

Proposition de corrigé

Concours : Banque PT

Année : 2022

Filière : PT

Épreuve : Informatique et Modélisation

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles de l'Ingénieur et d'Informatique, membres de l'[UPSTI](https://www.upsti.fr) (Union des Professeurs de Sciences et Techniques Industrielles), et publiée sur le site de l'association :

<https://www.upsti.fr/espace-etudiants/annales-de-concours>

A l'attention des étudiants

Ce document vous apportera des éléments de corrections pour le sujet traité, mais n'est ni un corrigé officiel du concours, ni un corrigé détaillé ou exhaustif de l'épreuve en question.

L'UPSTI ne répondra pas directement aux questions que peuvent soulever ces corrigés : nous vous invitons à vous rapprocher de vos enseignants si vous souhaitez des compléments d'information, et à vous adresser à eux pour nous faire remonter vos éventuelles remarques.

Licence et Copyright

Toute représentation ou reproduction (même partielle) de ce document faite sans l'accord de l'UPSTI est **interdite**. Seuls le téléchargement et la copie privée à usage personnel sont autorisés (protection au titre des [droits d'auteur](#)).

En cas de doute, n'hésitez pas à nous contacter à : corrigesconcours@upsti.fr.

Informez-vous !

Retrouvez plus d'information sur les [Sciences de l'Ingénieur](#), l'[orientation](#), les [Grandes Ecoles](#) ainsi que sur les [Olympiades de Sciences de l'Ingénieur](#) et sur les [Sciences de l'Ingénieur au Féminin](#) sur notre site : www.upsti.fr

L'équipe UPSTI

Suivi de la consommation d'eau d'une habitation

Corrigé UPSTI

Présentation de la problématique

La consommation d'eau, au même titre que la consommation d'électricité ou de gaz, représente une part non négligeable du budget de nombreux ménages, et contribue de façon significative à leur empreinte environnementale. La facturation de ces consommations étant souvent annuelle, leur suivi tout au long de l'année présente un intérêt aussi bien économique qu'écologique.

Principe du suivi

Le suivi de la consommation d'une habitation est réalisé à l'aide d'un compteur d'eau (Figure 1). Il s'agit d'un dispositif monté en série sur l'arrivée d'eau d'une habitation, qui comporte un dispositif mécanique mis en rotation par le passage de l'eau. Si le compteur a été correctement posé et étalonné, à chaque tour de la partie mobile correspond un volume d'eau connu. Un ensemble d'engrenages et de roues à cliquet permet d'afficher le volume consommé sur un cadran.

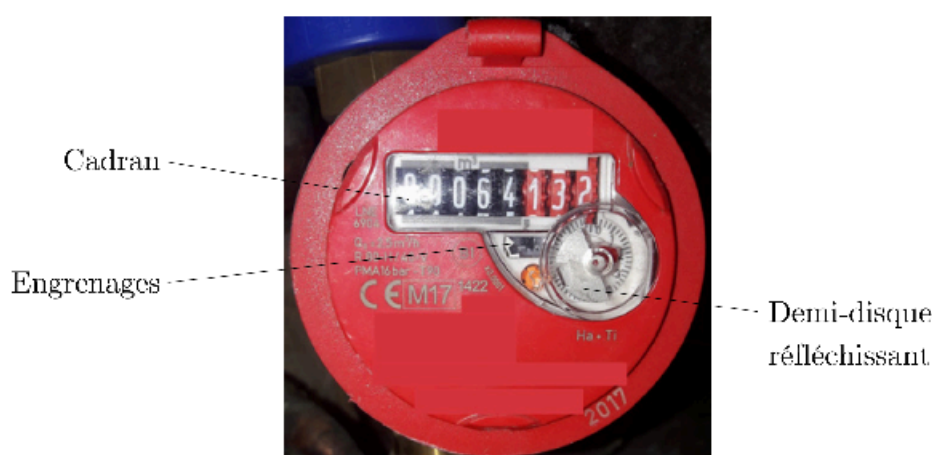


FIGURE 1 – Compteur d'eau installé dans une habitation

Le relevé fréquent d'un compteur étant une opération fastidieuse, son automatisation est d'un intérêt certain pour faciliter le suivi. On peut pour cela utiliser des compteurs dits "intelligents" ou "communicants", ou encore instrumenter un compteur d'eau classique. Dans les deux cas, le principe est le même : il s'agit de compter les tours d'un demi-disque métallique tournant avec la partie mobile au moyen d'un détecteur optique (phototransistor) ou magnétique (interrupteur à lame souple). Ces dispositifs, alimentés sous une tension continue, fournissent l'un comme l'autre une tension présentant une impulsion par tour de la partie mobile (Figure 2).

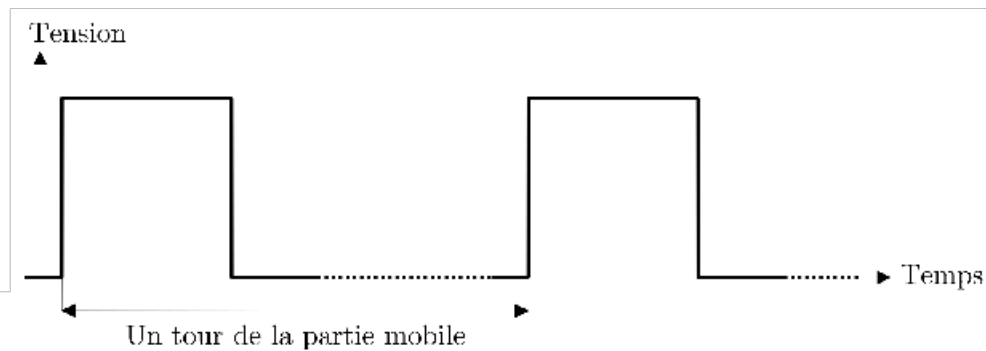


FIGURE 2 – Compteur d'eau installé dans une habitation

La mesure en temps réel de la consommation d'eau d'une habitation peut ainsi être réalisée en comptant les impulsions issues du capteur au moyen, par exemple, d'un microcontrôleur. En pratique, le signal obtenu peut être affecté par divers phénomènes indésirables (rebond des interrupteurs à lame souple, saturation progressive des phototransistors) et il est donc nécessaire de mettre préalablement en forme les impulsions pour éviter les erreurs de comptage.

Travail demandé

Ce sujet comporte deux parties.

La première partie (durée conseillée : 1h30) porte sur l'étude d'un montage électronique, utilisant un amplificateur opérationnel, destiné à mettre en forme les impulsions avant comptage.

La seconde partie (durée conseillée : 2h30) porte sur le stockage des impulsions dans le microcontrôleur, leur traitement informatique (sur un ordinateur classique) pour reconstruire les débits et volumes consommés, la transmission chiffrée des mesures vers un serveur distant mettant en œuvre une base de données, et l'exploitation de celle-ci afin d'accéder à distance au suivi de la consommation.

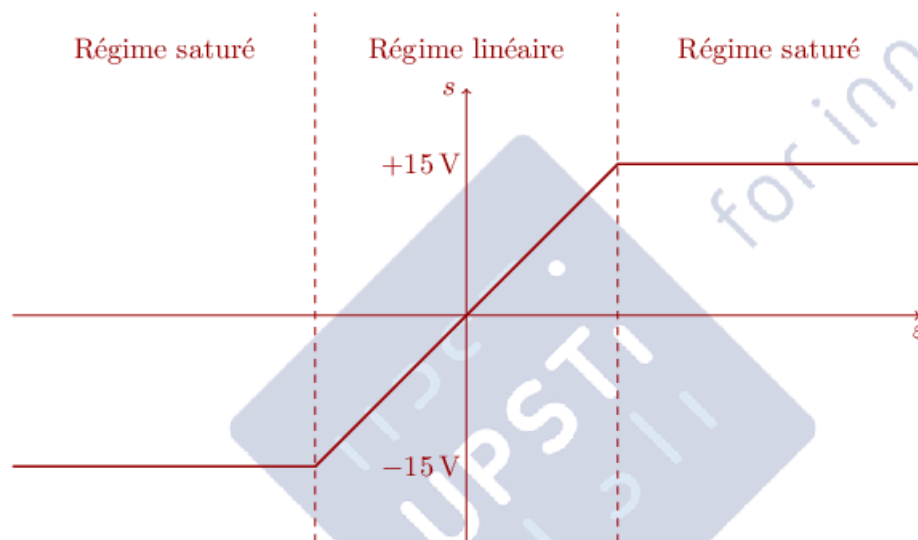
Ces deux parties sont indépendantes.

Compteur d'impulsions analogique

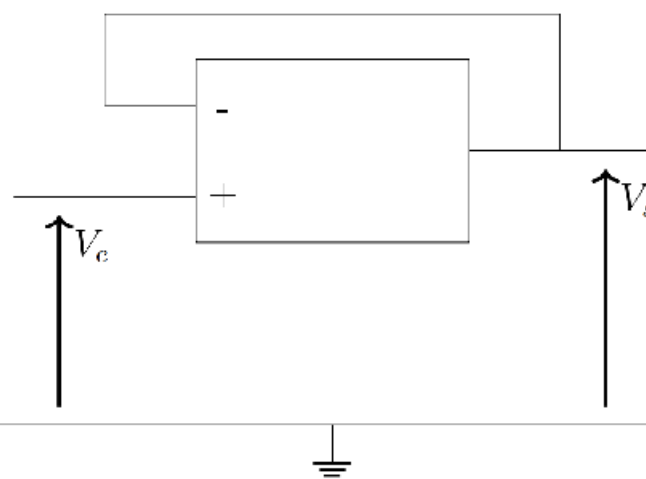
A Généralités sur les ALI (15%)

Dans cette partie, on considère un ALI alimenté en $+15/-15$ Volts par une alimentation à point milieu. On admettra que les tensions de saturation haute et basse sont $+/-15$ Volts.

Question 1 Représenter la tension de sortie en fonction de la tension différentielle d'entrée, en indiquant clairement les ordres de grandeur considérés (on indiquera la partie correspondant au régime linéaire et celle correspondant au régime saturé).



Question 2 On s'intéresse au montage représenté ci-dessous. Montrer que $V_s = V_e$. Comment s'appelle ce montage ? Quel est son intérêt ? (on considérera le gain de l'ALI comme infini)



On fait l'hypothèse que V_s et V_e ne dépassent pas les tensions de saturation haute et basse. Trois cas sont à étudier :

1. $V_s < V_e$

2. $V_s > V_e$
3. $V_s = V_e$

Dans le premier cas ($V_s < V_e$), on aurait $\varepsilon = E_+ - E_- = V_e - V_s < 0$ donc $V_s = +V_{sat} = +15\text{ V}$ (régime saturé), on aurait alors $V_e > +15\text{ V}$, ce qui est incompatible avec la condition $V_e < +15\text{ V}$.

