ORAL « *Mathématiques et algorithmique* » de la Banque PT
Rapport 2016

Les futurs candidats trouveront dans ce rapport des remarques et des conseils qui pourraient leur être utiles pour leur futur passage. Ce rapport n'est pas exhaustif et ne met l'accent que sur quelques points jugés importants par l'équipe d'interrogateurs de cet oral. Nous suggérons aux futurs candidats de lire également les rapports des années antérieures et de consulter le site de la Banque PT : http://www.banquept.fr/spip.php?article237 où ils trouveront les mémentos, disponibles lors de l'oral, et les exercices types d'informatique. Certains exercices d'algorithmique et de simulation numérique posés en 2015 et 2016 sont également fournis en annexe du présent rapport.

Intitulé

La durée de cet oral « *Mathématiques et algorithmique* » est de 1 heure, préparation incluse.

Il comporte deux exercices :

- l'un porte sur le programme de mathématiques de la filière PTSI/PT ;
- l'autre exercice porte sur les items 2, 3 et 5 du programme d'informatique.

Objectifs

Le but d'une telle épreuve est d'abord de contrôler l'assimilation des connaissances des programmes de mathématiques et d'informatique (items 2, 3 et 5) de toute la filière (première et deuxième années). Il semble que certains candidats aient hélas « *oublié* » ce qui a été vu en première année, voire les connaissances de base qui font partie du programme des classes du lycée (seconde, première, terminale).

Cette épreuve permet aussi d'examiner :

- la capacité d'initiative du candidat ;
- son aisance à exposer clairement ses idées avec un vocabulaire précis ;
- sa réactivité et son aptitude à communiquer dans un dialogue avec l'interrogateur ;
- son aptitude à mettre en œuvre ses connaissances et son savoir-faire pour résoudre un problème (par la réflexion et non par la mémorisation de solutions toutes faites) ;
- sa maîtrise des algorithmes et manipulations de base, des calculs sur des nombres entiers, décimaux ou complexes, et du langage de programmation pour mettre en œuvre une solution informatique ;
- sa faculté à critiquer, éventuellement, les résultats obtenus et à changer de méthode en cas de besoin.

Organisation

Cette dernière session s'est déroulée dans des conditions identiques aux sessions précédentes. Comme les autres années, elle a eu lieu dans les locaux de « *Arts et Métiers ParisTech* », 155 boulevard de l'Hôpital à Paris (13^{e}). En raison de l'état d'urgence, il n'a pas été possible cette année d'accueillir de futurs candidats ou des enseignants de classes préparatoires, hormis deux représentants de l'Union des Professeurs de classes préparatoires Scientifiques.

Les candidats avaient deux exercices à résoudre, classiques et ne faisant appel à aucune astuce particulière :

- un exercice de mathématiques « *au tableau* », portant sur le programme de mathématiques des deux années de la filière PT (algèbre, analyse, géométrie et probabilités), c'est-à-dire sur les programmes PTSI et PT ;
- un exercice d'algorithmique « *sur machine* », portant sur le programme d'informatique : algorithmique (items 2 et 5) avec l'utilisation du langage Python et simulation numérique (item 3) avec l'utilisation de l'environnement de simulation numérique (les bibliothèques *Numpy*/*Scipy*/*Matplotlib* de *Python*). Pour ce deuxième exercice, les candidats disposaient d'un ordinateur, sur lequel avaient été installés *Python 3.4* et ses bibliothèques ainsi que *Scilab 5.5* (aides incluses), des mémentos plastifiés et en couleurs au format A3, rendus publics bien avant l'oral, et de feuilles de brouillon, en général trop peu utilisées. L'environnement de développement était *IDLE*, comme annoncé depuis 2014, muni de l'extension *IDLEx* qui permet d'avoir les numéros de ligne. Quelques candidats ont avoué avoir préparé l'oral avec *Spyder* ou *Pyzo*, ce qui est un peu surprenant. Nous ne pouvons que conseiller de se placer dans les conditions de passage de l'oral (*IDLE*, avec l'extension *IDLEx* éventuellement, + Mémentos) tout au long des deux années de préparation. ***Aucun candidat n'a demandé à programmer en Scilab.***

Remarques générales

- Une indication est toujours une aide ; la capacité d'écoute et de réaction à celle-ci est un élément d'évaluation. De manière générale, la passivité, l'attentisme sont à proscrire lors de l'oral.
- Certains candidats ne sont pas assez attentifs :
    - en lisant trop vite le sujet, ils ne répondent pas à la question posée ;
    - ils n'écoutent pas les questions ou les consignes de l'interrogateur qui servent en général à les aider.

