CONCOURS COMMUNS POLYTECHNIQUES

EPREUVE SPECIFIQUE - FILIERE PSI

INFORMATIQUE

Vendredi 6 mai : 8 h - 11 h

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Les calculatrices sont interdites

Le sujet comporte 12 pages dont :
- 11 pages de texte de présentation et énoncé du sujet ;
- 1 page d'annexe.

Toute documentation autre que celle fournie est interdite.

REMARQUES PRELIMINAIRES

L'épreuve peut être traitée en langage **Python** ou langage **Scilab**. Il est demandé au candidat de bien préciser sur sa copie le choix du langage et de rédiger l'ensemble de ses réponses dans ce langage. Les syntaxes Python et Scilab sont rappelées en annexe, page 12.

Les différents algorithmes doivent être rendus dans leur forme définitive sur la copie en respectant les éléments de syntaxe du langage choisi (les brouillons ne sont pas acceptés).

Il est demandé au candidat de bien vouloir rédiger ses réponses **en précisant bien le numéro de la question traitée et, si possible, dans l'ordre des questions.** La réponse ne doit pas se cantonner à la rédaction de l'algorithme sans explication, les programmes doivent être expliqués et commentés.

La mission Cassini-Huygens

I Introduction

La mission spatiale Cassini-Huygens a pour objectif l'exploration de Saturne et de ses nombreux satellites naturels (62 « lunes » identifiées en 2009). Cassini orbite depuis 2004 autour de Saturne et collecte grâce à ses différents instruments d'observation, de précieuses informations sur la planète géante, ses anneaux caractéristiques et ses différents satellites.

Le sujet proposé aborde des problématiques associées à différentes phases de la mission :

- l'acquisition d'images par l'imageur spectral VIMS (Visible and Infrared Mapping Spectrometer) dans le domaine du visible et de l'infrarouge embarqué à bord de la sonde Cassini et la compression des données avant transmission vers la Terre pour leur exploitation,
- la mise en orbite autour de Saturne de la sonde, en utilisant le phénomène d'assistance gravitationnelle de Vénus, de la Terre, puis de Jupiter.

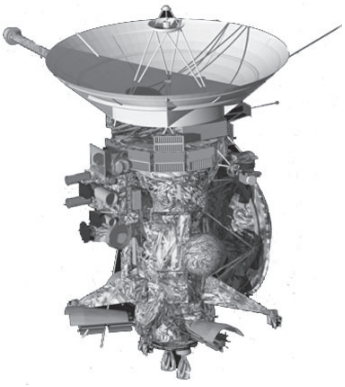

Figure 1 – La sonde Cassini-Huygens

II Acquisition d'images par l'imageur VIMS et compression des données

II.1 Présentation

La sonde Cassini embarque à son bord un ensemble de 12 instruments scientifiques destinés à l'étude de Saturne et son système, dont 4 instruments d'observation à distance permettant de couvrir une bande spectrale d'observation très large (**figures 2 et 3**).

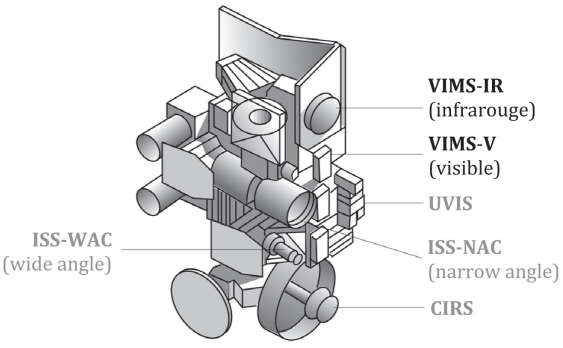

Figure 2 – Instruments d'observation à distance

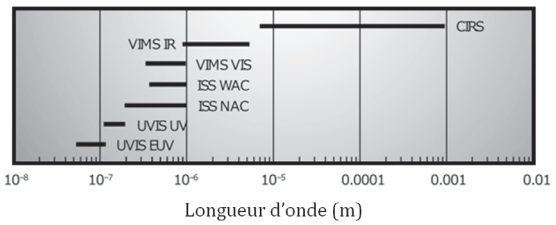

Figure 3 – Domaine de travail des différents instruments d'observation à distance

L'un des principaux objectifs scientifiques associés à l'instrument VIMS est de cartographier la distribution spatiale des caractéristiques minéralogiques et chimiques de différentes cibles : anneaux de Saturne, surface de ses satellites, atmosphère de Saturne et Titan, etc. Pour cela, l'instrument VIMS mesure les radiations émises et réfléchies par les corps observés, sur une gamme de 0,35 à 5,1 µm (domaines visible et infrarouge) avec au total 352 longueurs d'onde différentes.

