المملكة المغربية

ROYAUME DU MAROC

Ministère de l'Enseignement Supérieur de la Recherche
Scientifique de la Formation des Cadres

Présidence du Concours National Commun
Ecole Nationale Supérieure des Mines de Rabat

CONCOURS NATIONAL COMMUN

d'admission aux Établissements de Formation d'Ingénieurs et Établissements Assimilés

Session 2016

ÉPREUVE D'INFORMATIQUE

Filières : **MP/PSI/TSI**

Durée **2** heures

Cette épreuve comporte 10 pages au format A4, en plus de cette page de garde
L'usage de la calculatrice est **interdit**

L'énoncé de cette épreuve, commune aux candidats des filières MP/ PSI/ TSI, comporte 10 pages.
L'usage de la calculatrice est interdit .

*Les candidats sont informés que la précision des raisonnements **algorithmiques** ainsi que le soin apporté à la rédaction et à la présentation des copies seront des éléments pris en compte dans la notation. Il convient en particulier de rappeler avec précision les* références *des questions abordées. Si, au cours de l'épreuve, un candidat repère ce qui peut lui sembler être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.*

Remarques générales :

- L'épreuve se compose de deux problèmes indépendants.
- Toutes les instructions et les fonctions demandées seront écrites en **Python**.
- Les questions non traitées peuvent être admises pour aborder les questions ultérieures.
- Toute fonction peut être décomposée, si nécessaire, en plusieurs fonctions

PROBLÈME I : CALCUL SCIENTIFIQUE

✧✧✧

Méthodes à un pas :

Nous allons nous intéresser dans ce problème, à quelques méthodes permettant de résoudre numériquement les équations différentielles **du premier ordre**, avec une condition initiale, sous la forme :

$$\begin{cases} y'(t) = & f(t, y(t)) \\ y(t_0) = & y_0 \end{cases}$$

On note $I =]t_0, t_0 + T[$ l'intervalle de résolution. Pour un nombre de noeuds N donné, soit $t_n = t_0 + nh$, avec $n = 0, 1, 2, ..., N$, une suite de noeuds de I induisant une discrétisation de I en sous-intervalles $I_n = [t_n, t_{n+1}]$. La longueur h de ces sous-intervalles est appelée pas de discrétisation, le pas h de discrétisation est donné par $h = \frac{T}{N}$. Soit y_j l'approximation au noeud t_j de la solution exacte $y(t_j)$. Les méthodes de résolution numériques, étudiées dans ce problème, s'écrire sous la forme :

$$\begin{cases} y_{n+1} = & y_n + h\Phi(t_n, y_n, h) \\ t_{n+1} = & t_n + h \end{cases}$$

Avec

$$\Phi(t,y,h) = \alpha f(t,y) + \beta f(t+h, y+hf(t,y));$$

où α, β sont des constantes comprises entre 0 et 1.

Question 1 :

Pour quelles valeurs du couple (α,β) retrouve-t-on la méthode d'Euler ?

Dans la suite on considère une deuxième méthode dite de **Heun**, cette méthode correspond aux valeurs du couple $(\alpha,\beta) = (\frac{1}{2}, \frac{1}{2})$.

Question 2 :

Écrire une fonction de prototype **def Heun(f, t_0, y_0, T, N)** : qui prend en paramètres la fonction $f : (t,y) \rightarrow f(t,y)$, la condition initiale t_0, y_0 la valeur de f en t_0 le nombre de nœuds N et la valeur final du temps T et qui retourne deux listes $tt = [t_0, t_1, \ldots, t_N]$ et $y_h = [y_0, y_1, \ldots, y_N]$ donné par le schéma :

$$\begin{cases} y_{n+1} = & y_n + h\Phi(t_n, y_n, h) \\ t_{n+1} = & t_n + h \end{cases}$$

Avec

$$\Phi(t,y,h) = \frac{1}{2} f(t,y) + \frac{1}{2} f(t+h, y+hf(t,y));$$

Le pas de discrétisation h est donné par $h = \frac{T}{N}$.

Remarque : La valeur retournée par la fonction **Heun** pourra être un couple constitué de deux listes, un tableau constitué de deux listes (par exemple un type *array* du module *numpy*) ou tout autre structure de données constitué de deux listes.