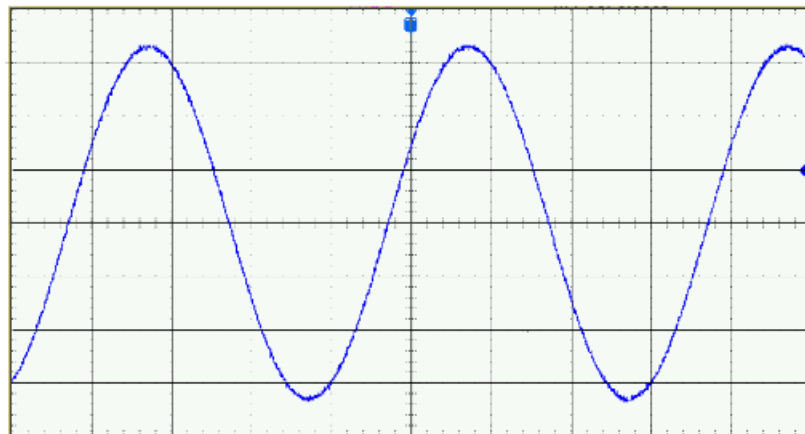
Dans le deuxième cas ($V_s > V_e$), on aurait $\varepsilon = E_+ - E_- = V_e - V_s > 0$ donc $V_s = -V_{sat} = -15\text{ V}$ (régime saturé), on aurait alors $V_e < -15\text{ V}$, ce qui est incompatible avec la condition $V_e > -15\text{ V}$.

Reste le troisième cas ($V_s = V_e$), on aurait $\varepsilon = E_+ - E_- = V_e - V_s = 0$, on serait alors en régime linéaire : $V_s = A_d \varepsilon = A_d(V_e - V_-)$, avec $A_d \rightarrow \infty$ donc $V_e - V_- \rightarrow 0$ (car V_s est une valeur finie), ce qui ne fait dépasser aucune saturation.

Seul le troisième cas étant compatible, on a donc bien $V_s = V_e$.

Il s'agit d'un montage suiveur non inverseur. L'intérêt d'un montage suiveur est d'isoler deux parties d'un circuit électrique, en "recopiant" une tension (V_e) mais sans prélever de courant en entrée. On peut ainsi réaliser des mesures de potentiel sans perturber le circuit mesuré.

Question 3 On alimente ce montage avec en entrée la tension dont l'oscillogramme est donné ci-dessous. Les réglages sont 2 V/div et $100\text{ }\mu\text{s/div}$, quelles sont les caractéristiques de cette tension ? Peut-on raisonnablement penser observer la même chose en sortie ?

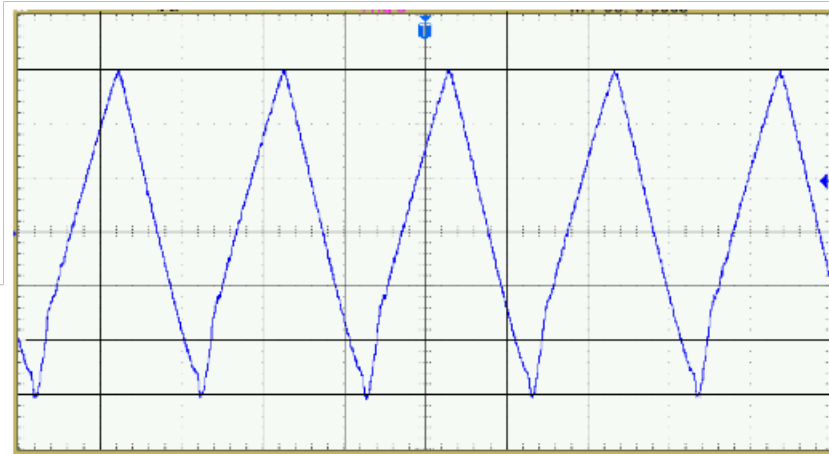


Par lecture graphique, on mesure une tension sinusoïdale :

- d'amplitude crête à crête de 13 V
- de tension moyenne 0 V
- de période $400\text{ }\mu\text{s}$, soit une fréquence de 2500 Hz

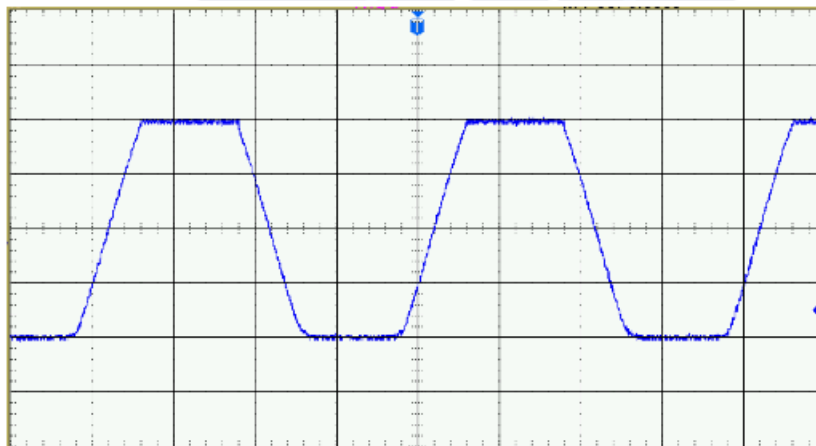
Aucune limite de l'ALI n'étant atteinte (en saturation ou en fréquence), on peut raisonnablement penser observer le même signal en sortie du montage suiveur.

Question 4 Toutes choses égales par ailleurs, on augmente la fréquence et on observe en sortie la tension ci-dessous. Les réglages sont 2 V/div et $1\text{ }\mu\text{s/div}$. Quelle caractéristique de l'ALI est ainsi mise en évidence ? Evaluer sa valeur numérique.



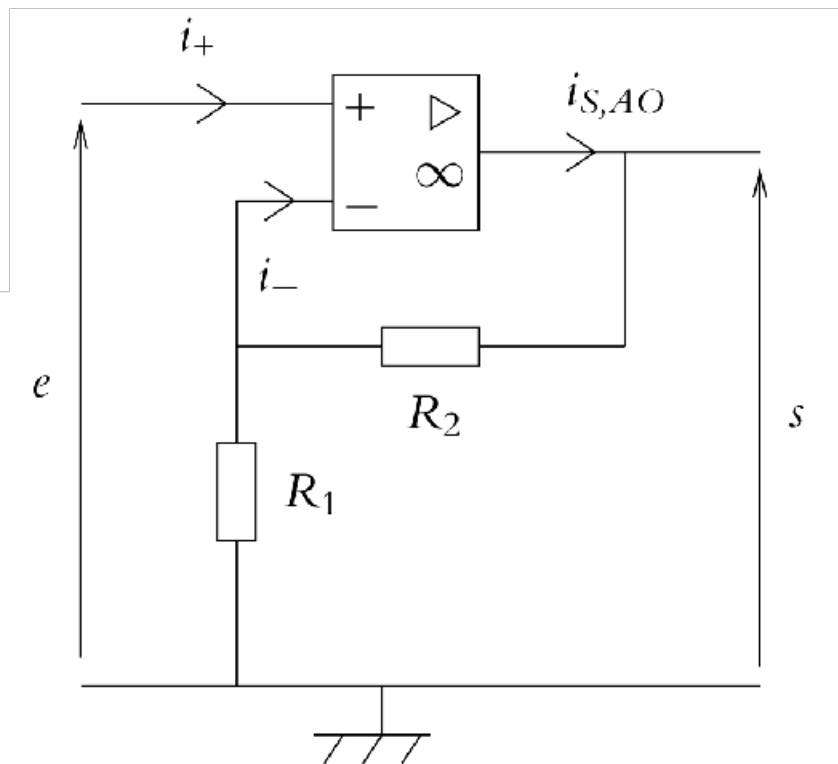
Par lecture graphique, on mesure une période de $2\mu\text{s}$ et d'amplitude crête à crête de 12V , soit une fréquence de 500kHz . On met ici en évidence la vitesse de balayage (ou *slew rate*) de l'ALI, c'est-à-dire la vitesse de suivi maximale du signal par l'ALI : il n'arrive pas à faire varier sa sortie aussi rapidement que l'entrée varie, il n'atteint donc pas les maximums du signal d'entrée.

Question 5 On revient à la fréquence de la question 3, et on ajoute une résistance de charge $R_0 = 50\Omega$ entre la sortie et la masse. Les réglages sont 2V/div et $100\mu\text{s/div}$. Quelle caractéristique de l'ALI est ainsi mise en évidence ? Evaluer sa valeur numérique.



Cette fois-ci, c'est une saturation en courant en sortie de l'ALI qui est mise en évidence : puisque la tension maximale mesurée est de 4V au lieu des $6,25\text{V}$ attendus, on peut en déduire que l'intensité maximale en sortie est de 80mA .

Question 6 Donner le schéma d'un montage amplificateur non inverseur utilisant un ALI et 2 résistances. Etablir l'expression du gain de ce montage.



On fait l'hypothèse que l'ALI est en régime linéaire, on a alors $v_- = v_+ = e$, or les entrées de l'ALI étant de résistances infinies ($i_+ = i_- = 0$), on a un pont diviseur de tension entre s , v_- et la masse : $v_- = \frac{R_1}{R_1 + R_2} s$, donc $s = \frac{R_1 + R_2}{R_1} v_- = \frac{R_1 + R_2}{R_1} e$:

$$\frac{s}{e} = \frac{R_1 + R_2}{R_1}$$

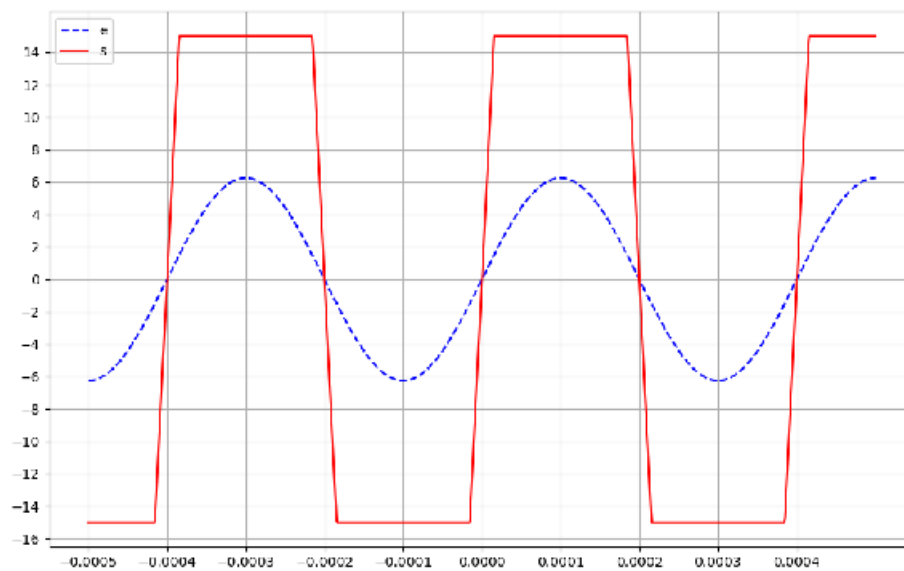
L'hypothèse (régime linéaire) sera vérifiée si $-15 \text{ V} < s < +15 \text{ V}$, donc si $-\frac{R_1}{R_1 + R_2} 15 \text{ V} < e < \frac{R_1}{R_1 + R_2} 15 \text{ V}$

Question 7 Proposer des valeurs pour les résistances pour avoir des gains de 10, 100 et 1000. Jusqu'à quelle valeur de gain peut-on aller avec un tel montage avec une tension d'entrée continue ? Avec une tension d'entrée sinusoïdale de fréquence 10kHz ? (de simples ordres de grandeur sont attendus)

Seul le rapport des résistances entre R_1 et R_2 est important, mais il peut être utile d'utiliser des valeurs importantes pour limiter le courant prélevé sur la sortie de l'ALI.

