S) pendant un intervalle de temps donné est proportionnel au produit du nombre d'individus infectés avec le nombre d'individus sains.

En notant $S(t)$, $I(t)$, $R(t)$ et $D(t)$ la fraction de la population appartenant à chacune des quatre catégories à l'instant t, on obtient le système :

$$\left.\begin{aligned} &\frac{d}{dt}S(t) = -r\,S(t)I(t)\\ &\frac{d}{dt}I(t) = r\,S(t)I(t) - (a+b)\,I(t)\\ &\frac{d}{dt}R(t) = a\,I(t)\\ &\frac{d}{dt}D(t) = b\,I(t) \end{aligned}\right\} \qquad (1)$$

avec r le taux de contagion, a le taux de guérison et b le taux de mortalité. On suppose qu'à l'instant initial $t = 0$, on a $S(0) = 0{,}95$, $I(0) = 0{,}05$ et $R(0) = D(0) = 0$.

❑ **Q10** – Préciser un vecteur X et une fonction f (en donnant son domaine de définition et son expression) tels que le système différentiel (1) s'écrive sous la forme

$$\frac{d}{dt}X = f(X).$$

❑ **Q11** – Compléter la ligne 4 du code suivant (on précise que `np.array` permet de créer un tableau numpy à partir d'une liste donnant ainsi la possibilité d'utiliser les opérateurs algébriques).

```
1   def f(X):
2       """ Fonction definissant l'equation differentielle """
3       global r, a, b
4       # a completer
5
6   # Parametres
7   tmax = 25.
8   r = 1.
9   a = 0.4
10  b = 0.1
11  X0 = np.array([0.95, 0.05, 0., 0.])
12
13  N = 250
14  dt = tmax/N
15
16  t = 0
17  X = X0
18  tt = [t]
19  XX = [X]
20
21  # Methode d'Euler
22  for i in range(N):
23      t = t + dt
24      X = X + dt * f(X)
25      tt.append(t)
26      XX.append(X)
```


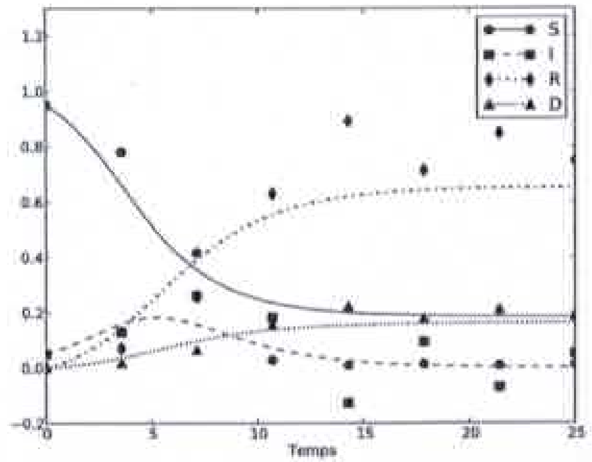

FIGURE 1 – Représentation graphique des quatre catégories S, I, R et D en fonction du temps pour $N = 7$ (points) et $N = 250$ (courbes).

❑ **Q12** – La figure 1 représente les quatre catégories en fonction du temps obtenues en effectuant deux simulations : la première avec $N = 7$ correspond aux points (cercle, carré, losange, triangle) et la seconde avec $N = 250$ correspond aux courbes. Expliquer la différence entre ces deux simulations. Quelle simulation a nécessité le temps de calcul le plus long ?

En pratique, de nombreuses maladies possèdent une phase d'incubation pendant laquelle l'individu est porteur de la maladie mais ne possède pas de symptômes et n'est pas contagieux. On peut prendre en compte cette phase d'incubation à l'aide du système à retard suivant :

$$\left\{\begin{aligned} &\frac{d}{dt}S(t) = -r\,S(t)I(t-\tau)\\ &\frac{d}{dt}I(t) = r\,S(t)I(t-\tau) - (a+b)\,I(t)\\ &\frac{d}{dt}R(t) = a\,I(t)\\ &\frac{d}{dt}D(t) = b\,I(t) \end{aligned}\right.$$

où τ est le temps d'incubation. On suppose alors que pour tout $t \in [-\tau, 0]$, $S(t) = 0{,}95$, $I(t) = 0{,}05$ et $R(t) = D(t) = 0$.

En notant $tmax$ la durée des mesures et N un entier donnant le nombre de pas, on définit le pas de temps $dt = tmax/N$. On suppose que $\tau = p \times dt$ où p est un entier ; ainsi p est le nombre de pas du retard.