Mathématiques

Généralités

- On exige d'un candidat qu'il soit précis dans ses propos, et en particulier qu'il énonce une définition complète et un théorème avec l'ensemble des hypothèses.
- L'oral n'est pas un écrit sur tableau. Les justifications et commentaires doivent être donnés au moment où l'on est interrogé. Il est inutile d'écrire un long texte pour justifier une linéarité ou une continuité triviales.
- De nombreux candidats maîtrisent les techniques de calcul mais rencontrent des difficultés dès qu'il faut raisonner et/ou revenir à la définition d'un concept. On a pu observer une méconnaissance fréquente des définitions au profit des propriétés et théorèmes, un peu mieux connus.
- Le jury attend d'un candidat qu'il connaisse les résultats de cours au programme. Il s'agit d'être précis. Interrogés sur une définition, il faut éviter de commencer sa réponse par : « *c'est quand...* » ou « *c'est par exemple...* ».
- Les candidats doivent s'attendre à être interrogés sur la nature des objets qu'ils manipulent ; il faut pouvoir dire si on manipule un nombre, une fonction, un vecteur ; il n'est pas acceptable à ce niveau de confondre aire et primitive, ou de voir un candidat dériver un endomorphisme.
- Il est bon de connaître certaines inégalités usuelles comme : $\ln(1+x) \leq x$, $|\sin(x)| \leq |x|$, etc.

- Pour accélérer les calculs et éviter les erreurs, respecter quelques principes simples comme :
    - éviter au maximum les quotients et les racines carrées, raisonner sur le carré de la norme d'un vecteur ou du module d'un nombre complexe ;
    - lors d'une étude de signe, la faire toujours dans le même sens : par exemple, si l'on privilégie systématiquement les termes positifs, écrire $b-a>0$ plutôt que $a-b<0$ pour montrer que $a<b$, ou vérifier que u_n-u_{n+1} est positif pour montrer que la suite (u_n) est décroissante, plutôt que $u_{n+1}-u_n$ négatif ;
    - ne pas écrire $\lim(f(x))$ ou $\lim(u_n)$ avant d'avoir prouvé l'existence de cette limite ;
    - lorsqu'on cherche à prouver la convergence d'une série ou d'une intégrale, raisonner sur le terme général ou la fonction et non pas directement sur la somme ou l'intégrale.
    - etc.

Polynômes et nombres complexes

- La manipulation des nombres complexes pose trop souvent des difficultés.
- La résolution dans $\mathbb{C}$ d'équations polynomiales n'est pas toujours maîtrisée, et les racines n-ièmes de l'unité pas toujours connues.
- Les identités et factorisations remarquables devraient être mieux connues : x^2+2x+1, x^n-1, …
- La confusion entre « *polynôme scindé* » et « *polynôme scindé à racines simples* » est hélas trop répandue, révélant le manque de précision du langage évoqué plus haut.

Algèbre linéaire

- Certains candidats n'ont pas été capables d'exprimer dans une base donnée la matrice associée à une application linéaire.
- La plupart des candidats semble à l'aise avec la réduction de matrices, même si les techniques de calcul utilisées ne sont pas toujours optimales. Quelques lacunes ont été également observées sur la capacité à rendre une matrice de changement de base orthogonale dans le cas de la réduction de matrices symétriques à coefficients réels.
- On peut déplorer que l'interprétation géométrique des symétries et projections, et de leurs sous-espaces propres, soit négligée au profit d'une simple propriété opérationnelle sur $u\circ u$.
- Pour montrer qu'une famille est libre, penser aux lignes (ou colonnes) échelonnées pour les matrices, ou aux degrés échelonnés pour les polynômes.