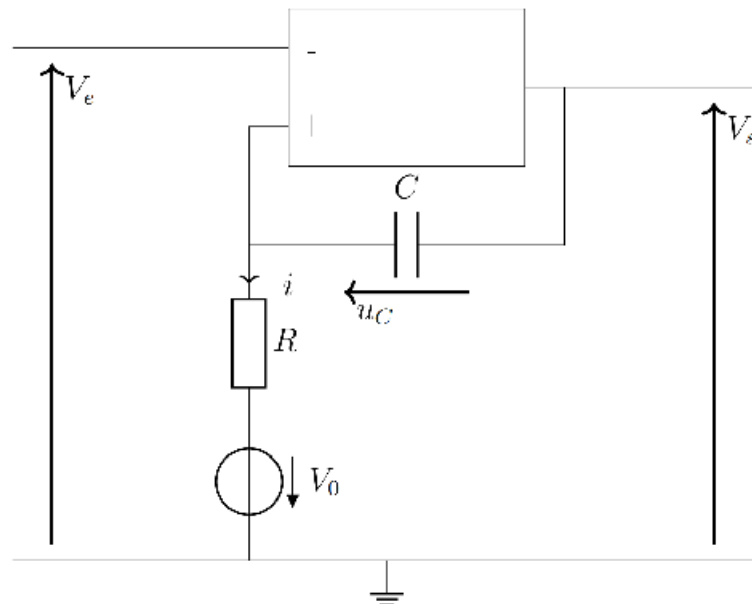
- Pour un gain de 10 : $R_1 = 10 \text{ k}\Omega$, $R_2 = 90 \text{ k}\Omega$
- Pour un gain de 100 : $R_1 = 10 \text{ k}\Omega$, $R_2 = 990 \text{ k}\Omega$
- Pour un gain de 1000 : $R_1 = 10 \text{ k}\Omega$, $R_2 = 9990 \text{ k}\Omega$

Question 8 On alimente ce montage, en prenant un gain de 10, avec la tension d'entrée de la question 3. Dessiner l'allure de la tension attendue en sortie.



B Compteur d'impulsions (25%)

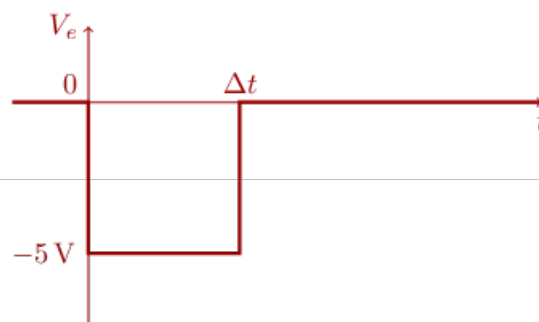
Le montage ci-dessous permet de réaliser un compteur d'impulsions analogique. L'ALI est alimenté en $+V_{cc}$ / $-V_{cc}$ avec $V_{cc} = 7\text{ V}$ par une alimentation à point milieu. Dans toute cette partie, il fonctionne en régime saturé et les tensions de saturation $+V_{sat}$ et $-V_{sat}$ sont considérées comme égales aux tensions d'alimentation $+V_{cc}$ et $-V_{cc}$. On considérera que le temps de réponse de l'ALI est négligeable (on bascule de $\pm V_{sat}$ à son opposé de manière instantanée). On prend pour ce montage $R = 10\text{ k}\Omega$, $C = 650\text{ nF}$ et $V_0 = 1\text{ V}$.



Question 9 Quelle est la valeur de i en régime stationnaire? De V_+ ? Justifier le fait que $V_s = -V_{sat}$ en régime stationnaire si $V_e = 0$. Quelle est alors la valeur de u_C ?

- En régime stationnaire, $i = 0$ (sinon u_C varierait, et on ne serait pas en régime stationnaire);
- Donc $V_+ = -V_0$ (car la tension aux bornes de R est nulle);
- Si $V_e = 0$, alors $\varepsilon = V_+ - V_- = -1\text{ V} < 0$ donc $V_s = -V_{sat}$;
- On a alors $V_s + u_C + R \times i + V_0 = 0$, donc $u_C = -V_s - V_0 = 7 - 1 = 6\text{ V}$.

Question 10 A $t = 0$, on envoie en entrée une impulsion très brève de durée Δt : V_e passe instantanément de 0 à -5 V puis, un temps Δt plus tard, repasse à 0 (toujours instantanément). Représenter cette impulsion.



Question 11 Que signifie *très brève* pour Δt ? (Avec quelle grandeur caractéristique du circuit faut-il comparer). En déduire une condition sur Δt . Expliquer pourquoi la valeur de u_C ne varie quasiment pas entre $t = 0$ et $t = \Delta t$.

Il faut comparer Δt à la constante de temps du circuit RC : $\tau = RC = 10^4 \times 650 \times 10^{-9} = 6,5 \text{ ms}$.

La tension aux bornes d'un condensateur étant continue, et le temps de l'impulsion étant largement inférieur à la constante de temps du circuit RC, la valeur de u_C ne variera quasiment pas (ça peut être vérifié en calculant $u_C(\Delta t)$ grâce à la relation donnée à la question 14).

Question 12 Expliquer pourquoi le passage de V_e de 0 à -5 V à $t = 0$ fait basculer la sortie à $+V_{sat}$, et pourquoi le retour à 0 à $t = \Delta t$ ne provoque pas un autre basculement.

A $t = 0$, on passe de $V_+ - V_- = -1 - 0 = -1 \text{ V}$ (donc $V_s = -V_{sat}$) à $V_+ - V_- = -1 - (-5) = +4 \text{ V}$, donc $V_s = +V_{sat}$ (régime saturé).

A $t = \Delta t$, $V_+ = V_s + u_C = 7 - 6 = 1 \text{ V}$ (car u_C n'a quasiment pas varié), donc $V_+ - V_- = 1 - 0 = 1 \text{ V}$ donc $V_s = +V_{sat}$ (pas de basculement).

Question 13 Montrer que, suite au basculement (on prend $t = 0$ au moment du basculement), u_C évolue de la manière suivante :

$$u_C = 2V_{sat}e^{-t/\tau} - (V_{sat} + V_0)$$

Donner l'expression de τ ainsi que son sens physique.

La loi des mailles donne :

$$V_s + u_C + R \cdot i + V_0 = 0$$

$$V_{sat} + u_C + R \cdot C \cdot \frac{du_C}{dt} + V_0 = 0$$

$$u_C + R \cdot C \cdot \frac{du_C}{dt} = V_{sat} - V_0$$

Il s'agit d'une équation différentielle linéaire du premier ordre, qui a pour solution (avec la condition initiale $u_C(0) = V_{sat} - V_0$) :

$$u_C = 2V_{sat}e^{\frac{-t}{RC}} - (V_{sat} + V_0)$$

Avec $\tau = RC$ la constante de temps, durée caractéristique du délai de charge ou décharge du condensateur.

Question 14 A quel instant t_1 la sortie va-t-elle repasser en saturation basse? Donner l'expression en fonction de R , C , V_{sat} et V_0 et faire l'application numérique. On donne $\ln(14) \approx 2,6$.

La sortie va repasser en saturation basse lorsque $V_+ < V_-$, donc lorsque $V_s + u_C < V_-$, donc lorsque $u_C < V_- - V_s = 0 - 7 \text{ V}$.

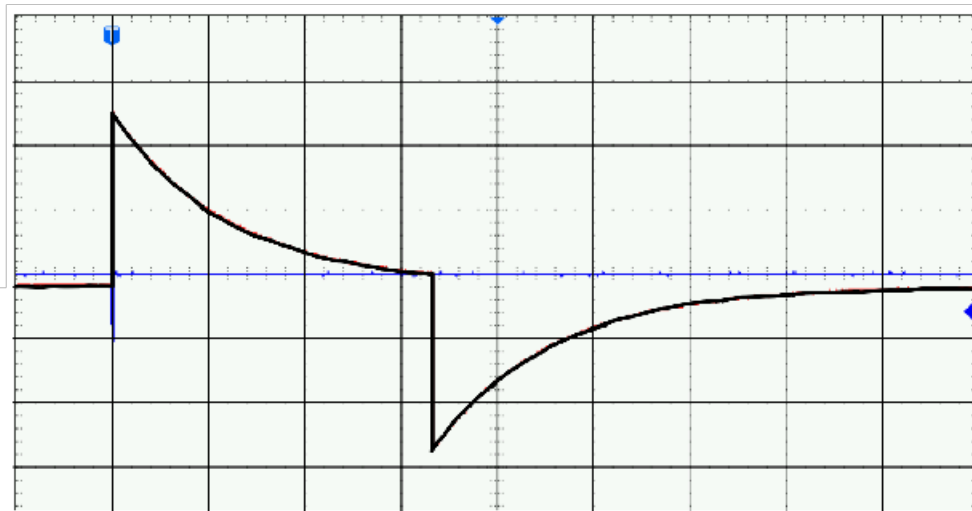
On cherche t_1 tel que $u_C(t_1) = -7 \text{ V}$.

$$u_C(t_1) = 2V_{sat}e^{\frac{-t_1}{RC}} - (V_{sat} + V_0) = -7 \text{ V}$$

$$t_1 = -RC \times \ln\left(\frac{-7 + (V_{sat} + V_0)}{2 \cdot V_{sat}}\right)$$

$$t_1 = -6,5 \text{ ms} \times \ln\left(\frac{-7 + (7 + 1)}{2 \cdot 7}\right) = 6,5 \text{ ms} \times \ln(14) \approx 17 \text{ ms}$$

On donne ci-dessous un enregistrement de la tension V_+ suite à une impulsion. L'instant $t = 0$ est décalé à une division après la gauche de l'écran. Les réglages sont 5 V/div et 5 ms/div .