Pour résoudre numériquement ce système d'équations différentielles à retard (avec `tmax = 25`, `N = 250` et `p = 50`), on a écrit le code suivant :

```
1   def f(X, Itau):
2       """
3       Fonction definissant l'equation differentielle
4       Itau est la valeur de I(t - p * dt)
5       """
6       global r, a, b
7       # a completer
8
9   # Parametres
10  r = 1.
11  a = 0.4
12  b = 0.1
13  X0 = np.array([0.95, 0.05, 0., 0.])
14
15  tmax = 25.
16  N = 250
17  dt = tmax/N
18  p = 50
19
20  t = 0
21  X = X0
22  tt = [t]
23  XX = [X]
24
25  # Methode d'Euler
26  for i in range(N):
27      t = t + dt
28      # a completer
29      tt.append(t)
30      XX.append(X)
```

❑ **Q13** – Compléter les lignes 7 et 28 du code précédent (utiliser autant de lignes que nécessaire).

On constate que le temps d'incubation de la maladie n'est pas nécessairement le même pour tous les individus. On peut modéliser cette diversité à l'aide d'une fonction positive d'intégrale unitaire (dite de densité) $h : [0, \tau] \to \mathbb{R}_+$ telle que représentée sur la figure 2. On obtient alors le système intégro-différentiel :

$$\begin{cases} \dfrac{d}{dt}S(t) = -r\,S(t)\displaystyle\int_0^\tau I(t-s)h(s)\,ds \\ \dfrac{d}{dt}I(t) = r\,S(t)\displaystyle\int_0^\tau I(t-s)h(s)\,ds - (a+b)\,I(t) \\ \dfrac{d}{dt}R(t) = a\,I(t) \\ \dfrac{d}{dt}D(t) = b\,I(t) \end{cases}$$

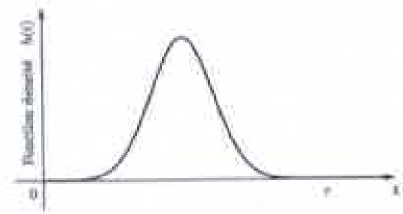

FIGURE 2 – Exemple d'une fonction de densité

On supposera à nouveau que pour tout $t \in [-\tau, 0]$, $S(t) = 0{,}95$, $I(t) = 0{,}05$ et $R(t) = D(t) = 0$. Pour j entier compris entre 0 et N, on pose $t_j = j \times dt$. Pour un pas de temps dt donné, on peut calculer numériquement l'intégrale à l'instant t_i ($0 \le i \le N$) à l'aide de la méthode des rectangles à gauche en utilisant l'approximation :

$$\int_0^\tau I(t_i - s)h(s)\,ds \approx \mathrm{dt} \times \sum_{j=0}^{p-1} I(t_i - t_j)h(t_j).$$

❑ **Q14** – On suppose que la fonction `h` a été écrite en `Python`. Expliquer comment modifier le programme de la question précédente pour résoudre ce système intégro-différentiel (on explicitera les lignes de code nécessaires).

Partie III. Modélisation dans des grilles

On s'intéresse ici à une seconde méthode de simulation numérique (dite par *automates cellulaires*).

Dans ce qui suit, on appelle *grille de taille* $n \times n$ une liste de n listes de longueur n, où n est un entier strictement positif.

Pour mieux prendre en compte la dépendance spatiale de la contagion, il est possible de simuler la propagation d'une épidémie à l'aide d'une grille. Chaque case de la grille peut être dans un des quatre états suivants : saine, infectée, rétablie, décédée. On choisit de représenter ces quatre états par les entiers :

0 (Sain), 1 (Infecté), 2 (Rétabli) et 3 (Décédé).

L'état des cases d'une grille évolue au cours du temps selon des règles simples. On considère un modèle où l'état d'une case à l'instant $t+1$ ne dépend que de son état à l'instant t et de l'état de ses huit cases voisines à l'instant t (une case du bord n'a que cinq cases voisines et trois pour une case d'un coin). Les *règles de transition* sont les suivantes :

- une case décédée reste décédée ;
- une case infectée devient décédée avec une probabilité p_1 ou rétablie avec une probabilité $(1 - p_1)$;
- une case rétablie reste rétablie ;
- une case saine devient infectée avec une probabilité p_2 si elle a au moins une case voisine infectée et reste saine sinon.

On initialise toutes les cases dans l'état sain, sauf une case choisie au hasard dans l'état infecté.

❑ **Q15** – On a écrit en `Python` la fonction `grille(n)` suivante