Intégration

- Dans l'étude de la convergence d'intégrales généralisées, ont été observés fréquemment des oublis d'hypothèses de base (continuité de la fonction, convergence de chaque intégrale dans le cas d'une décomposition en somme d'intégrales, etc.).
- Les études de fonctions définies par des intégrales sont souvent maltraitées : les candidats mélangent ou confondent le théorème fondamental du calcul intégral et les théorèmes sur les intégrales dépendant d'un paramètre.
- La confusion entre *primitive* et *intégrale* a été trop souvent observée.

Géométrie

- De nombreux sujets de géométrie sont posés. C'est une particularité de la filière PT. Il est plus que conseillé de faire un dessin ; cela permet de mieux comprendre le sujet, et est très apprécié par les examinateurs.
- Le jury est surpris que certaines notions et résultats comme l'aire d'un trapèze, le cercle circonscrit à un triangle, ne soient pas connus de certains candidats.

- Les sujets de géométrie utilisent fréquemment la trigonométrie ; il convient donc de pouvoir donner rapidement les formules utiles à l'exercice, et aussi d'être capable d'étudier des fonctions trigonométriques simples, qui paramètrent souvent les courbes.
- Il faut surtout que les candidats, au lieu de se précipiter sur les calculs, mettent en place une démarche de résolution et annoncent à l'examinateur la liste des tâches pour arriver à la solution du problème posé.

Fonctions de plusieurs variables et géométrie des courbes et surfaces

Liées aux notions de champs et de courbes et surfaces, les fonctions de plusieurs variables, indispensables notamment en ingénierie mécanique, mériteraient d'être mieux maîtrisées. En particulier, il est indispensable de :

- connaître la définition des dérivées partielles d'une fonction à plusieurs variables et de savoir les calculer ;
- savoir utiliser *la règle de la chaîne* ;
- savoir déterminer la tangente et la normale à une courbe ainsi que le plan tangent à une surface, à partir d'équations cartésienne et paramétrique.

Équations différentielles linéaires et suites à récurrence linéaire forte

- Puisque dans ces deux situations, les candidats ont appris à rechercher les racines du polynôme discriminant, les interrogateurs ont bien trop souvent observé une confusion finale troublante entre $\exp(\lambda\, t)$ et λ^n.

Probabilités

- Les candidats sont en général bien préparés, avec une amélioration sensible par rapport à 2015.
- On apprécie qu'un candidat justifie naturellement un résultat obtenu (probabilités totales, conditionnelles, etc) et donne des définitions correctes, notamment celle de l'indépendance de deux événements, ou de deux variables aléatoires.
- On a cependant pu relever parfois des confusions entre : un évènement et sa probabilité ; évènements incompatibles et évènements indépendants ; probabilité de « *A et B* » et probabilité de « *A sachant B* ».

Informatique

Généralités

- Les candidats ont en général été bien préparés pour cet oral, et une amélioration sensible a été observée par rapport à 2015. On peut se réjouir qu'en 2016, les candidats qui confondent `return` et `print`, ou ne font pas la distinction entre `if`, `for` et `while` sont maintenant de rares exceptions.
- Quelques rares candidats ne connaissaient pas IDLE, alors qu'il est indiqué clairement et publiquement que c'est l'environnement de développement intégré (IDE) utilisé pour cet oral.
- Si quelques lignes de code sont proposées à la compréhension, il est conseillé au candidat de taper ce code et de le comprendre en modifiant certains paramètres.
- Ne pas négliger les premières questions : elles contiennent le plus souvent des éléments de réponse pour la suite, voire des rappels.
- Ne pas hésiter à utiliser l'interpréteur pour effectuer des vérifications élémentaires et savoir utiliser les instructions `help` et `numpy.info`.
- Il faut savoir mettre en œuvre une démarche en cas d'erreur : insérer des `print` pour contrôler pas à pas une exécution, etc. Il s'agit d'une compétence valorisée par le jury.
- Préférer une boucle `for` à un `while` quand le nombre d'itérations est connu à l'avance.