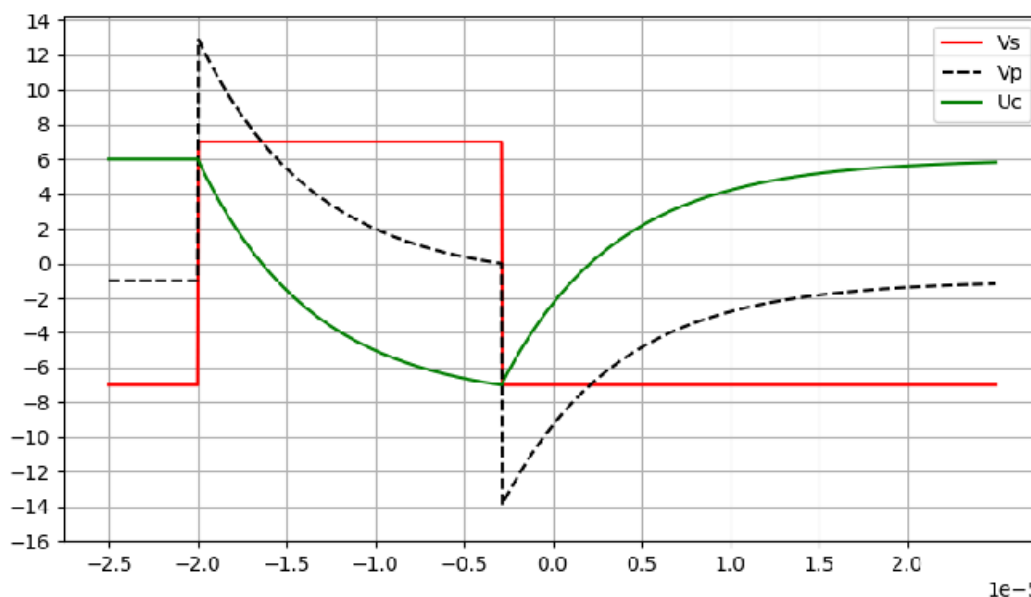


Question 15 Expliquer les deux phases observées dans l'évolution de cette tension (expliquer en particulier la valeur minimale prise par V_+).

La première phase débute par une impulsion qui bascule V_s en saturation haute : on observe alors sur V_+ le déchargement du condensateur. La deuxième phase commence lorsque que $V_+ < V_-$ (17 ms après le début de la première phase) : la sortie bascule alors en saturation basse, et le condensateur se charge.

Au moment du basculement entre les deux phases, V_+ prend sa valeur minimale : u_C est à sa charge minimale (la tension a ses bornes est de -7 V), et V_s passe de 7 V à -7 V , or $V_+ = V_s + u_C$, ce qui explique cette valeur minimale.

Question 16 Sur la même échelle de temps que l'enregistrement précédent, représenter l'évolution des tensions u_C et V_s .

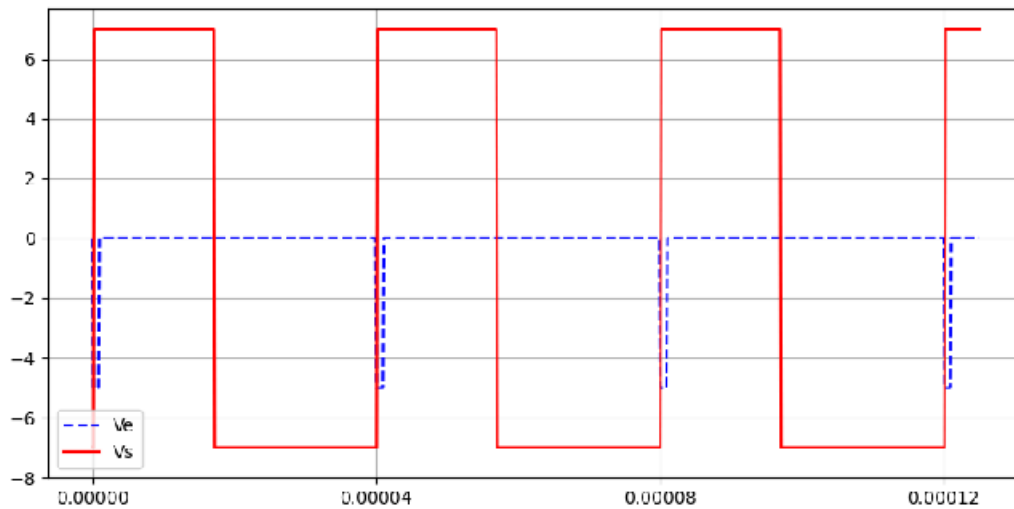


Le montage reçoit en entrée des impulsions périodiques, toujours de largeur Δt et avec une période T , dans le but de pouvoir mesurer la fréquence f de ces impulsions.

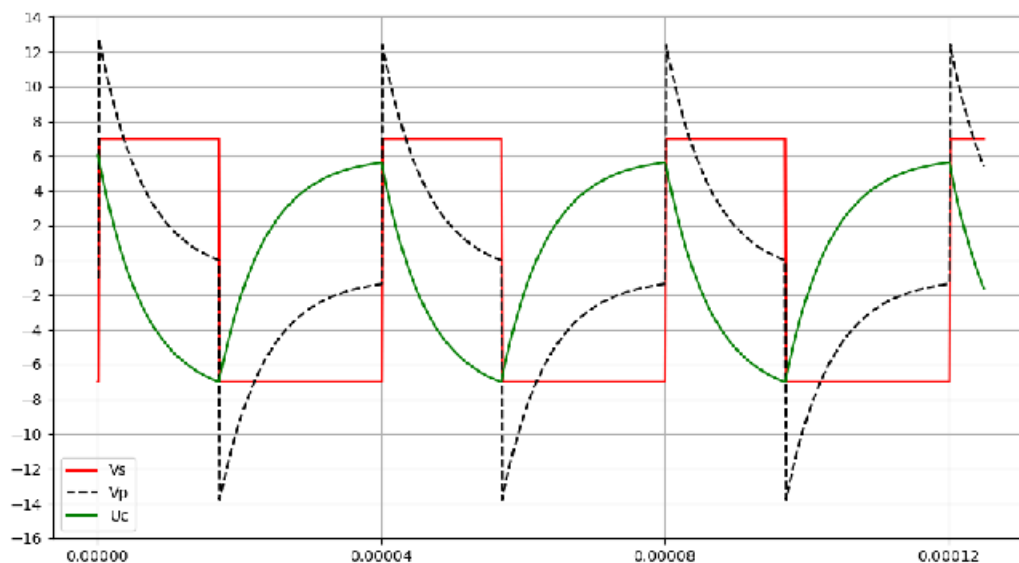
Question 17 Quelle condition doit respecter T vis-à-vis des caractéristiques du montage ?

Pour que les impulsions puissent être comptées, il faut que $T > 2 \times t_1$, pour que le condensateur ait le temps de suivre un cycle de décharge-charge.

Question 18 Représenter sur un même graphe, sans échelle mais en respectant les conditions des questions 11 et 17, les tensions V_e et V_s (sur au moins une période T et au plus deux). Indiquer clairement les trois temps caractéristiques.



Avec u_C et V_+ :

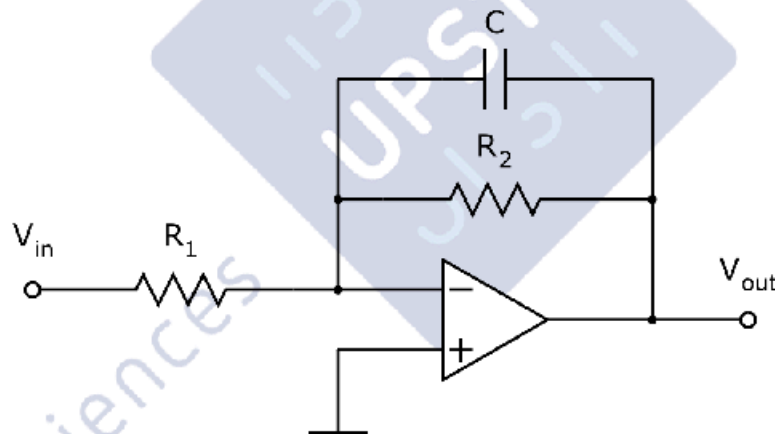


Question 19 Donner l'expression de la valeur moyenne $\langle V_s \rangle$ de V_s en fonction de V_{sat} , f et t_1 .

$$\begin{aligned}
 \langle V_s \rangle &= \frac{1}{T} \int_0^T V_s \times dt \\
 \langle V_s \rangle &= f \left(\int_0^{t_1} V_{sat} \times dt + \int_{t_1}^T -V_{sat} \times dt \right) \\
 \langle V_s \rangle &= f \left(\int_0^{t_1} (V_{sat} + V_{sat} - V_{sat}) \times dt + \int_{t_1}^T -V_{sat} \times dt \right) \\
 \langle V_s \rangle &= f \left(\int_0^{t_1} 2 \times V_{sat} \times dt + \int_0^T -V_{sat} \times dt \right) \\
 \langle V_s \rangle &= f \left(\int_0^{t_1} 2 \times V_{sat} \times dt \right) - V_{sat} \\
 \langle V_s \rangle &= (2 \times f \times t_1 - 1) V_{sat}
 \end{aligned}$$

Question 20 Quel type de filtre pourrait-on utiliser en aval du montage précédent pour obtenir $\langle V_s \rangle$? Préciser comment il doit être branché (faire un schéma) et proposer des valeurs pour ses composants si $f = 10 \text{ Hz}$.

Pour obtenir la valeur moyenne de V_s , un filtre passe-bas (aussi appelé intégrateur) pourrait être utilisé.

