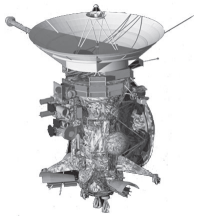
# DS 4



## La mission Cassini-Huygens

*Concours CCINP – PSI 2016*

## 1 Exercices

```
def factorielle(n) :
    """
    Donnée : un entier n >= 0
    Resultat : la factorielle de n
    """
    F=1
    i = n
    while i > 0 :
        F = F*i
        i = i-1
    return F
```

**Question 1** *Montrer que $F \times i! = n!$ est un invariant de boucle.*

```
def racine(n):
    """
    Données : un entier n >= 0
    Résultat : la racine carrée de n (arrondie à l'
        entier inférieur)
    """
    c = 0
    s = 1
    while s <= n :
        c = c + 1
        s = s + 2*c + 1
    return c
```

**Question 2** *Montrer que $n - s$ est un variant de boucle.*

## 2 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 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.

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

### 3.1 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 (Figure 1 et Figure 2).

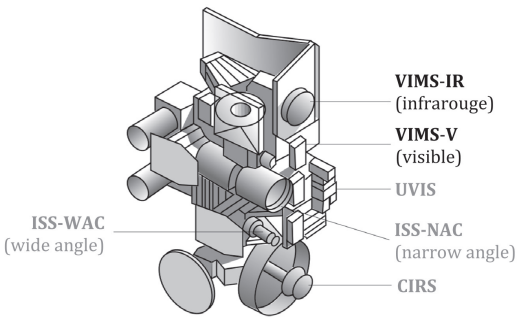



FIGURE 1 – Instruments d'observation à distance

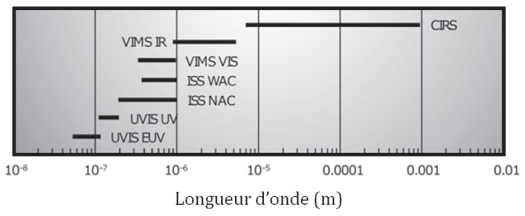



FIGURE 2 – Domaine de travail des différents instruments d'observation à distance

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 $\mu$m (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 3) constitué d'un ensemble de 352 « tranches » (associées aux différentes longueurs d'onde $\lambda$) 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 ($\lambda_k$) ou un spectre associé à un pixel de coordonnées spatiales ($x_i$, $y_j$).

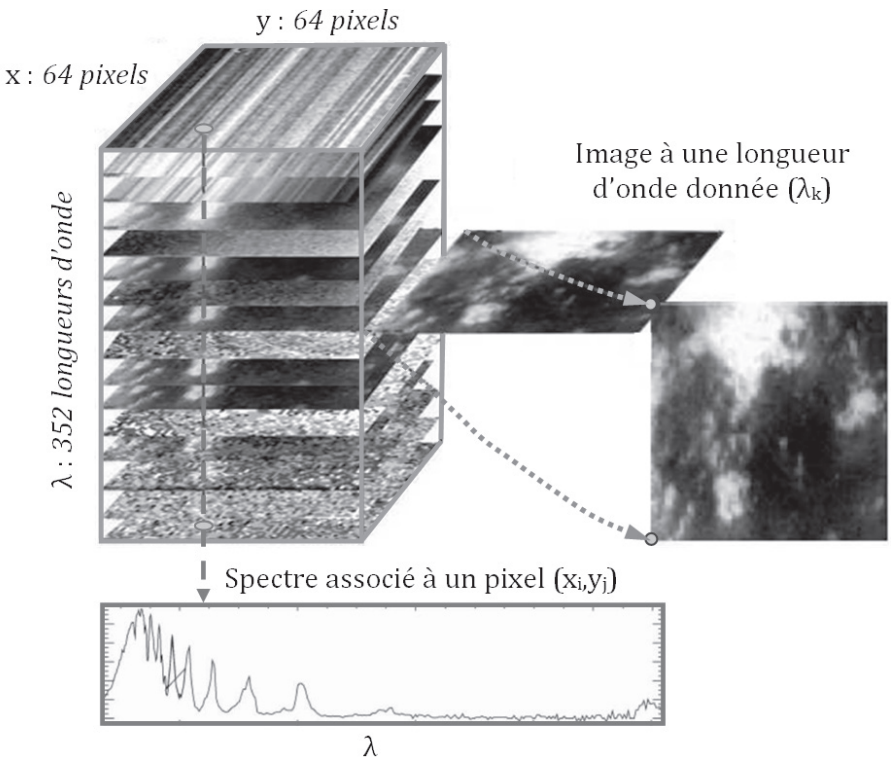



FIGURE 3 – Structure d'un cube de données hyperspectrales

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 (mégaoctet).

**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.
**Après compression, la taille maximale attendue pour un cube de données est de** 1 Mo **(mégaoctet).**

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

On rappelle qu'un octet est égal à 8 bits.

**Question 3** *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 $\tau$ est défini comme : $\tau = \dfrac{T - T_c}{T}$.