- De trop nombreux candidats ne font pas une distinction claire entre les entiers (type **`int`**) et les nombres à virgule flottante (type **`float`**), et ne maîtrisent par conséquent pas les conséquences induites. On a pu également déplorer parfois un manque d'aisance dans la manipulation des entiers (opérateurs **`//`** et **`%`**) ou celle des complexes, avec une méconnaissance de la notation **`1j`** et des écritures **`z.real`**, **`z.imag`** et **`z.conjugate()`**.
- Il est conseillé également de bien connaître les propriétés des listes d'une part, et des tableaux **`ndarray`** du module **`numpy`** d'autre part, en particulier ce qui les différencie, de façon à pourvoir choisir le type le mieux adapté au problème à résoudre.
- La manipulation des chaînes de caractères fait aussi partie des capacités exigibles.

Fonctions

- Il n'est pas nécessaire de définir systématiquement une fonction pour chaque tâche demandée.
- En revanche, une fonction doit toujours être testée, soit dans le programme, soit dans la console, même si ce n'est pas spécifié dans l'énoncé.
- Attention à ne pas donner le nom de la fonction à un objet créé dans cette fonction.
- L'appel d'une fonction sans argument nécessite des parenthèses vides.
- L'utilisation de variables globales n'est pas conseillée, et encore moins exigée.

Gestion du temps

- Quelques candidats perdent un temps considérable avec des pratiques peu adaptées pour une épreuve orale en 30' :
    - De trop nombreux « **`for i in range(len(iterable)) :`** » au lieu de simplement « **`for e in iterable`** » font perdre du temps et de la lisibilité.
    - Pour savoir si un objet **`e`** est dans un iterable **`I`**, « **`test = e in I`** » est quand même nettement plus rapide que l'écriture suivante utilisée par certains candidats et qui ne tient absolument pas compte des facilités du langage :

```
test = False
for i in range(len(I)) :
    if I[i] == e :
        test = True
```

    - L'opérateur booléen **`not`** est très souvent méconnu alors qu'il peut s'avérer très utile, comme par exemple dans : **`if e not in L : L.append(e)`**.
    - Dans le même ordre d'idée, « **`return booleen`** » est nettement plus efficace que l'écriture suivante que l'on a pu observer chez certains candidats et qui montre des lacunes évidentes sur la compréhension de la nature d'un booléen :

```
if booleen == True :
    return True
else :
    return False
```

    - On préférera également « **`S = L[d:f:p]`** » plutôt que la reconstruction « *manuelle* », qui en plus ne fonctionne que sur les listes et n'est pas aussi générale :

```
S = []
for i in range(d,f,p) :
    S.append(L[i])
```

- Plus généralement, on ne mettra en œuvre « *à la main* » un algorithme élémentaire que s'il est explicitement demandé. Sinon, on doit connaître et savoir utiliser les fonctions intrinsèques `min`, `max`, `sum`, `sorted`, et/ou la méthode **`sort`** pour les listes, les méthodes **`min`**, **`max`**, **`sum`**, **`mean`**, **`std`**, **`conjugate`**… pour les tableaux **`ndarray`** (ainsi que **`T.real`** et **`T.imag`** pour un tableau de complexes).
- Même si les *listes en compréhension* ne sont pas exigibles, leur utilisation maîtrisée permet de gagner en efficacité et en lisibilité.
- Ne pas hésiter non plus à réutiliser les fonctions créées dans les questions précédentes, ou même à créer si cela peut être utile de petites fonctions intermédiaires ; les exercices sont très souvent structurés dans cet esprit.
- Dans l'écriture de fonctions, l'utilisation systématique de `assert` ou l'écriture systématique d'en-têtes ("*docstring*") ne sont pas exigibles ; même si elles peuvent être préconisées en génie logiciel, ces pratiques font perdre un temps précieux.
- Il n'y a aucun « *dogme* » à respecter ; seule importe l'efficacité pour résoudre un problème.