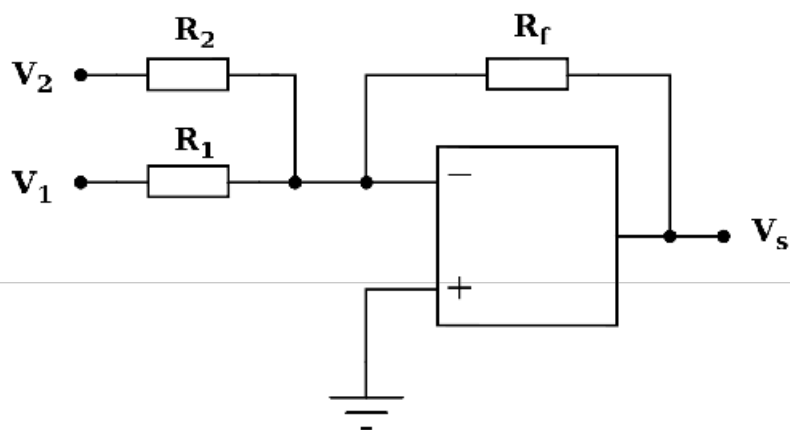


Pour un tel montage, la fréquence de coupure est $f_c = \frac{1}{2\pi R_2 C}$: on peut prendre $C = 1000 \text{ nF}$ et $R_2 = 2000 \Omega$ pour obtenir une fréquence de coupure de $80 \text{ Hz} > f = 10 \text{ Hz}$.

On choisira R_1 de manière à ne pas saturer la sortie du premier ALI (compteur d'impulsion), et en le prenant en compte pour le gain proportionnel de ce montage intégrateur.

Question 21 On souhaite obtenir avec un voltmètre une tension directement proportionnelle à f . Expliquer comment compléter le montage en aval du filtre pour obtenir ce résultat.

En aval du filtre intégrateur, on obtient une tension proportionnelle à f , mais avec un décalage (offset) de V_{sat} . Un montage additionneur permettrait d'obtenir une tension strictement proportionnelle à f .



En reliant V_1 à la sortie du filtre intégrateur, V_2 à $+V_{sat}$, et en choisissant $R_1 = R_2$, on pourra alors obtenir avec un voltmètre une tension directement proportionnelle à f .

Exploitation informatique des mesures

Dans cette partie, les fonctions et programmes attendus devront être écrits en langage Python (sauf, bien entendu, lorsqu'une requête SQL est demandée). On prendra garde à la lisibilité du code, et notamment aux indentations. Le code devra être documenté. Toute fonction définie dans le sujet pourra être utilisée, même sans avoir été codée.

C Estimation des débits et volumes (30%)

La sortie du dispositif précédent est raccordée à une entrée logique d'un microcontrôleur. À chaque fois qu'une impulsion se produit, le microcontrôleur mémorise l'instant auquel l'impulsion a été détectée. Cette information est ensuite utilisée pour reconstituer l'évolution temporelle du volume d'eau et du débit consommés. On se propose ici d'implanter partiellement cette reconstitution.

C.1 Justification du format des données

Si l'on fait l'hypothèse que le débit volumique Q traversant le compteur est constant dans l'intervalle de temps séparant deux impulsions, alors on l'obtient par la relation :

$$Q = \frac{c}{\tau} \quad (1)$$

où $c = 7 \cdot 10^{-5} \text{ m}^3$ est le volume d'eau consommé à chaque impulsion, et τ la durée séparant deux impulsions. Le débit maximal pouvant traverser le compteur ne dépasse pas $Q_{\max} = 2,5 \text{ m}^3/\text{h}$.

Question 22 Estimer la plus petite valeur numérique que prend τ .

$$\tau_{\min} = \frac{c}{Q_{\max}} = \frac{7 \times 10^{-5} \times 3600}{2,5} \approx 0,1 \text{ s}$$

Le microcontrôleur effectue le chronométrage et le stockage en mémoire des instants avec une résolution limitée. On suppose donc que la durée τ est connue avec une incertitude $\delta\tau$. Par suite, le débit Q est déterminé avec une incertitude δQ . On souhaite que l'incertitude relative $\delta Q/Q$ soit d'au plus 1% en valeur absolue (toujours sous l'hypothèse simplificatrice d'un débit constant).

Question 23 Exprimer, en valeur absolue, $\delta Q/Q$ en fonction de $\delta\tau/\tau$. Estimer alors la valeur maximale que doit avoir $\delta\tau$ pour que la condition ci-dessus soit satisfaite au débit $Q_{\max}$.

Pour obtenir l'incertitude relative, on passe au logarithme népérien :

$$\ln Q = \ln \frac{c}{\tau}$$

Puis on prend la différentielle :

$$\delta Q/Q = \delta c/c - \delta\tau/\tau$$

c étant une constante, et en passant à la valeur absolue :

$$|\delta Q/Q| = |\delta\tau/\tau|$$

On souhaite $|\delta Q/Q| < 1\%$, donc $|\delta\tau/\tau| < 1\%$, donc $\delta\tau < \frac{\tau}{100}$. Pour que ce soit tout le temps vérifié, on doit avoir :

$$\delta\tau < \frac{\tau_{\min}}{100} \quad \Rightarrow \quad \delta\tau < 1 \text{ ms}$$

Les instants des impulsions sont représentés au format *timestamp*, défini comme le nombre de secondes écoulées depuis le 1^{er} janvier 1970 à minuit. Par exemple, lors d'une acquisition effectuée début 2021, des impulsions ont été détectées aux timestamp suivants, exprimés en secondes :

1609825201,145 1609825202,948 1609825206,532 ...

Le microcontrôleur utilisé pour l'acquisition stocke les nombres sur au plus 32 bits. Il propose des types entier et flottant définis comme suit :

- les entiers peuvent prendre des valeurs comprises entre -2^{31} et $2^{31} - 1$;
- les flottants sont au format simple précision de la norme IEEE 754, qui prévoit une mantisse codée sur 23 bits et un exposant codé sur 8 bits.

Quelle que soit la valeur obtenue précédemment, on admet que les instants des impulsions doivent être mémorisés à 1 ms près. Pour les applications numériques, on pourra considérer que $2^{10} \approx 1000$.

Question 24 Justifier qu'il n'est pas possible de stocker directement les instants des impulsions dans le microcontrôleur avec la résolution demandée, que ce soit avec le stockage entier (en exprimant les instants en millisecondes) ou avec le stockage flottant.

Le nombre de millisecondes depuis le 1^{er} janvier 1970 est d'environ $1,6 \times 10^{12}$. Or, $2^{31} - 1 \approx 1,6 \times 2 \times (2^{10})^3 \approx 3 \times (10^3)^3 \approx 3 \times 10^9 \ll 1,6 \times 10^{12}$. Il n'est donc pas possible de stocker ces grandeurs sous forme d'entiers.

De même, la précision des flottants avec une mantisse codée sur 23 bits est insuffisante (puisque $23 < 32$, et que 32 bits sont insuffisants).

Il n'est donc pas possible de stocker directement les instants avec la résolution demandée, que ce soit avec le stockage entier ou avec le stockage flottant.

C.2 Récupération des données

Compte tenu de la limitation mise en évidence dans la partie précédente, on décide de représenter les instants des impulsions de la façon suivante :

- on synchronise le microcontrôleur à l'aide d'une horloge extérieure : on repère un instant où le timestamp est un nombre entier de secondes, et on relève la valeur entière correspondante ;
- on relève ensuite, à chaque impulsion, le nombre entier de millisecondes écoulées depuis la synchronisation.

Ces valeurs sont stockées en mémoire et, à intervalles de temps réguliers, transmises à un ordinateur où elles sont écrites dans un fichier texte. Par exemple, les timestamp donnés dans la partie précédente :

1609825201,145 1609825202,948 1609825206,532 ...

pourraient être représentés par le fichier texte suivant :

```
1609825200
1145
2948
6532
...
```

On souhaite, à partir de ce fichier, construire en Python une liste de flottants contenant les timestamp. Supposons par exemple que le fichier précédent s'appelle log.txt et se limite aux quatre premières lignes ci-dessus. On exécute alors les instructions suivantes, dont on donne le résultat affiché dans l'interpréteur :


```

1 >>> f = open("log.txt", "r")
2 >>> lignes = f.readlines()
3 >>> f.close()
4 >>> print(lignes)
5 ['1620115200\n', '1145\n', '2948\n', '6532\n']

```

On précise que la séquence de caractères `\n` indique le passage à la ligne et ne perturbe pas les fonctions de conversion. On précise également que le type flottant de Python est au format double précision sur la plupart des ordinateurs, ce qui suffit largement à stocker les instants des impulsions avec la résolution exigée.

Question 25 À l'aide de l'exemple ci-dessus, écrire une fonction `recup(nomFichier)` qui prend pour argument une chaîne de caractères `nomFichier` donnant le nom du fichier texte contenant les instants stockés au format décrit dans cette partie, et renvoie une liste de flottants contenant les instants des impulsions exprimés en seconde.

```

1 def recup(nomFichier):
2     # Lecture du fichier
3     f = open(nomFichier, "r")
4     lignes = f.readlines()
5     f.close()
6
7     res = [] # Conteneur des dates des impulsions
8
9     temps_initial = float(lignes[0]) # date de la synchronisation
10
11     for i in range(1, len(lignes)): # On omet la première ligne,
12                                     # qui correspond à la synchronisation
13                                     # et non à une impulsion
14         res.append(temps_initial + float(lignes[i])/1000)
15
16     return res

```

C.3 Première reconstruction : débit constant par morceaux

On suppose dans cette partie que l'on dispose de la liste des instants des impulsions, notés T_i pour $0 \leq i \leq n$ (il y en a donc $n + 1$). Par construction, cette liste est croissante au sens strict, deux impulsions ne pouvant pas survenir simultanément.

On envisage tout d'abord l'hypothèse d'un débit constant par morceaux. On pose alors, pour tout i tel que $0 \leq i \leq n - 1$:

$$q_i = \frac{c}{T_{i+1} - T_i} \quad (2)$$

où c est le volume consommé entre deux impulsions.

Question 26 Écrire une fonction `debts1(lImp,c)` qui prend pour arguments la liste `lImp` des instants des impulsions et un flottant c , et renvoie la liste des débits définis par l'équation (2).

```

1 def debts1(lImp,c):
2     res = [] # Conteneur des débits
3
4     for i in range(len(lImp) - 1): # On aura un débit de moins que d'impulsions
5         res.append(c / (lImp[i+1] - lImp[i]))
6
7         # Les débits sont en (unité de c) par seconde
8
9     return res

```

À partir de ces deux listes, on souhaite reconstruire l'évolution temporelle du débit volumique consommé sur l'intervalle de temps correspondant à l'enregistrement, de la façon suivante :

$$q(t) = \begin{cases} 0 & \text{si } t < T_0 \text{ ou } t \geq T_{n-1} \\ q_i & \text{si } T_i \leq t \leq T_{i+1}, 0 \leq i < n-1 \end{cases} \quad (3)$$

Question 27 Écrire une fonction `indice(lImp,t)` qui prend pour arguments la liste `lImp` des instants T_i des impulsions et un flottant `t`, tels que `t` est compris entre le premier et le dernier élément de `lImp` (premier inclus, dernier exclu), et recherche l'entier i tel que $T_i \leq t \leq T_{i+1}$. On ne demande pas de vérifier la condition précédente (ce sera fait lors de l'appel de la fonction).

```
1 def indice(lImp,t):
2     trouve = False
3     i = 0
4
5     # On parcourt la liste des temps par indice
6     # jusqu'à ce que l'on trouve un temps supérieur à t.
7     # La boucle se termine avant que i = len(lImp) d'après la condition du sujet
8     while i < len(lImp) and not trouve:
9         if lImp[i] >= t:
10             trouve = True
11             indice = i - 1
12             i += 1
13     return indice
```

L'évaluation précédente est menée en m instants de calcul utilisés pour le tracé et l'interpolation, stockés dans une liste `lTemps`.