Application au circuit RLC :

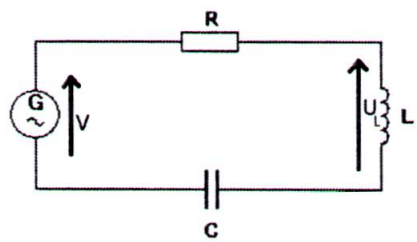

FIGURE 1: circuit RLC

On souhaite travailler sur un circuit *RLC* en série qui sera constitué des éléments suivants : une résistance *R(1Ω)*, une inductance *L(1H)*, une capacité *C(1F)* et un générateur qui délivrera une source de tension sinusoïdale $V(t) = \cos(2\pi t)$.

On s'intéresse à la tension U_l aux bornes de la bobine qui satisfait l'équation différentielle :

$$\ddot{U}_l + \dot{U}_l + U_l = -4\pi^2 \cos(2\pi t) \qquad \text{(I)}$$

Question 3 :

Écrire une fonction de prototype **def Euler(f, t_0, T, y_0, N)** : qui prend en paramètres la fonction $f : (t, y) \rightarrow f(t, y)$, la condition initiale t_0, la valeur de f en t_0, le nombre de nœuds N et la valeur final du temps T et qui retourne deux listes $tt = [t_0, t_1, \ldots, t_N]$ et $y_h = [y_0, y_1, \ldots, y_N]$ donné par le schéma d'*Euler* permettant de résoudre des équations différentielles du premier ordre.

Question 4 :

Donner la complexité de la fonction d'*Euler* en expliquant le descriptif du calcul.

On considère les valeurs suivantes : $y(t) = U_l(t)$ et $z(t) = \dot{U}_l(t)$. L'équation différentielle (I) est du second ordre.

Question 5 :

Définissez un système de deux équations différentielles du premier ordre qui puisse satisfaire l'équation (I).

Question 6 :

En posant $Y(t) = [U_l(t), \dot{U}_l(t)]$, montrez que l'équation (I) peut s'écrire sous la forme :

$$Y'(t) = F(t, Y(t)) \qquad \text{(II)}$$

avec $F : t, X \rightarrow F(t, X)$ est une fonction à préciser (X = [X[0], X[1]]).

Question 7 :

Implémenter en Python l'équation différentielle (II) en utilisant la fonction *odeint()* du module *scipy.integrate*, la méthode d'*Euler* et celle de *Heun*. On affichera dans une même fenêtre le résultat de ses trois méthodes avec les valeurs suivantes :

N = 1000, $t_0 = 0$, T = 3, $U_l(0) = 0$ et $\dot{U}_l(0) = 0$. Voir les résultats sur la *figure 1* se trouvant dans l'annexe du document.

On ajoutera quatre types d'informations :

- le titre de la figure : 'circuit RLC, tension bobine'
- un label associé à l'axe des abscisses : 'temps (s)'
- un label associé à l'axe des ordonnées : 'tension (mV)'
- une légende associée au graphique : '$Odeint$' associé à la méthode d'*odeint*
- une légende associée au graphique : '$Euler$' associé à la méthode d'*Euler*
- une légende associée au graphique : '$Heun$' associé à la méthode d'*Heun*

PROBLÈME II : PROGRAMMATION

✧✧✧

Remarque 2.1 *Dans ce problème il ne sera pas possible d'utiliser les méthodes de la classe String suivantes : capitalize(), upper(), lower(), encode(), replace(), swapcase(), translate() . Par contre l'utilisation des autres méthodes est possible comme la méthode index().*

Nous allons nous intéresser au code de *Vigenère* (1523-1596) qui est plus robuste que le code de *César*, le premier code reconnu comme élément à coder les messages. Nous considérons dans ce problème uniquement l'alphabet de la langue française. On ne s'intéressera qu'aux caractères visibles. On ne prendra pas en compte les ligatures comme par exemple œ ou æ. On ne s'intéresse qu'à des textes composés uniquement de majuscules, les 26 lettres de l'alphabet. Tout autre caractère sera supprimé du texte initial.