Algorithmes itératifs et récursifs

- La distinction entre *algorithme récursif* et *algorithme itératif* ne semble pas claire pour une majorité de candidats.
- Dans l'écriture d'une fonction récursive, la condition d'arrêt est trop souvent mal formulée (par exemple, `while` à la place de `if`), sinon oubliée.

Extraction de données utilisables à partir d'un fichier texte

- Les candidats, dans leur grande majorité, savaient extraire les lignes d'un fichier ASCII sous forme de chaînes de caractères, mais beaucoup trop peu ont su transformer ces lignes en données numériques utilisables.
- Il est conseillé de bien connaître les méthodes **`split`** et **`strip`** des chaînes de caractères qui s'avèrent souvent utiles pour la conversion en données utilisables.
- Il est dommage que la notion de chemin, relatif et absolu, pour accéder à un fichier soit en général mal maîtrisée, voire méconnue par certains candidats.

Résolution numérique d'un problème différentiel

Rappel du programme, chapitre « *simulation numérique* » : « *Problème dynamique à une dimension, linéaire ou non, conduisant à la résolution approchée d'une équation différentielle ordinaire par la méthode d'Euler. On compare les résultats obtenus avec les fonctions de résolution approchée fournies par une bibliothèque numérique. On met en évidence l'impact du pas de discrétisation et du nombre d'itérations sur la qualité des résultats et sur le temps de calcul.* »

- Qu'il s'agisse de la mise en œuvre de la méthode d'Euler ou de l'utilisation de la fonction **`odeint`** du module **`scipy.integrate`** de *Python* (ou **`ode`** de *Scilab*), il est nécessaire de connaître la forme « $\mathbf{V}'(t) = \mathbf{F}(\mathbf{V}(t),t)$ » avec la condition initiale « $\mathbf{V}(t_0) = \mathbf{V_0}$ », où **V** est la fonction cherchée, et **F** une fonction en général non linéaire.

Analyse des résultats

1469 candidats présents, répartis en 9 jurys sur 11 jours du 27 juin au 8 juillet 2016, ont passé cet oral.

Même si les notes publiées des candidats sont globales, nous donnons les résultats, exercice par exercice, à des fins statistiques :

	Général	*Mathématiques*	*Algorithmique*
Moyenne (sur 20)	**10,82**	*10,48*	*11,16*
Écart-type	**3,82**	*4,88*	*4,86*
Note minimale	**1**	*0*	*0*
Note maximale	**20**	*20*	*20*

La répartition des notes est la suivante :

		Mathématiques					
Algorithmique		[0,4[	[4,8[	[8,12[	[12,16[	[16,20]	*Total Algo*
	[0,4[	1.1%	1.8%	1.1%	0.9%	0.5%	*5.4%*
	[4,8[	1.4%	5.1%	4.6%	5.0%	2.2%	*18.5%*
	[8,12[	1.5%	5.3%	6.7%	6.6%	3.2%	*23.4%*
	[12,16[	1.9%	6.5%	7.0%	8.6%	5.2%	*29.2%*
	[16,20]	0.7%	4.1%	4.8%	6.0%	8.0%	*23.6%*
	Total Math	*6.7%*	*22.8%*	*24.3%*	*27.1%*	*19.2%*	[100.0%]

Exercices d'algorithmique et de simulation numérique fournis en annexe

Rappel : ***nous rappelons que les exercices peuvent être de longueurs variables. L'objectif poursuivi est que l'interrogateur puisse évaluer les capacités de chaque candidat grâce à l'exercice proposé et non pas que le candidat termine l'exercice.***

Avertissement : ***les exercices fournis ici ne sont pas représentatifs de l'ensemble des exercices pouvant être posés. S'il y a ici sensiblement plus d'exercices à tonalité algorithmique et mathématique, c'est que ces exercices ont été plus facilement « divulgués » que ceux de simulation numérique comportant notamment des lectures de fichiers, du traitement de données et des tracés. En réalité, autant d'exercices d'algorithmique que de simulation numérique ont été posés en 2016. Nombre d'entre eux ne comportaient aucune question mathématique.***