Les données acquises par l'instrument sont organisées sous la forme d'un cube (**figure 4**) constitué d'un ensemble de 352 « tranches » (associées aux différentes longueurs d'onde λ) comportant 64 pixels dans les deux directions spatiales x et y. Il est ensuite possible d'en extraire une image à une longueur d'onde donnée (λ_k) ou un spectre associé à un pixel de coordonnées spatiales (x_i,y_j).

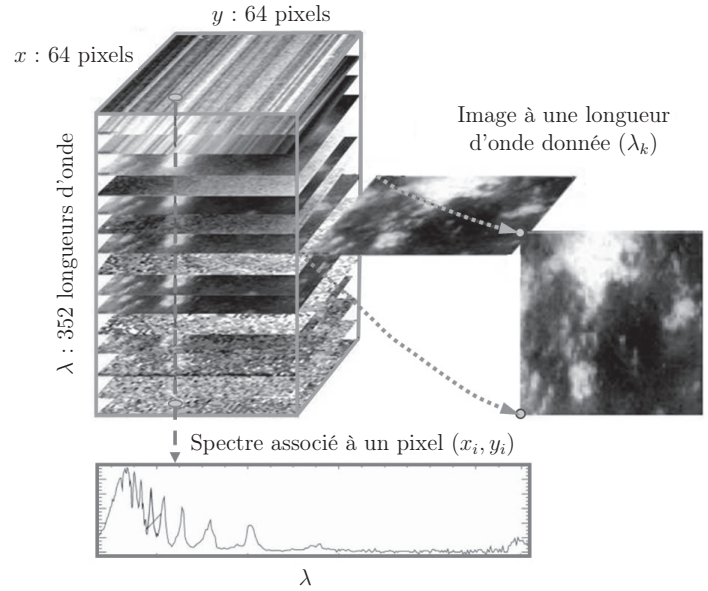

Figure 4 – Structure d'un cube de données hyperspectrales

Les contraintes technologiques liées au transfert de l'ensemble des données recueillies par la sonde vers la Terre imposent une réduction de leur volume. Pour l'imageur VIMS, cela consiste en une *compression* des données prise en charge par une unité de traitement numérique embarquée à bord de la sonde. Cette compression doit toutefois s'effectuer *sans perte d'information.* Après compression, la taille maximale attendue pour un cube de données est de 1 Mo (méga-octet).

Objectif

Cette partie s'intéresse à l'implémentation d'algorithmes spécifiques visant à améliorer les performances de la compression en vue d'une mise à jour des programmes de l'unité de traitement embarquée.

A chaque pixel p_{ijk} du cube de données hyperspectrales (de coordonnées x_i, y_j et appartenant à une image de longueur d'onde λ_k) est associé un nombre moyen de photons reçus par les capteurs de l'instrument VIMS. Cette dernière information est codée sur 12 bits.

Q1. Déterminer en octets la taille d'un cube de données hyperspectrales avant compression.

Si T et T_c représentent respectivement la taille des données avant compression et après compression, le taux de compression τ est défini comme : $\tau = \frac{T-T_c}{T}$.

Q2. Déterminer alors le taux de compression à appliquer aux données afin de respecter le cahier des charges.

II.2 Principe de la compression des données

La compression des données est réalisée à l'issue de l'acquisition d'une ligne (soit 64 pixels pour la dimension spatiale et 352 longueurs d'onde pour la dimension spectrale) par l'imageur

VIMS. Les données sont traitées par blocs de 64 pixels × 32 longueurs d'onde. L'algorithme mis en œuvre pour la compression est un algorithme dit entropique, qui réalise un codage des données exploitant la redondance de l'information. L'exemple proposé ci-après permet d'en illustrer le principe, avec le codage d'une suite de 10 entiers naturels.