Question 28 Écrire une fonction `temporel(lQ,lImp,lTemps)` qui prend pour arguments trois listes `lQ` (débits moyens calculés par l'équation (2)), `lImp` (instants des impulsions) et `lTemps` (instants de calcul), et renvoie la liste des débits moyens aux instants de `lTemps`. Ceux-ci ne sont pas forcément compris entre le premier et le dernier élément de `lImp`. Lorsqu'ils le sont, on appellera la fonction `indice`.

```
1 def temporel(lQ,lImp,lTemps):
2     res = [] # Conteneur des débits moyens aux instants de lTemps
3
4     for temps in lTemps:
5         # Le temps est-il dans la plage de mesure ?
6         if temps < 0 or temps > lTemps[-1]: # Si non
7             res.append(0)
8         else: # Si oui
9             # Récupération de l'indice correspondant à ce temps
10            i = indice(lImp, temps)
11            # Récupération du débit moyen
12            debit = lQ[i]
13            res.append(debit)
14
15     # Les débits moyens sont dans la même unité que les débits de lQ
16     return res
```

Question 29 Donner et justifier la complexité asymptotique de la fonction `temporel` en fonction du nombre m d'instants de calcul et du nombre n d'impulsions dans le pire des cas.

Dans le pire des cas :

- La fonction `temporel()` fait m itérations de sa boucle, soit m appels à `indice()` ;
- La fonction `indice()` fait n itérations de sa boucle (dans le cas où le temps est à la fin de la liste des impulsions).

Soit une complexité dans le pire des cas en $\mathcal{O}(n \times m)$ opérations élémentaires.

En pratique, la liste des impulsions et celle des instants de calcul sont toutes deux triées par ordre croissant. En optimisant le parcours des deux listes, il est possible de réduire la complexité des calculs.

Question 30 Justifier qu'il n'est pas possible de réduire la complexité tout en conservant l'appel à la fonction `indice`. Expliquer, en deux ou trois phrases éventuellement complétées par un schéma, comment passer en revue les éléments des deux listes pour minimiser le nombre d'opérations si l'on suppose que les deux tailles m et n sont voisines. Donner la complexité obtenue pour $m = n$.

La fonction `temporel()` doit forcément itérer sur chaque instant de calcul, sa complexité en fonction du nombre d'instant de calcul ne peut donc pas être meilleure que linéaire. La seule possibilité pour réduire la complexité est donc de ne plus faire appel à la fonction `indice` telle qu'implémentée actuellement.

Pour passer en revue les éléments des deux listes pour minimiser le nombre d'opérations :

- Comme les deux listes sont triées, on peut utiliser une implémentation dichotomique de la recherche de la fonction `indice` : la complexité de `temporel()` serait alors quasi-linéaire ($\mathcal{O}(m \times \log(n))$ opérations élémentaires) ;
- Comme les deux listes sont triées et de tailles voisines, on peut supposer que l'indice correspondant au temps recherché dans la liste `lImp` est proche de son indice dans la liste `lTemps` : on peut donc commencer la recherche d'indice autour de cet indice. La complexité de la fonction `indice()` serait alors constante (seules quelques itérations seraient nécessaires, une quantité indépendante de la taille de la liste `lImp`). La complexité de la fonction `temporel()` serait alors linéaire ($\mathcal{O}(m)$ opérations élémentaires).

En plus du débit volumique, on souhaite pouvoir visualiser le volume consommé, défini comme l'intégrale temporelle du débit volumique. Pour tout j tel que $0 \leq j < m$, on note t_j les instants de calcul, $q_j = q(t_j)$ les débits et $v_j = v(t_j)$ les volumes consommés. Les instants de calcul étant supposés régulièrement espacés, on notera $\Delta t = t_{j+1} - t_j$.

Question 31 Donner la relation de récurrence permettant d'exprimer, par la méthode des trapèzes, v_j en fonction de v_{j-1} , q_j , q_{j-1} et Δt .

$$v_j = v_{j-1} + \frac{q_j + q_{j-1}}{2} \times \Delta t$$

Question 32 Écrire une fonction `volumes(lTemps, lQT, v0)` qui prend pour arguments deux liste `lTemps` (instants de calcul) et `lQT` (débits aux instants de `lTemps`) et un flottant `v0`, et renvoie la liste des volumes consommés, calculés par la méthode des trapèzes. Le volume initial (avant le premier instant de `lTemps`) sera pris égal au paramètre `v0`.

```
1 def volumes(lTemps, lQT, v0):
2     res = [v0] # Conteneur des volumes
3
4     for i in range(1, len(lTemps)):
5         deltat = lTemps[i] - lTemps[i-1]
6         qmoy = (lQT[i] + lQT[i-1])/2
7
8         vi = res[-1] + qmoy * deltat # Intégration par la méthode des trapèzes
9         res.append(vi)
10
11     # Les volumes sont en (unité des débits) fois (unité des temps)
12     return res
```

On dispose d'un relevé d'impulsions contenu dans un fichier nommé `2021-01-31.txt`. On souhaite faire appel aux fonctions précédentes pour extraire les impulsions stockées dans ce fichier, déterminer les débits et préparer

les deux listes "temporelles" nécessaires au tracé.

Question 33 Écrire une suite d'instructions appelant entre autres les fonctions précédentes de sorte à :

- placer le contenu du fichier 2021-01-31.txt dans une liste lImp ;
- construire la liste des débits lQ associés, en litres par seconde, si à chaque impulsion correspond 0,07 litre ;
- générer une liste lTemps de 1000 flottants régulièrement espacés, allant de la première à la dernière valeur de lImp, tous deux inclus ;
- construire la liste des débits lQT (tirés de lQ) aux instants de lTemps ;
- en déduire la liste lVT des volumes exprimés en litres, toujours aux instants de lTemps, pour un volume initial nul.

```
1 import numpy as np
2
3 lImp = recup("2021-01-31.txt") # Les impulsions sont en secondes
4 lQ = debitsl(lImp, 0.07) # L'unité de volume utilisée est le litre, celle des débits sera donc en litre
   par seconde
5
6 # Version numpy
7 lTemps = np.linspace(lImp[0], lImp[-1], 1000)
8
9 # Version sans numpy
10 lTemps = []
11 for i in range(1000):
12     lTemps.append(lImp[0] + (lImp[-1] - lImp[0]) * i / 999)
13
14 lQT = temporel(lQ, lImp, lTemps) # Débits aux instants de lTemps
15
16 lVT = volumes(lTemps, lQT, 0.0) # Volumes en litres aux instants de lTemps
```

À partir des listes venant d'être construites, on peut tracer l'évolution temporelle du débit et du volume, donnée Figure 3.

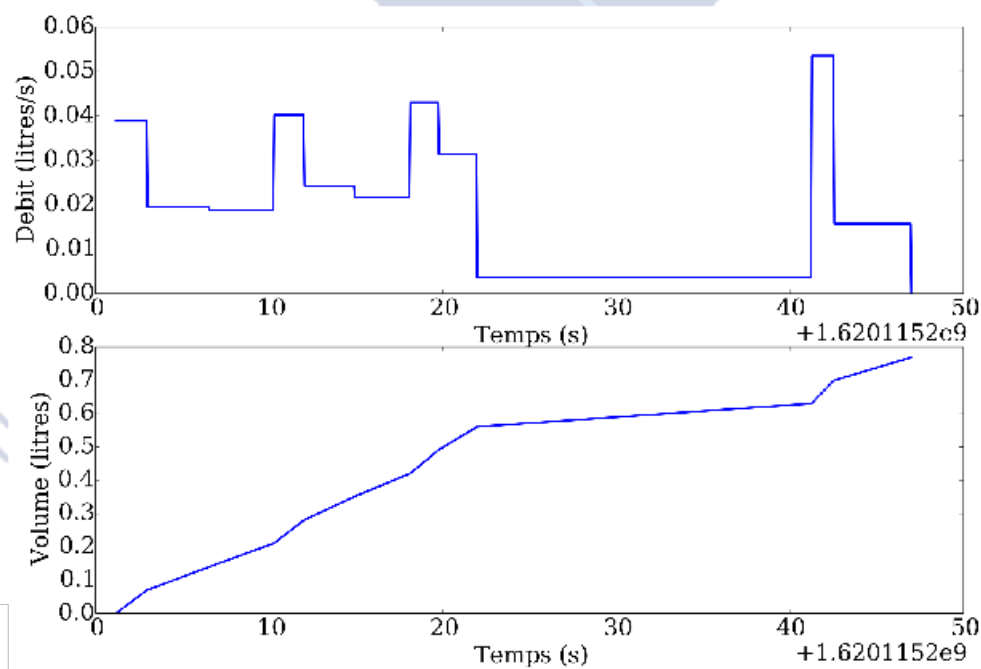


FIGURE 3 – Évolution temporelle du débit et du volume consommé

Une limitation de cet algorithme est qu'il ne permet pas, en l'état, de détecter l'absence de consommation d'eau pendant un laps de temps donné. En effet, un tel cas de figure se manifeste par la présence de deux impulsions

très espacées, et l'algorithme estime alors, de façon logique, un débit volumique constant et très faible entre ces deux impulsions. On se propose donc de modifier le mode d'évaluation des débits pour permettre la détection des périodes sans consommation, selon le principe schématisé Figure 4.