Exercice d'algorithmique

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

La suite $(K_n)_{n\in\mathbb{N}}$ d'entiers naturels est définie par :

- $K_0 = 1$
- $\forall n \in \mathbb{N}^\star,\ K_n = \sum_{i=0}^{n-1} K_i\, K_{n-1-i}$

1. Écrire une fonction récursive `K` d'argument un entier n et renvoyant K_n.
2. Calculer `K(2)`, `K(5)`, `K(10)` et `K(15)`. Que pensez-vous de l'efficacité de la fonction `K` ?
3. Écrire une fonction `LK` non récursive d'argument un entier n, qui renvoie la liste des K_i, pour i compris entre 0 et n.
4. Écrire une fonction `L` d'argument un entier n et renvoyant le nombre L_n donné par

$$L_n = \frac{1}{n+1}\binom{2\,n}{n}$$

   On pourra utiliser `comb` *qui se trouve dans la sous-bibliothèque* `scipy.misc` *du langage Python.*
5. Afficher en les comparant les 10 premières valeurs de K_n et de L_n. Que peut-on conjecturer ?
6. On admet que la conjecture précédente est vérifiée pour tout entier n. Après avoir exprimé L_n en fonction de L_{n-1}, proposer une méthode permettant un calcul plus efficace de K_n.

Exercice d'algorithmique

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

---

1. Écrire une fonction `binaire` d'argument un entier naturel n et qui renvoie la liste des chiffres, bit de poids fort en tête, de l'écriture en base 2 de n. Par exemple, `binaire(23)` renvoie `[1,0,1,1,1]`.
2. Écrire une fonction `nombreDeUns` d'argument un entier naturel n, qui renvoie le nombre de chiffres 1 dans l'écriture binaire de n. Par exemple, `nombreDeUns(23)` renvoie 4.
3. Soit n un entier naturel. On dit que n est un 2-palindrome si sa représentation en base 2 est la même, qu'elle soit écrite de gauche à droite ou de droite à gauche. Par exemple, 9 est un 2-palidrome car 9 s'écrit `1001` en base 2.

   Écrire une fonction `palindrome` d'argument un entier naturel n qui renvoie un booléen indiquant si n est un 2-palindrome, ou pas.
4. Faire afficher tous les 2-palindromes inférieurs à 100.
5. Écrire une fonction `baseB` de deux arguments, un entier naturel n et un entier b compris entre 2 et 10, qui renvoie la liste des chiffres, chiffre de poids fort en tête, de l'écriture en base b de n.
6. Faire afficher les 10 plus petits entiers naturels non nuls qui sont à la fois des 4-palindromes et des 9-palindromes.

Exercice d'algorithmique

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

Une matrice carrée d'ordre n est dite « *magique* » si elle contient tous les nombres de 1 à n^2 et si les sommes des nombres de chaque ligne, de chaque colonne et de chaque diagonale sont toutes égales à une même constante s.

1. Au brouillon, exprimer la constante s en fonction de n.
2. Créer une fonction `EstMagique`, d'argument une matrice T (carrée de taille n) et qui renvoie un booléen indiquant si T est magique ou pas.
   Tester cette fonction sur les matrices :

$$A = \begin{pmatrix} 4 & 9 & 2 \\ 3 & 5 & 7 \\ 8 & 1 & 6 \end{pmatrix} \text{ et } B = \begin{pmatrix} 1 & 8 & 2 \\ 4 & 5 & 7 \\ 6 & 9 & 3 \end{pmatrix}$$

Une méthode permettant de construire une matrice magique de taille impaire $n = 2p+1$ est la suivante :

- On construit une matrice de taille n remplie de zéros. On considère cette matrice comme la représentation sur une période d'une matrice infinie n-périodique en lignes et en colonnes.
- On remplace ensuite les zéros de la matrice avec les nombres de 1 à n^2 comme suit :
  — on met le 1 dans la case située sous la case centrale de la matrice ;
  — on place ensuite chaque nombre de 2 à n^2 dans la case située ligne suivante et colonne suivante de celles où on a mis le nombre précédent. Si cette case est déjà remplie, on avance encore d'une ligne et on recule d'une colonne.