**Question 4** *Déterminer alors le taux de compression à appliquer aux données afin de respecter le cahier des charges.*

## 3.2 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 Table 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 |

TABLE 1 – Table de codage

## 3.3 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 :

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

avec :

- $N_v$ : nombre de valeurs $v_i$ différentes contenues dans la série ;
- $p_i$ : probabilité d'apparition de la valeur $v_i$ ($p_i = \dfrac{n_i}{n}$ 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 $x \in \mathbb{R}$, $\log_2(x) = \dfrac{\ln(x)}{\ln(2)}$ où ln est la fonction logarithme népérien. On suppose que la fonction $\log_2(x)$ est définie dans le langage de programmation choisi.

**Question 5** *Calculer l'entropie associée 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.

```
def entropie (S) :
    # 1. Commentaire question 4
    valeurs = list(set(S))
    # 2. Détermination du nombre d'occurrences (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
```

La fonction réalise différentes étapes. La première étape utilise la fonction `set` en Python. 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`.

```
>>> 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
```

**Question 6** *Compléter le commentaire 1 de la fonction* `entropie(S)` *correspondant à la ligne de programme définissant la variable* `valeurs`.

**Question 7** *Compléter la 2e étape de la fonction* `entropie(S)` *à partir du commentaire afin de calculer les probabilités* $p_i$.

**Question 8** *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 $4095 = 2^{12} - 1$).

On rappelle que $H$ est une valeur limite du taux de compression que l'on peut espérer atteindre.

**Question 9** *Écrire la suite d'instructions permettant de calculer et d'afficher la valeur du taux de compression limite* $\tau$ *associé aux données contenues dans le tableau* `bloc_image`.

## 3.4 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 $\lambda$). 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.

*Pour information, les étapes sont 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`.

## 3.5 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 4.

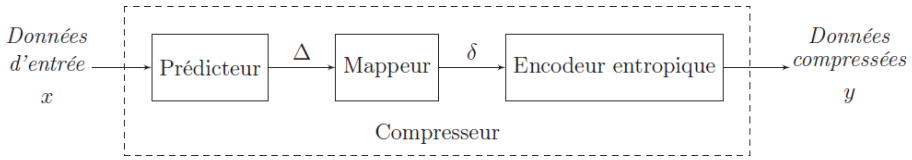



FIGURE 4 – Architecture fonctionnelle du compresseur

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 $\Delta_k$, 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 $\Delta_k$ et un entier non signé $\delta_k$ 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 $\delta_k$ en exploitant la redondance des données tel que présenté précédemment.

### 3.5.1 Prédiction du mappage

Le sous-ensemble prédicteur-mappeur a pour fonction d'associer aux échantillons d'entrée $x_k$ une série d'entiers $\delta_k$. Cette association est réalisée de telle sorte que les valeurs $\delta_k$ 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 $I = [0, 4095 = 2^{12} - 1]$.

| $k$ | $x_k$ | $\hat{x}_k$ | $\Delta_k$ | $\theta_k$ | $\delta_k$ |
|---|---|---|---|---|---|
| 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 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

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

Le tableau 2 illustre les opérations réalisées par le prédicteur et le mappeur permettant d'obtenir la série de valeur $\delta_k$ :

- le premier échantillon $x_1 = 2025$ est un échantillon de référence sur lequel se base la prédiction au rang suivant $\hat{x}_2$ ;
- à partir du deuxième échantillon, l'erreur de prédiction $\Delta_k = x_k - \hat{x}_k$ est calculée ;
- un entier non signé $\delta_k$ est associé à l'erreur de prédiction $\Delta_k$ 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 : $\theta_k = \min(\hat{x}_k, 4095 - \hat{x}_k)$.

**Question 10** *Écrire une fonction* `prediction(x)` *recevant en argument d'entrée le tableau à une dimension* `x` *et renvoyant le tableau* `erreur` *contenant les valeurs* $\Delta_k$.

**Question 11** *À 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* $\delta_k$.

### 3.5.2 Codage entropique - Algorithme de Rice

L'ensemble de valeurs $\delta_k$ 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 ($q$ occurrences de 1 suivies d'un 0, voir Table 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 Table 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 $\delta_k$ à coder permet de déterminer la valeur optimale $p_{opt}$ du paramètre $p$. Le codage de chaque valeur $\delta_k$ est ensuite réalisé par une fonction `codage(delta_k,p_opt)` recevant en argument d'entrée la valeur $\delta_k$ et l'entier $p_{\text{opt}}$ et renvoyant le code de Rice associé.

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

TABLE 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 |

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

**Question 12** *Déterminer le code de Rice associé à la suite de valeurs $\delta_k$ données dans le Table 2, page 7 pour $p = 3$. On pourra remplir le tableau suivant.*

| $\delta_k$ | Quotien par $2^p = 8$ | Codage unaire | Reste | Codage binaire | Codage Complet |
|---|---|---|---|---|---|
| 4 | | | | | |
| 10 | | | | | |
| 18 | | | | | |
| 18 | | | | | |
| 6 | | | | | |
| 1 | | | | | |
| 3 | | | | | |

**Question 13** *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).*

**Question 14** *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. L'instruction* `bin` *de Python pourra être utilisée. Elle permet de convertir nu entier en binaire (on supposera que cette fonction renvoie une chaîne de caractères associée au code binaire).*

**Fin du sujet**