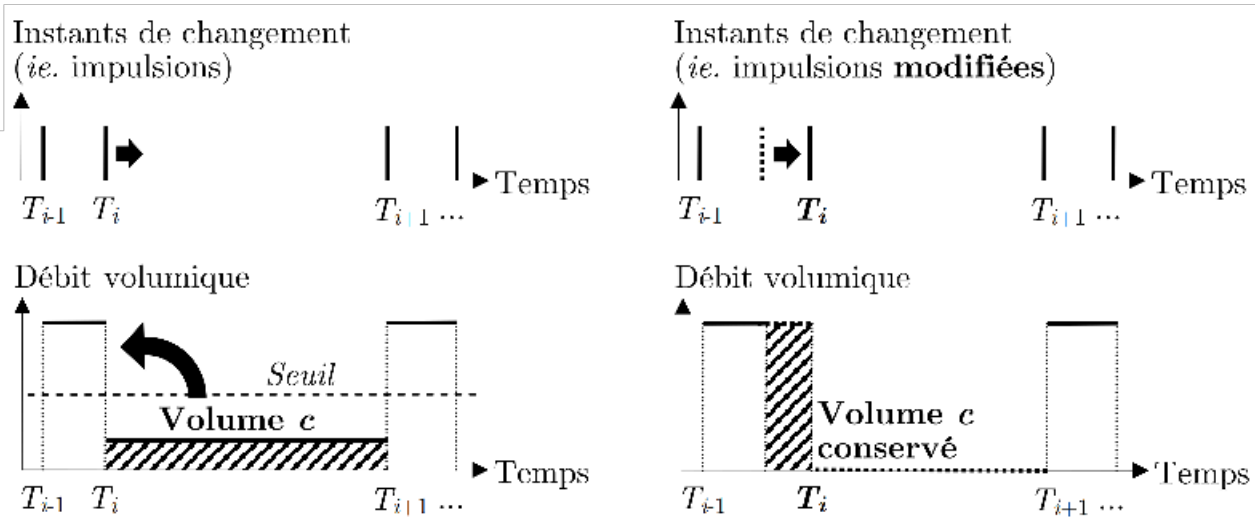


FIGURE 4 – Principe de la détection des périodes sans consommation

Pour cela, on se donne un seuil, ou débit minimal admissible; le débit q_i calculé par l'équation (2) ne sera retenu que s'il est supérieur à ce seuil. Si le débit q_i est inférieur au seuil, on le met à zéro, et on retarde l'instant T_i de l'impulsion précédente de sorte à conserver le volume total consommé. On remarquera que le volume consommé entre deux impulsions est toujours égal à c .

Enfin, on n'effectuera ce processus que sur un échantillon à la fois : si plusieurs débits vérifiant le critère précédent se suivent, on ne décalera que la première impulsion intervenant dans le calcul de ces débits, sans modifier les suivantes. Un tel cas de figure signifie en effet qu'il existe soit un faible débit continu, soit une consommation intermittente de faibles volumes; il n'est pas possible de distinguer ces deux cas en l'état, mais il n'est pas non plus souhaitable de les négliger.

La fonction ne devra pas modifier la liste des impulsions. On rappelle que lorsque l'argument d'une fonction est une liste, alors la fonction peut modifier son argument. Pour éviter cela, il est possible de copier une liste avec une instruction du type `Lcopie = L[:]`.

Question 34 Écrire une fonction `debits2(Limp, c, seuil)` prenant pour arguments la liste `Limp` des instants des impulsions et deux flottants `c` et `seuil`, qui renvoie un tuple contenant :

- la liste des instants des impulsions modifiés comme indiqué ci-dessus (l'argument `Limp` ne doit pas être modifié),
- et la liste des débits calculés comme indiqué ci-dessus.

Comme indiqué précédemment, on ne modifiera pas plus d'une impulsion d'affilée (mais on pourra modifier plusieurs impulsions dans la liste).

```

1 def debits2(Limp, c, seuil):
2     impulsions = [Limp[0]] # Conteneur des instants des impulsions modifiés
3     debits = [] # Conteneur des débits
4
5     debitprecedentmodifie = False # Drapeau pour ne pas modifier deux débits d'affilée
6
7     # On fait l'hypothèse que le débit entre T0 et T1 n'est pas faible
8     debit = c / (Limp[1] - Limp[0])
9     debits.append(debit)

```

```

10 impulsions.append(LImp[1])
11
12 for i in range(2, len(lImp)):
13
14     debit = c / (lImp[i] - lImp[i-1])
15     if debit < seuil and not debitprecedentmodifie: # Cas de faible débit non continu
16         # On déplace impulsions[i-1] pour conserver le volume
17         deltat = c / debits[-1] # Temps pour consommer le volume c avec le débit précédent
18         impulsions[-1] = impulsions[-1] + deltat
19         impulsions.append(lImp[i])
20         debits.append(0.0)
21         debitprecedentmodifie = True
22     else: # Cas de débit non faible (ou de débit faible continu)
23         debitprecedentmodifie = False
24         impulsions.append(lImp[i])
25         debits.append(debit)
26
27 return (impulsions, debits)

```

Pour l'exemple précédent, le résultat obtenu est donné Figure 5.

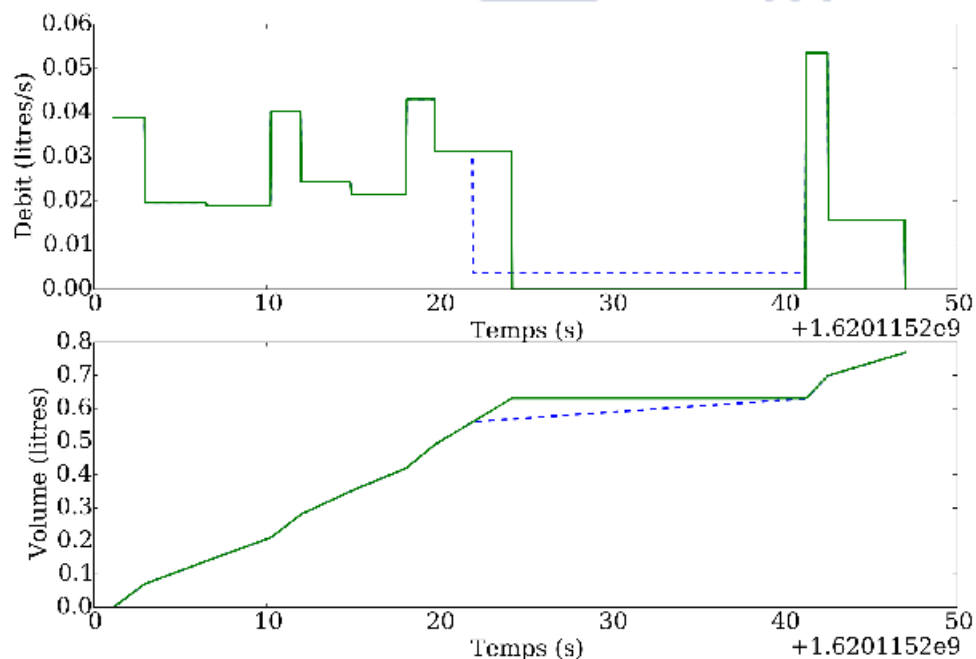


FIGURE 5 – Évolution temporelle avec prise en compte des périodes sans consommation

D Transmission des données et consultation à distance (30%)

Les données sur la consommation élaborées dans la partie précédente font ensuite l'objet d'un chiffrement, puis sont transmises à un serveur dans le but de permettre un suivi à distance de la consommation. Ces opérations sont l'objet de cette partie.

D.1 Chiffrement et déchiffrement des données

Les données sur la consommation d'eau ayant un caractère confidentiel, elles font l'objet d'un chiffrement avant d'être transmises au serveur distant. L'algorithme utilisé est le chiffrement XOR, ou "OU exclusif".

On rappelle que le OU exclusif, noté $\oplus$ et représenté par l'instruction $\wedge$ en Python, est un opérateur logique binaire qui renvoie 1 (ou Vrai) si les deux opérandes sont différents, 0 (ou Faux) s'ils sont identiques. Ainsi, $0 \wedge 0$ et $1 \wedge 1$ renvoient tous deux 0, tandis que $1 \wedge 0$ et $0 \wedge 1$ renvoient tous deux 1.

L'opérateur $\wedge$ permet d'appliquer un OU exclusif bit à bit entre deux entiers : le OU exclusif est alors appliqué sur les écritures binaires respectives des deux entiers, entre chaque paire de bits de même rang.

Considérons par exemple l'instruction $6 \wedge 3$. Si l'on écrit les bits de poids fort en premier (le résultat ne dépend pas de la convention choisie) :

- 6 a pour écriture binaire $(110)_2$;
- 3 a pour écriture binaire $(011)_2$;
- le OU exclusif bit à bit donne alors $1 \wedge 0$ (poids fort) puis $1 \wedge 1$ puis $0 \wedge 1$ (poids faible), soit $(101)_2$, soit 5.

Question 35 Déterminer le résultat renvoyé par l'instruction $13 \wedge 7$.

- 13 a pour écriture binaire $(1101)_2$;
- 7 a pour écriture binaire $(0111)_2$;
- le OU exclusif bit à bit donne alors $(1010)_2$, soit 10.

On considère dans cette partie que les données à transmettre consistent en une liste d'entiers, et on se donne un autre entier nommé clé. Le chiffrement XOR consiste à appliquer un OU exclusif bit à bit entre les données à transmettre et la clé.

La façon la plus simple de procéder consiste à appliquer un OU exclusif bit à bit entre chacun des entiers à transmettre et la clé.

Question 36 Écrire une fonction `code1(L,cle)` prenant pour arguments une liste L d'entiers et un entier cle, et renvoyant la liste des éléments de L chiffrés à l'aide de cle.

```
1 def code1(L, cle):
2     res = [] # Conteneur des éléments chiffrés
3     for entier in L:
4         res.append(entier ^ cle)
5     return res
```

Question 37 Justifier que, si a et b sont deux bits valant 0 ou 1, $(a \oplus b) \oplus b = a$ (on utilisera par exemple une table de vérité). Indiquer alors comment décoder une liste codée par l'instruction $L_c = \text{code1}(L, \text{cle})$ et préciser pourquoi cet algorithme de chiffrement est dit symétrique.

À l'aide d'une table de vérité :

a	b	$a \oplus b$	$(a \oplus b) \oplus b$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

Les deux colonnes a et $(a \oplus b) \oplus b$ sont identiques, on a donc :

$$\forall (a, b) \in \{0, 1\}^2, \quad (a \oplus b) \oplus b = a$$

Ainsi, pour décoder une liste codée par $L_c = \text{code1}(L, \text{cle})$, il suffit de lui réappliquer le codage par la même clé : $L_{\text{decode}} = \text{code1}(L_c, \text{cle})$.

Cet algorithme de chiffrement est dit symétrique, car les opérations de codage et de décodage sont les mêmes.

En pratique, la fonction `code1` n'est pas un moyen de chiffrement sûr, car tous les éléments de la liste sont chiffrés de la même façon. S'il existe une ressemblance entre ces éléments (par exemple s'ils sont proches les uns des autres) et si l'on dispose d'informations sur les données, il est possible d'identifier la clé puis d'en déduire la totalité du message. Pour y remédier, on se propose d'appliquer le chiffrement non pas à chaque élément de la liste (ce que l'on appelle le chiffrement par blocs) mais à la totalité des données mises bout à bout, en répétant pour cela la clé autant de fois que nécessaire (ce que l'on appelle le chiffrement par flux).