Soit la série S_n de 10 entiers naturels $s_k \in [0,8]$: $4-5-7-0-7-8-4-1-7-4$.
Un codage en binaire naturel des entiers 0 à 8 nécessitant au minimum 4 bits, le choix d'un tel codage pour la série S_n conduirait à un code d'une longueur totale de 40 bits.
La longueur du code associé à la série S_n peut être réduite en adoptant un codage exploitant les propriétés suivantes de la série :

– seules 6 valeurs v_i sont utilisées parmi les 9 possibles ;

– certaines valeurs v_i possèdent des probabilités p_i d'apparition plus importantes que d'autres.

Une solution consiste par exemple à adopter un codage à longueur variable, où les codes les plus courts sont associés aux valeurs qui possèdent la probabilité d'apparition la plus importante. Un tel codage est proposé dans le **tableau 1** pour les entiers de la série S_n. La série S_n est alors codée de la manière suivante :

4	5	7	0	7	8	4	1	7	4
0 0	1 1 0	1 0	0 1 0	1 0	1 1 1	0 0	0 1 1	1 0	0 0

La longueur du code associé à la série est à présent de 24 bits (soit une longueur moyenne de 2,4 bits par caractère). Le décodage est unique, mais il nécessite la connaissance de la table de codage établissant la correspondance entre un code et la valeur associée.

C'est cette stratégie qui est adoptée pour les opérations de compression. Pour la série S_n considérée, avec des valeurs initialement codées sur 4 bits en binaire naturel, le codage à longueur variable présenté permet d'obtenir un taux de compression $\tau = 40\,\%$.

valeur v_i	probabilité p_i	code
4	0,3	0 0
7	0,3	1 0
0	0,1	0 1 0
1	0,1	0 1 1
5	0,1	1 1 0
8	0,1	1 1 1

Tableau 1 – Table de codage

II.3 Limite du taux de compression

Les algorithmes de compression entropiques étant basés sur l'exploitation de la redondance dans les données à coder, leur efficacité est directement liée à la « quantité d'information » qu'elles contiennent : plus il y a répétitions de certaines valeurs dans les données à coder (l'entropie des données est alors faible), plus le taux de compression atteint est élevé.

Une valeur limite du taux de compression que l'on peut espérer atteindre peut être approchée par le calcul de l'entropie de Shannon H, grandeur fournissant une mesure de la « quantité d'information » contenue dans les données, en bits par caractère. Pour une série de données S_n (considérée comme une variable aléatoire) l'entropie H est définie de la manière suivante :

$$H(S_n) = -\sum_{i=1}^{i=N_v} p_i \log_2 p_i, \text{ avec} \tag{1}$$

- N_v : nombre de valeurs v_i différentes contenues dans la série ;
- p_i : probabilité d'apparition de la valeur v_i ($p_i = \frac{n_i}{n}$ avec n_i nombre d'occurrences de la valeur v_i et n nombre de termes de la série S_n) ;
- $\log_2$: fonction logarithme binaire (base 2). Pour $x \in \mathcal{R}$, $\log_2(x) = \frac{\ln(x)}{\ln(2)}$ où ln est la fonction logarithme népérien. On suppose que la fonction `log2(x)` est définie dans le langage de programmation choisi.

Q3. Calculer l'entropie associeé à l'exemple précédent. En déduire le taux de compression limite de cet exemple et le comparer à la longueur moyenne de 2,4 bits par caractère.

Dans le cadre d'une étude visant à améliorer les performances de la compression des données, les ingénieurs souhaitent caractériser l'entropie des images acquises par l'instrument VIMS. La fonction `entropie(S)` dont le code est partiellement présenté ci-après reçoit en argument d'entrée une série d'entiers naturels S sous forme de tableau à une dimension (liste en Python) et renvoie la valeur H de l'entropie de Shannon associée.