Le code de *César* repose sur une substitution mono alphabétique particulière, c'est-à-dire qu'on remplace une lettre de l'alphabet par une autre en utilisant une clé. Chaque lettre de l'alphabet a une valeur entière (A=0, B=1, C=2, D=3, ..., K=10, ..., T=19, ..., Z=25) que l'on ajoute à une clé modulo 26 ($x = y mod 26 \iff x \equiv y[26]$). Par exemple : si la clé est K (sa valeur est 10, position de la lettre K dans l'alphabet), la lettre T aura comme valeur chiffrée : $19 + 10 mod 26 = 29 mod 26 = 3$, soit le caractère D.
Le code de *Vigenère* est construit sur le même principe mais il permet de donner une réponse au point faible du code de *César*, à savoir l'analyse des fréquences de l'apparition des lettres. Le code de *Vigenère* utilise des clés plus longues que celui de *César*. Le principe est que chaque lettre peut être codée de plusieurs façons. Par exemple, prenons une clé de dimension 3 (son nombre de lettres). Soit *'ABC'* cette clé alors toutes les lettres en position $3k$ seront codées selon le code de *César* avec 'A' comme clé, toutes les lettres en position $3k+1$ seront codées avec la clé 'B' et toutes les lettres en position $3k+2$ avec la clé 'C'.

Par exemple : comment chiffrer le texte "c'est un bel été" avec la clé *'ABC'* ?

Avant d'appliquer le code de *Vigenère* le texte en clair sera converti en majuscules et ceci sans aucune lettre accentuée ni d'espacement ni autre caractère.

Le texte sera donc transformé en : "CESTUNBELETE".

Appliquons le codage de *Vigenère* : la clé sera répétée autant de fois que nécessaire afin que le texte composé de la répétition globale ou partielle de la clé soit de même taille que le texte initial.

On applique le code de *César* à chaque lettre du texte en clair associée à une lettre de la clé. Par exemple si on prend la deuxième lettre du texte en clair soit 'E' (indice 4 de l'alphabet), associée à la lettre correspondante de la clé, soit 'B' (indice 1), le chiffré est calculé comme :

Texte en clair :	CESTUNBELETE
La clé :	ABCABCABCABC
Texte crypté :	CFUTVPBFNEUG

TABLE 2.1: Un message crypté

$4 + 1 mod\ 26 = 5$, soit la lettre F. On réitère le processus sur l'ensemble des autres lettres du texte en clair. Voir la table 2.1 pour la description du mécanisme.

Première partie

Dans la suite du problème nous utiliserons la phrase suivante définie à l'aide de la variable texte.

In [1] : texte = "Hey Hey, cet été là, le gang des niçois a été arrêté !"

In [2] : texte

Out[3] : 'Hey Hey, cet été là, le gang des niçois a été arrêté !'

Question 8 :

Écrire une fonction ***enMajuscule(CH)*** qui transforme tous les caractères de la chaine de caractères **CH** passée en tant que paramètres en une chaine composée uniquement de majuscules. On transformera tous les caractères accentués en majuscules non accentuées. Par exemple le caractère 'Ï' sera transformé en caractère 'I'. Tous les autres caractères seront conservés comme par exemple les caractères de ponctuation.

À titre indicatif voici la liste des caractères accentués : âàéèêëïîôùûüÿ. On ajoutera à cette liste le caractère ç (c cédille). Attention on ne pourra pas utiliser la méthode upper() de la classe *String* pour traiter cette question et les suivantes.

Exemple :

In [4] : enMajuscule(texte)

Out[5] : 'HEY HEY, CET ÉTÉ LA, LE GANG DES NICOIS A ETE ARRETE !'

Question 9 :

Écrire la fonction ***majusculesSeules(CH)*** qui prend une chaine de caractères **CH**, comme paramètre et retourne une chaine de caractères composée **uniquement** des lettres de l'alphabet : ABCDEFGHIJKLMNOPQRSTUVWXYZ.

Exemple :

In [6] : majusculesSeules(texte)

Out[7] : 'HEYHEYCETETELALEGANGDESNICOISAETEARRETE'

Question 10 :

Écrire la fonction ***vigenereEncode(CH,CL)*** qui a deux paramètres de type chaine de caractères, la chaine de caractères **CH**, à encoder suivant le critère de *Vigenère* et la clé **CL** . Cette fonction retourne la chaine encodée.

Remarque : la clé sera uniquement composée de lettres de l'alphabet : ABCDEFGHIJKLMNOPQRSTUVWXYZ.

Exemple :

In [8] : vigenereEncode(texte, "ABC")

Out[9] : 'HFAHFACFVEUGLBNEHCNHFETPIDQITCEUGASTEUG'

Question 11 :

Calculer la complexité de la fonction ***vigenereEncode()***.