Considérons par exemple la suite d'entiers positifs (12, 3, 5, 8) codés sur 4 bits, avec la clé 2 codée sur 3 bits. Écrivons ces entiers en binaire les uns à la suite des autres, puis la clé (010) répétée tout au long du message (la dernière répétition est ici partielle), et enfin le résultat du OU exclusif bit à bit :

```
1100 0011 0101 1000
0100 1001 0010 0100
1000 1010 0111 1100
```

Enfin, on interprète le résultat binaire final comme une suite d'entiers codés sur 4 bits (comme la liste de départ). On obtient donc (8, 10, 7, 12).

Dans l'exemple ci-dessus, la clé étant écrite sur 3 bits et les données sur 4 bits, chaque entier est codé de façon différente. Ce principe permet d'obtenir un chiffrement relativement sûr si les nombres de bits utilisés sont premiers entre eux et suffisamment élevés.

On souhaite programmer cet algorithme pour transmettre des entiers positifs codés sur 64 bits. Les représentations binaires seront codées sous la forme de chaînes de caractères contenant des 0 et des 1. On rappelle que :

- la fonction `bin` permet de convertir un entier en binaire : `bin(10)` renvoie `'0b1010'`, le préfixe `'0b'` indiquant une écriture binaire. On remarquera que l'écriture utilise le nombre minimal possible de bits, de sorte que le bit de poids fort est toujours 1 ;
- la fonction `int` permet de convertir une représentation binaire en entier, si l'on indique la base de numération (ici 2) en second argument : `int('0b1010', 2)` et `int('1010', 2)` renvoient tous deux 10.

On suppose que tous les entiers dont il est question dans cette partie sont positifs et représentables sur 64 bits.

Question 38 Écrire une fonction `bin64(n)` prenant pour argument un entier positif `n`, et retournant une chaîne de caractères contenant son écriture binaire sur 64 bits (c'est-à-dire complétée par autant de zéros que nécessaire), sans le préfixe `'0b'`. On utilisera la fonction `bin`.

```
1 def bin64(n):
2     rbin = bin(n)[2:] # Représentation binaire de n sans le "0b"
3
4     rbin64 = "0" * (64 - len(rbin)) + rbin # Ajout des 0 nécessaires en suffixe
5
6     return rbin64
```

Question 39 Écrire une fonction `bits(L)` prenant pour argument une liste `L` d'entiers positifs, et retournant une chaîne de caractères constituée des écritures binaires des éléments de `L` sur 64 bits, concaténées les unes aux autres. On utilisera la fonction `bin64`.

```
1 def bits(L):
2     res = "" # Chaîne de caractères contenant la concaténation des représentations sur 64 bits des
3             # nombres de L
4
5     for nombre in L:
6         res += bin64(nombre)
7
8     return res
```

Question 40 Écrire une fonction `entiers(flux)` prenant pour argument une chaîne `flux` au format précédent (constituée des caractères '0' et '1' et de longueur multiple de 64), et renvoyant la liste des entiers dont les codages respectifs sur 64 bits se trouvent bout à bout dans la chaîne `flux`. On utilisera la fonction `int`.

```

1 def entiers(flux):
2     res = [] # Conteneur des entiers au format int
3
4     longueur = len(flux) // 64 # Nombre d'entiers dans flux
5
6     for i in range(longueur):
7         r64 = flux[i*64:(i+1)*64] # Représentation binaire sur 64 bits du i-ème entier
8         rdec = int(r64, 2) # Transcodage
9         res.append(rdec)
10
11     return res

```

Question 41 Si l'on appelle l la longueur de la clé, quelle est la position du bit de la clé (entre 0 et $l - 1$) correspondant au bit de rang i (compté à partir de zéro) de la liste à chiffrer ? On donnera la réponse sous la forme d'une opération arithmétique simple.

L'indice du bit de la clé est le reste de la division entière de i par l , soit, en Python : `i % l`.

Pour que l'algorithme précédent réalise un chiffrement sûr, on souhaite que le nombre de bits sur lequel est écrite la clé soit premier avec 64, c'est-à-dire impair.

Question 42 Écrire une fonction `code2(L,cle)` prenant pour arguments une liste `L` d'entiers et un entier `cle`, et renvoyant une liste contenant le résultat du chiffrement par flux de `L` à l'aide de `cle` selon l'algorithme décrit ci-dessus. L'entier `cle` sera écrit en binaire sur un nombre de bits impair (on ajoutera donc un 0 en tête de son écriture binaire si cela est nécessaire). On utilisera entre autres les fonctions `bits` et `entiers`.

```

1 def code2(L, cle):
2     flux = bits(L)
3
4     # Mise en forme de la clé
5     cleb = bin(cle)[2:]
6     if (len(cleb) % 2) == 0:
7         cleb = "0" + cleb
8
9     # Codage bit à bit sur les chaînes de caractère
10    fluxc = ""
11    for i in range(len(flux)):
12        fluxc += str(int(flux[i]) ^ int(cleb[i % len(cleb)]))
13
14    return entiers(fluxc) # Transcodage du flux chiffré

```

Une limitation importante du chiffrement par OU exclusif est que si l'on dispose d'un message et de sa version chiffrée, alors il est très facile de retrouver la clé de chiffrement. Pour cette raison, sur les applications présentant un risque particulier (ce qui n'est pas le cas ici), la clé de chiffrement est à usage unique (l'algorithme est alors dit à masque jetable).

Question 43 À partir des résultats précédents et en vous aidant notamment de la réponse à la question 37, expliquer simplement comment il est possible de reconstruire la clé si l'on dispose d'un message et de son chiffrement par la fonction `code1`. Faire de même pour la fonction `code2`.

Le codage étant symétrique, il suffit pour le chiffrement réalisé par la fonction `code1` d'appliquer un OU exclusif bit à bit entre le premier nombre du message et le premier nombre du message chiffré (ou entre n'importe quel nombre non chiffré et sa version chiffrée) pour obtenir la clé.

La difficulté dans le cas de `code2` est que l'on ne connaît pas la longueur de la clé. On peut toutefois tester

toutes les longueurs n possibles, en se limitant à n impair.

On prend $n = 1$, puis :

- On prend les n premiers bits du message chiffré, les n premiers bits du message non chiffré, et on réalise un OU bit à bit entre eux : on obtient alors la représentation binaire d'une clé potentielle.
- On teste le chiffrement du message avec cette clé :
 - ◊ Si on obtient un message identique au message chiffré, alors c'est la bonne clé ;
 - ◊ Sinon, on incrémente n de 2 (pour tester la longueur impaire supérieure), et on recommence.

Exemple d'implémentation :

```

1 def decodage2(L, Lc):
2     decode = False
3     n = 1
4     while not decode:
5         cle1 = bits(L)[:n]
6         cle2 = bits(Lc)[:n]
7         clepotentielle = int(cle1, 2) ^ int(cle2, 2)
8         if (code2(L, clepotentielle) == Lc):
9             decode = True
10        else:
11            n += 2
12
13    print("Clé trouvée : %u" % clepotentielle)
14    return clepotentielle

```

D.2 Consultation des données

Un particulier souhaite suivre précisément sa consommation d'eau et enregistre à cet effet chaque jour la quantité consommée. Les données ainsi recueillies sont enregistrées dans une base de données. On donne ci-dessous un extrait de la table `eau` utilisée.

Table eau (extrait)		
id	date	conso
...	...	...
36	2017-11-11	497
37	2017-11-12	156
38	2017-11-13	392
39	2017-11-14	274
40	2017-11-15	343
41	2017-11-16	296
42	2017-11-17	153
...	...	...

Cette table contient :

- un identifiant associé à chaque mesure, de type entier,
- la date, au format `aaaa-mm-jj`, de type chaîne de caractères,
- la consommation (en litres), de type flottant.

Question 44 Écrire en SQL les requêtes permettant d'obtenir :

1. la consommation le 17 novembre 2017,
2. les dates des consommations supérieures à 500 litres.


```
1 SELECT conso FROM eau WHERE date = "2017-11-17";  
2 SELECT date FROM eau WHERE conso > 500;
```

Question 45 L'opérateur LIKE permet de faire des comparaisons partielles entre chaînes de caractères, et en particulier de remplacer une partie d'une chaîne par un %, la comparaison étant vraie quel que soit la partie de chaîne comparée avec le % (par exemple 'informatique' LIKE 'info%' est vrai). Écrire les requêtes permettant d'obtenir :

1. les consommations de tous les jours du mois de novembre 2017,
2. les consommations totale et moyenne du mois de novembre 2017.

```
3 SELECT conso FROM eau WHERE date LIKE "2017-11-%" ;  
4 SELECT SUM(conso), MEAN(conso) FROM eau WHERE date LIKE "2017-11-%";
```

Question 46 On envisage maintenant de gérer les consommations de tout un ensemble d'habitations (par exemple, la mairie enregistre les informations de chaque logement de la commune). Expliquer comment organiser les données en utilisant, en plus de la table `eau`, une table `habitations` dont on précisera les attributs les plus importants. Quel attribut faudra-t-il ajouter à la table `eau`? Expliquer soigneusement quelle sera la condition de jointure utilisée pour faire des requêtes sur ces deux tables.

La table `habitations` doit contenir, outre les informations utiles concernant chaque habitation (adresse, nom de l'habitant, informations de facturation...), un identifiant unique qui sera utilisé comme clé primaire. On peut par exemple utiliser le numéro du compteur d'eau, si celui-ci est bien unique parmi tous les compteurs de l'ensemble des habitations. Pour l'exemple, appelons cette colonne `id`.

La table `eau` contenant tous les relevés, doit permettre d'associer à chaque relevé l'habitation concernée : il faut donc y ajouter une colonne qui contiendra l'`id` de l'habitation concernée (qui sera donc une clé étrangère). Pour l'exemple, appelons cette colonne `habitation`.

On pourra alors obtenir l'ensemble des consommations du 17 novembre 2017 par habitation en réalisant la jointure `habitations.id = eau.habitation` :

```
5 SELECT habitation.adresse, eau.conso FROM eau JOIN habitations ON habitations.id = eau.habitation WHERE  
   date = "2017-11-17";
```

Ou encore, pour la facturation mensuelle, le total de la consommation de chaque habitation sur le mois de novembre 2017 :

```
6 SELECT habitation.adresse, SUM(eau.conso) FROM eau JOIN habitations ON habitations.id = eau.habitation  
   WHERE date LIKE "2017-11-%" GROUP BY habitations.id;
```