Code Python de la fonction `entropie(S)` :

```
### DEFINITION DE LA FONCTION ENTROPIE
def entropie(S):
# 1.COMMENTAIRE1 à compléter à la question Q4
    valeurs=list(set(S))
# 2.Détermination du nombre d'occurences
  (nombre d'apparitions) occ_i de chaque
  valeur v_i et calcul de la probabilité
  proba[i] associée.
    # QUESTION Q5 : zone à compléter
# 3.Calcul de l'entropie de Shannon H
    # QUESTION Q6 : zone à compléter
    return H
```

Code Scilab de la fonction `entropie(S)` :

```
// DEFINITION DE LA FONCTION ENTROPIE
function H=entropie(S)
// 1. COMMENTAIRE1 à compléter à la question Q4
    valeurs=unique(S)
// 2.Détermination du nombre d'occurences
  (nombre d'apparitions) occ_i de chaque
  valeur v_i et calcul de la probabilité
  proba[i] associée.
    // QUESTION Q5 : zone à compléter
// 3.Calcul de l'entropie de Shannon H
    // QUESTION Q6 : zone à compléter
endfunction
```

La fonction réalise différentes étapes. La première étape utilise la fonction `set` en Python ou `unique` en Scilab. Un exemple extrait de la documentation de la fonction `set` est donnée ci-dessous. En Python, il faut transformer le résultat en `list` pour itérer sur le résultat de `set` (le principe est similaire en Scilab avec la fonction `unique`).