Question 12 :

Écrire la fonction ***vigenereEncode()*** de manière récursive. On nommera cette fonction ***vigenereEncodeRec()*** qui comportera autant de paramètres que nécessaire. Vous devez **changer et commenter** les différents paramètres d'entrées que vous allez utiliser.

Pour cette question le texte à traiter ne sera composé que de lettres de l'alphabet : ABCDEFGHIJKLMNOPQRSTUVWXYZ

Question 13 :

Calculer la complexité de la fonction ***vigenereEncodeRec()***.

Question 14 :

Écrire la fonction ***vigenereDecode(CH,CL)*** qui a deux paramètres de type chaines de caractères : la chaîne de caractères **CH**, à décoder suivant le critère de ***Vigenère*** et la clé **CL**. Cette fonction retourne le message en clair.

Exemple :

In [10] : vigenereDecode("HFAHFACFVEUGLBNEHCNHFETPIDQITCEUGASTEUG", "ABC")

Out[11] : 'HEYHEYCETETELALEGANGDESNICOISAETEARRETE

Question 15 :

Indiquer comment on doit choisir la clé afin que le message soit considéré comme robuste à une personne malveillante qui voudrait cracker ce message.

Seconde partie

On s'intéresse maintenant à la manière de casser le code de *Vigenère*, c'est-à-dire à partir du texte crypté de pouvoir déterminer la clé. On peut remarquer que le codage d'une même lettre du texte à crypter avec une lettre se trouvant à une même position de la clé est toujours le même. Ainsi si p est la longueur de la clé utilisée pour coder un texte on peut remarquer que toutes les lettres identiques aux positions $k * l + p$ avec $0 \leqslant l \leqslant p - 1$ sont codées par la même lettre pour une valeur l spécifique. Une analyse des fréquences pour les différentes valeurs de l nous permet alors, de deviner la clé utilisée. La table 2.1 donne un exemple d'un message crypté avec une clé de longueur 3. Si la clé est de longueur p, on aura p lettres pour crypter le message. La première lettre du message sera cryptée à l'aide de la première lettre de la clé. La seconde lettre du message sera cryptée à l'aide de la seconde lettre de la clé, et ainsi de suite.

0	p	2*p	...	k*p	...
1	1 + p	1 + 2*p	...	1 + k*p	...
2	2 + p	2 + 2*p	...	2 + k*p	...
...					
i	1 + p	1 + 2*p	...	1 + k*p	...
...					
p-1	p-1 + p	p-1 + 2*p	...	p-1 + k*p	...

TABLE 2.2: Mécanisme de cryptage

Tous les caractères du message distants de p caractères utiliseront une même lettre de la clef pour réaliser le chiffrement. La table 2.2 présente le mécanisme d'un tel chiffrement. Il est possible d'utiliser p lettres différentes pour coder chacune des séquences, cela dépend de la nature de la clé.

Afin de simplifier le problème nous considérerons, par la suite, que la dimension de la clé est connue. Nous allons mettre en œuvre une méthode relativement simple qui ne fonctionnera que dans un cadre bien spécifique. Voici un message crypté dont vous devez retrouver la clé : message = 'HFAHFACFVEUGLBNEHCNHFETPIDQITCEUGASTEUG'

Question 16 :

Écrire une fonction ***sousChaines(CH,d,p)*** qui prendra en compte trois paramètres : **CH** le texte à déchiffrer, **d** le déplacement (c'est-à-dire la lettre utilisée de la clé) et **p** la dimension de la clé. Cette fonction retourne une chaîne de caractères. Par exemple : à l'indice *d* de la clé on va extraire du texte chiffré tous les caractères qui sont espacés d'une distance *p*, la dimension de la clé.

Autrement dit cette fonction rend la sous chaîne composée des caractères du texte aux indices d, $d + p$, $d + 2p$, etc.

Exemple :

In [12] : sousChaines(message, 1,3)

Out[13] : 'FFFUBHHTDTUSU'

Question 17 :

Écrire la fonction ***listeDesSousChaines(CH,d)*** qui prendra en compte deux paramètres, **CH** le texte crypté et **d** la dimension de la clé. Cette fonction retourne comme valeur un tableau avec l'ensemble de toutes les sous-chaines définies à la question précédente.