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> fruit = set(basket)               # create a set without duplicates
>>> fruit
set(['orange', 'pear', 'apple', 'banana'])
>>> 'orange' in fruit                 # fast membership testing
True
>>> 'crabgrass' in fruit
False
```

Q4. Compléter le commentaire 1 de la fonction `entropie(S)` correspondant à la ligne de programme définissant la variable `valeurs`.

Q5. Compléter la 2e étape de la fonction `entropie(S)` à partir du commentaire afin de calculer les probabilités p_i.

Q6. Compléter la 3e étape de la fonction `entropie(S)` afin de calculer l'entropie de Shannon H définie par l'équation (1).

Suite à une acquisition d'image par l'instrument VIMS, l'utilisateur dispose de données brutes dont il souhaite évaluer l'entropie. Un bloc élémentaire de données se présente sous la forme d'un tableau à une dimension `bloc_image` constitué de valeurs entières (allant de 0 à $4\,095 = 2^{12} - 1$).

Q7. Ecrire la suite d'instructions permettant de calculer et d'afficher la valeur du taux de compression limite τ associé aux données contenues dans le tableau `bloc_image`.

II.4 Prétraitement des données avant compression

Les données brutes sont traitées par blocs correspondants à une acquisition de 64 pixels (dimension spatiale y) par 32 longueurs d'onde (dimension spectrale λ). Ces données sont organisées sous forme d'une matrice `donnees_brutes` composée de 32 lignes et 64 colonnes.

Une stratégie efficace permettant d'améliorer la compression consiste à réaliser un prétraitement des données visant à réduire leur entropie. Il a été constaté pour les images acquises par l'instrument VIMS que les importantes variations de luminance (intensité lumineuse par unité de surface) constituent la contribution dominante dans l'entropie des données. Ainsi, le prétraitement consiste à estimer ces variations, puis à les soustraire aux données brutes à compresser afin de constituer un nouvel ensemble de données, d'entropie inférieure.

La procédure de prétraitement consiste d'abord à construire une matrice modèle `matrice_modele` permettant d'estimer les variations de luminance associées au bloc de données brutes, suivant les étapes décrites ci-après :

1. construction d'une ligne `luminance_moy` représentative de la luminance moyenne, en effectuant la moyenne de 4 lignes de la matrice `donnees_brutes` régulièrement espacées et associées à différentes longueurs d'ondes $\lambda_i,\ i = 1,\ 11,\ 21,\ 31$;
2. identification du pixel de luminance maximale (de valeur `luminance_max`) dans la ligne moyenne `luminance_moy` construite à l'étape précédente ;
3. extraction d'un vecteur colonne `spectre_max` contenant le spectre (32 composantes) associé au pixel de luminance maximale ;
4. normalisation du vecteur `spectre_max` en divisant (division réelle) chacune de ses composantes par la valeur `luminance_max` relevée à l'étape 2. Le résultat est une liste contenant des réels ;
5. construction de la matrice modèle `matrice_modele` de dimension (32 lignes, 64 colonnes) en effectuant le produit du vecteur colonne `spectre_max` par la ligne de luminance moyenne `luminance_moy`.

Q8. A partir de la description des étapes **1** et **2**, écrire une fonction `pretraitement12(donnees_brutes)` recevant en argument d'entrée la matrice `donnees_brutes` et renvoyant le vecteur ligne `luminance_moy`, la valeur de luminance maximale `luminance_max` relevée pour cette ligne moyenne, ainsi que l'indice j (variable `j_max`) associé (sans utiliser la fonction `max` interne au langage).

Q9. A partir de la description des étapes **3** et **4**, écrire une fonction `pretraitement34(donnees_brutes, luminance_max,j_max)` recevant en argument d'entrée la matrice `donnees_brutes`, la valeur de luminance maximale `luminance_max` ainsi que l'indice `j_max` et renvoyant le vecteur colonne `spectre_max` normalisé.

Q10. A partir de la description de l'étape **5**, écrire une fonction `pretraitement5(luminance_moy, spectre_max)` recevant en argument d'entrée le vecteur ligne `luminance_moy`, le vecteur colonne `spectre_max` et renvoyant la matrice modèle `matrice_modele`.

La matrice modèle `matrice_modele` obtenue constitue une estimation des variations de luminance associées au bloc de données brutes. Elle est alors soustraite à la matrice `donnees_brutes` pour construire la matrice de données prétraitées `matrice_pretraitee`.

Q11. Evaluer la complexité calculatoire (additions, soustractions, ...) associée à chacune des fonctions de prétraitement, en fonction des dimensions de la matrice modèle (n lignes, m colonnes). En déduire la complexité calculatoire de la phase de prétraitement.

II.5 Compression des données

Les données de la matrice `matrice_pretraitee`, d'entropie réduite, sont ensuite envoyées vers le compresseur qui prend en charge leur compression. Il en est de même pour les vecteurs

`luminance_moy` et `spectre_max` qui sont nécessaires à la reconstruction des données après réception. L'architecture fonctionnelle du compresseur qui prend en charge ces opérations est décrite sur la **figure 5**.

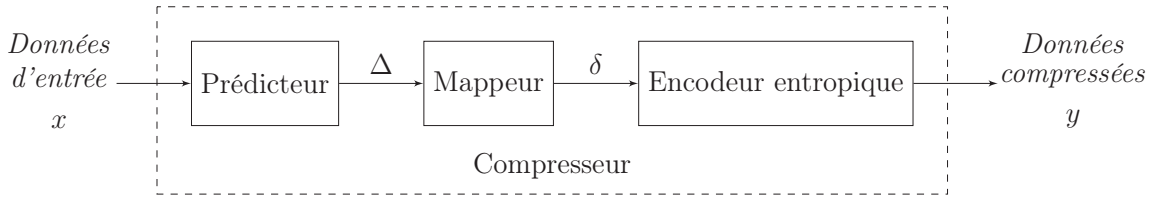

Figure 5 – Architecture fonctionnelle du compresseur

Le compresseur est composé :
- d'un *prédicteur*, qui effectue une prédiction $\hat{x}_k$ pour chaque échantillon x_k des données d'entrée x. La prédiction $\hat{x}_k$ est réalisée simplement ici, en prenant la valeur x_{k-1} de l'échantillon précédent. Le prédicteur retourne l'erreur Δ_k, différence entre la valeur de l'échantillon x_k et la prédiction $\hat{x}_k$;
- d'un *mappeur*, qui réalise un mappage (correspondance) entre l'erreur de prédiction Δ_k et un entier non signé δ_k représenté avec un nombre de bits équivalent à celui de l'échantillon x_k ;
- d'un *encodeur entropique*, qui réalise un codage sans perte des valeurs δ_k en exploitant la redondance des données tel que présenté en **partie II.2**, page 3.

II.5.1 Prédiction-mappage

Le sous-ensemble prédicteur-mappeur a pour fonction d'associer aux échantillons d'entrée x_k une série d'entiers δ_k. Cette association est réalisée de telle sorte que les valeurs δ_k les plus faibles (donc celles qui pourront être codées avec le moins de bits) sont celles qui ont la probabilité d'apparition la plus élevée.

Pour l'application proposée ici, le compresseur reçoit en entrée les données x constituées d'un paquet de 32 échantillons x_k. Les échantillons x_k considérés étant des entiers naturels représentés sur 12 bits, les valeurs associées appartiennent à l'intervalle $I = [0\ ,\ 4\,095 = 2^{12}-1]$.

Le **tableau 2** illustre les opérations réalisées par le prédicteur et le mappeur permettant d'obtenir la série de valeur δ_k :
- le premier échantillon $x_1 = 2\,025$ est un échantillon de référence sur lequel se base la prédiction au rang suivant $\hat{x}_2$;

k	x_k	$\hat{x}_k$	Δ_k	θ_k	δ_k
1	2 025	—	—	—	—