Exemple :

In [14] : listeDesSousChaines(message, 3)

Out[15] : ['HHCELENEIIEAE', 'FFFUBHHTDTUSU', 'AAVGNCFPQCGTG']

Question 18 :

C'est le caractère 'E' qui apparaît le plus souvent dans un texte français. Les caractères suivants, les plus fréquents, sont dans l'ordre ['A', 'S', 'I', 'T', 'N']. Nous nous limiterons uniquement au caractère 'E'. Dans chaque sous-chaine de caractères constituée à la question précédente

il suffit de connaître la fréquence des caractères afin de déterminer quel est le caractère qui apparaît le plus. Chacun de ces caractères correspond à un chiffrement déterminé à partir du caractère 'E'.

Écrire la fonction ***frequenceCaracteres(CH)*** qui prend comme paramètre un texte crypté **CH** et retourne comme valeur un tableau constitué des fréquences de chaque caractère de l'alphabet de ce texte.

Exemple :

In [16] : frequenceCaracteres(message)

Out[17] : [3,1,3,1,5,4,3,4,2,0,0,1,0,2,0,1,1,0,1,3,3,1,0,0,0,0]

Question 19 :

À l'aide de la question précédente écrire une fonction qui permettra l'extraction de la clé. Afin de simplifier le traitement on fera référence uniquement à la lettre 'E' comme le caractère qui apparaît le plus dans le texte en clair.

Cette fonction que l'on nomme ***code(CH,p)*** a deux paramètres : **CH** le texte crypté et **p** la longueur de la clé. cette fonction retourne une chaine de caractères, la valeur de la clé.

Exemple :

In [18] : code(message, 3)

Out[19] : 'ABC'

On retrouve bien la clé initiale à partir du texte crypté.

Annexe

A) Image du document

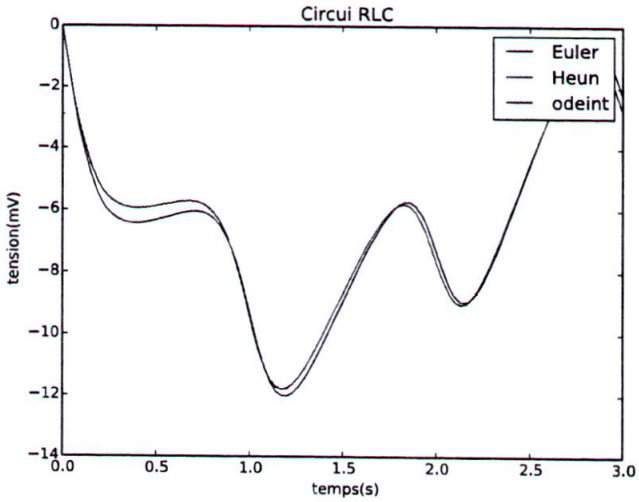

FIGURE 1: circuit RLC

B) Syntaxe de quelques fonctions du langage Python

1) matplotlib.pyplot.xlabel

matplotlib.pyplot.xlabel(s, *args, **kwargs)

Parameters : **s** : string.

Returns : Set the x axis label of the current axi.

2) matplotlib.pyplot.ylabel

matplotlib.pyplot.ylabel(s, *args, **kwargs)

Parameters : **s** : string.

Returns : Set the y axis label of the current axi.

3) matplotlib.pyplot.plot

matplotlib.pyplot.plo(*args, **kwargs)

Parameters : ***args** :
args is a variable length argument, allowing for multiple x, y pairs with an optional format string.
y0 : array
Initial condition on y (can be a vector).

character : ***description***
'-' : solid line style
'–' : dashed line style
'-.' : dash-dot line style
' :' : dotted line style
'.' : point marker
',' : pixel marker
'o' : circle marker
'v' : triangle_down marker

character : ***color***
'b' : blue
'g' : green
'r' : red

Returns Return value is a list of lines that were added.

4) scipy.integrate.odeint

scipy.integrate.odeint(func, y0, t, args=(), Dfun=None, col_deriv=0, full_output=0, ml=None, mu=None, rtol=None, atol=None, tcrit=None, h0=0.0, hmax=0.0, hmin=0, mxstep=0, mxhnil=0, mxordn=12, mxords=5, printmessg=0)

5) matplotlib.pyplot.legend

matplotlib.pyplot.legend(*args, **kwargs)

Parameters :
Returns : Places a legend on the axes.
To make a legend for lines which already exist on the axes (via plot for instance), simply call this function with an iterable of strings, one for each legend item.

6) matplotlib.pyplot.title

matplotlib.pyplot.title(s, *args, **kwargs)

✧✧✧

FIN DE L'ÉPREUVE