2	2 027	2 025	2	2 025	4
3	2 032	2 027	5	2 027	10
4	2 041	2 032	9	2 032	18
5	2 050	2 041	9	2 041	18
6	2 053	2 050	3	2 045	6
7	2 052	2 053	−1	2 042	1
8	2 050	2 052	−2	2 043	3
⋮	⋮	⋮	⋮	⋮	⋮

Tableau 2 – Illustration de la prédiction et du mappage de l'erreur associée

- à partir du deuxième échantillon, l'erreur de prédiction $\Delta_k = x_k - \hat{x}_k$ est calculée ;
- un entier non signé δ_k est associé à l'erreur de prédiction Δ_k par la fonction de mappage définie comme :

$$\delta_k = \begin{cases} 2\Delta_k & \text{si } 0 \leq \Delta_k \leq \theta_k \\ 2|\Delta_k| - 1 & \text{si } -\theta_k \leq \Delta_k < 0 \\ \theta_k + |\Delta_k| & \text{sinon} \end{cases} \quad (2)$$

avec $\theta_k = \min(\hat{x}_k - x_{min},\ x_{max} - \hat{x}_k)$, soit ici : $\theta_k = \min(\hat{x}_k,\ 4\,095 - \hat{x}_k)$.

Q12. Ecrire une fonction `prediction(x)` recevant en argument d'entrée le tableau à une dimension `x` et renvoyant le tableau `erreur` contenant les valeurs Δ_k.

Q13. A partir de la définition de la fonction de mappage, équation (2), écrire une fonction `mappage(erreur,x)` recevant en arguments d'entrée les tableaux à une dimension `erreur` et `x` et renvoyant le tableau `delta` contenant les valeurs δ_k.

II.5.2 Codage entropique - Algorithme de Rice

L'ensemble de valeurs δ_k est ensuite codé en utilisant un codage entropique spécifique : le codage de Rice. Celui-ci est particulièrement adapté à la compression de données dans lesquelles les valeurs les plus faibles sont celles qui ont la probabilité d'apparition la plus élevée, mais également lorsque la vitesse d'exécution de l'algorithme est un paramètre important.

Le principe du codage d'un entier N avec l'algorithme de Rice de paramètre p est le suivant :
- l'entier N à coder est divisé par 2^p (division entière) ;
- le quotient q de la division entière est codé avec un codage unaire (q occurences de 1 suivies d'un 0, voir **tableau 3**), ce qui constitue la première partie du code de Rice ;
- le reste r de la division entière est codé en binaire sur p bits, ce qui constitue la seconde partie du code de Rice.

Le **tableau 4** illustre le codage des entiers 0 à 7 avec un codage de Rice de paramètre $p = 2$.

Une procédure d'analyse de l'ensemble de valeurs δ_k à coder permet de déterminer la valeur optimale p_{opt} du paramètre p. Le codage de chaque valeur δ_k est ensuite réalisé par une fonction `codage(delta_k,p_opt)` recevant en argument d'entrée la valeur δ_k et l'entier p_{opt} et renvoyant le code de Rice associé.

Q14. Déterminer le code de Rice associé à la suite de valeurs δ_k données dans le **tableau 2**, page 7 pour $p = 3$.

Q15. Définir la suite d'instructions de la fonction `codage(delta_k,p_opt)` permettant d'obtenir un tableau à une ligne `code1` associé à la première partie du code de Rice (codage unaire du quotient).

Q16. Définir la suite d'instructions de la fonction `codage(delta_k,p_opt)` permettant d'obtenir un tableau à une ligne `code2` associé à la seconde partie du code de Rice (codage binaire du reste) et réalisant ensuite l'assemblage des deux parties dans le tableau `code`. Les instructions `bin` de Python ou `dec2bin` de Scilab (voir annexe, page 12) pourront être utilisées (on supposera que ces fonctions retournent une chaîne de caractères associée au code binaire).

Décimal	Code unaire
0	0
1	10
2	110
3	1110
4	11110
5	111110
6	1111110
7	11111110

Tableau 3 – Codage unaire des entiers 0 à 7

Décimal	Rice $p = 2$
0	0 00
1	0 01
2	0 10
3	0 11
4	10 00
5	10 01
6	10 10
7	10 11

Tableau 4 – Codage de Rice de paramètre $p = 2$ des entiers 0 à 7

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**.

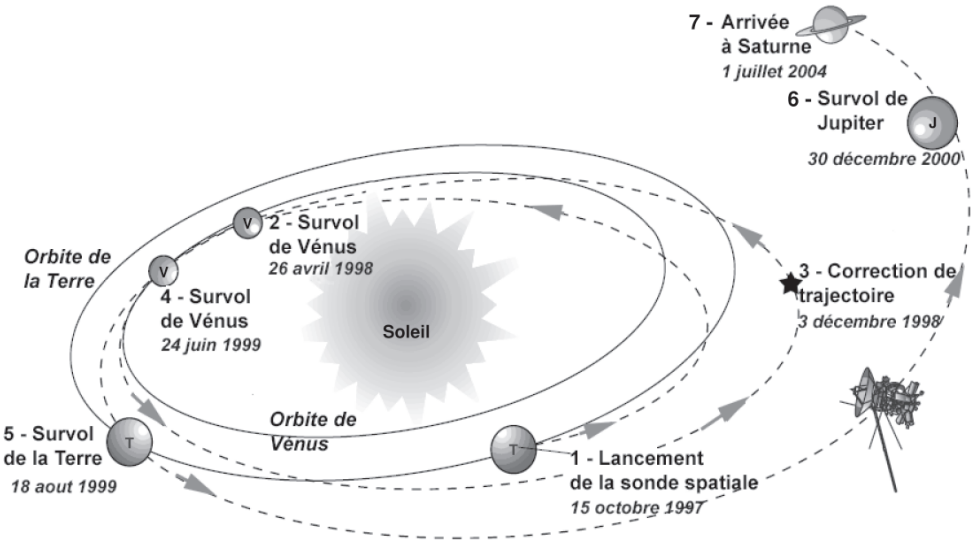

Figure 6 – Trajet de la sonde depuis la Terre vers Saturne

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 :

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

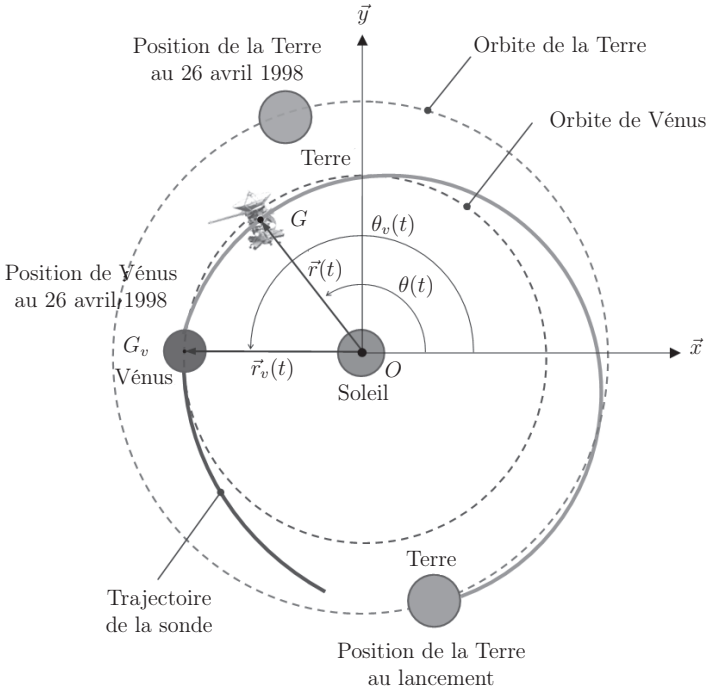

Figure 7 – Paramétrage du mouvement

avec $r_v = 1{,}082\,09 \times 10^{11}$ m, $\Omega = \dot{\theta}_v(t) = 3{,}236\,39 \times 10^{-7}$ 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 :

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

avec G constante universelle de gravitation, m_s masse du Soleil, m_v masse de Vénus ($Gm_S = 1{,}327\,24 \times 10^{20}$ N $\cdot$ m^2 $\cdot$ kg^{-1} et $Gm_V = 3{,}249\,16 \times 10^{14}$ 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 :

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

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

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 $v_x(t) = \dfrac{\mathrm{d}x(t)}{\mathrm{d}t}$ et $v_y(t) = \dfrac{\mathrm{d}y(t)}{\mathrm{d}t}$. 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 $\mathrm{km \cdot s^{-1}}$ et les axes seront légendés. On pourra utiliser les informations données en annexe pour l'utilisation des commandes de tracés.

Annexe : rappels des syntaxes en Python et Scilab

Remarque : sous Python, l'import du module numpy permet de réaliser des opérations pratiques sur les tableaux : from numpy import *. Les indices de ces tableaux commencent à 0.

Remarque : sous Scilab, les indices des tableaux commencent à 1.

	Python	Scilab
tableau à une dimension	`L=[1,2,3]` (liste) `v=array([1,2,3])` (vecteur)	`v=[1, 2, 3]` ou `[1 2 3]`
accéder à un élément	`v[0]` renvoie 1	`v(1)` renvoie 1
ajouter un élément	`L.append(5)` uniquement sur les listes	`v($+1) = 5`
tableau à deux dimensions (matrice)	`M=array(([1,2,3],[3,4,5]))`	`M=[1,2,3;3,4,5]`
accéder à un élément	`M[1,2]` ou `M[1][2]` donne 5	`M(2,3)` donne 5
extraire une portion de tableau (2 premières colonnes)	`M[:,0:2]`	`M(:,1:2)`
tableau de 0 (2 lignes, 3 colonnes)	`zeros((2,3))`	`zeros(2,3)`
dimension d'un tableau `T` de taille (i,j)	`T.shape` donne `[i,j]`	`length(T)` donne `(i,j)`
séquence équirépartie quelconque de 0 à 10.1 (exclus) par pas de 0.1	`arange(0,10.1,0.1)`	`[0:0.1:10]`
définir une chaîne de caractères	`mot="Python et Scilab"`	`mot="Python et Scilab"`
taille d'une chaîne	`len(mot)`	`length(mot)`
extraire des caractères	`mot[2:7]`	`part(mot,[11:16])`

boucle For	`for i in range(10):`
	`  print(i)`
		`for i=1:10`
		`  disp(i);`
		`end`

condition If	`if (i>3):`
	`  print(i)`
	`else:`
	`  print("hello")`
		`if (i>3) then`
		`  disp(i)`
		`else`
		`  disp("hello")`
		`end`

définir une fonction qui possède un argument et renvoie 2 résultats	`def f(param):`
	`  a=param`
	`  b=param*param`
	`  return a,b`
		`function [a,b]=f(param)`
		`  a=param`
		`  b=param*param`
		`endfunction`

conversion entier > binaire	`bin(entier)`	`dec2bin(entier)`
tracé d'une courbe de deux listes de points x et y	`plot(x,y)`	`plot(x,y)`
ajout d'un titre sur les axes d'une figure	`xlabel(texte)` `ylabel(texte)`	`xlabel(texte)` `ylabel(texte)`
ajout d'un titre principe sur une figure	`title(texte)`	`title(texte)`

FIN

IMPRIMERIE NATIONALE – 16 1226 – D'après documents fournis