On admet que la matrice ainsi construite est magique.

3. Construire « *à la main* », selon cette méthode, une matrice magique d'ordre 3.
4. Écrire une fonction `Magique` d'argument un entier p qui renvoie la matrice magique de taille $2p+1$ créée à l'aide de la méthode précédente.

Exercice d'algorithmique

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

On travaille avec des triplets (jour, mois, année) où jour et année sont des nombres entiers (avec la condition année $\geqslant 1582$: date de mise en place du calendrier grégorien) et mois est une chaîne de caractères.

1. Créer une liste notée `MC` contenant les mois de l'année qui ont 30 jours et une liste notée `ML` contenant les mois de l'année qui ont 31 jours.
2. On rappelle qu'une année est bissextile lorsqu'elle est divisible par 4 mais pas par 100, ou bien lorsqu'elle est divisible par 400. Créer une fonction `estBissextile` d'un argument `an` qui renvoie `True` si l'année `an` est bissextile et `False` sinon.

   Créer une fonction `longueurmois` de deux arguments `ms` et `an` qui renvoie le nombre de jours du mois `ms` de l'année `an`.
3. Créer une fonction `valide` de trois arguments `jr`, `ms`, et `an` qui renvoie `True` si le triplet `(jr,ms,an)` est valide et `False` sinon. Par exemple `valide(25,'janvier',1896)` devra renvoyer `True` alors que `valide(30,'février',1972)` devra renvoyer `False`.
4. Écrire une fonction `nab` de deux arguments `date1` et `date2` qui renvoie le nombre de « *29 février* » entre ces deux dates. `date1` et `date2` sont deux triplets sous la forme `(jr,ms,an)`.
5. Écrire une fonction `jours` de deux arguments `date1` et `date2` qui renvoie le nombre de jours séparant les deux dates.

Exercice d'algorithmique

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

---

1. On considère un nombre n. Que donne la commande `list(str(n))` ?
2. Écrire une fonction nommée `somme` d'argument un nombre entier naturel et qui renvoie la somme de ses chiffres.
3. On considère qu'un nombre est « *adéquat* » si la somme de ses chiffres est un multiple de 10. Écrire une fonction nommée `adequat` d'argument un entier naturel n et qui renvoie un booléen indiquant si n est « *adéquat* », ou pas.
4. Écrire une fonction `modification` d'argument un entier naturel n qui renvoie un nombre p ayant les mêmes chiffres des dizaines, centaines, etc, que n et avec un chiffre des unités tel que p soit *adéquat*. Si n est déjà *adéquat*, on aura $n = p$.
5. Tester cette fonction en demandant d'afficher `adequat(modification(n))` pour 10 valeurs aléatoires de n entre 10 000 et 100 000 (fonction `randint` du module `random`).
6. Tirer au hasard plusieurs milliers d'entiers entre 1 et 100 000 et calculer la proportion de nombres *adéquats*. Ce résultat vous surprend-il ?

Exercice d'algorithmique

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

Dans cet exercice, on manipule des suites (finies) d'entiers sous forme de listes d'entiers. Ainsi la suite $(0, 1, 2, 3)$ sera représentée par la liste `[0,1,2,3]`.

La liste est croissante (respectivement décroissante, monotone) si la suite est croissante (respectivement décroissante, monotone).

1. Écrire une fonction `estCroissante` d'argument une liste d'entiers et qui renvoie un booléen indiquant si cette liste est croissante ou pas. Cette fonction devra être de complexité linéaire dans le pire des cas.
2. Écrire de même des fonctions `estDecroissante` et `estMonotone` qui testent si une liste d'entiers est respectivement décroissante, monotone.

Soit la liste $\mathtt{L} = [\mathtt{u_0}, \mathtt{u_1}, \cdots, \mathtt{u_{n-1}}]$ de longueur n. On appelle tranche de `L` une liste de la forme $[\mathtt{u_i}, \mathtt{u_{i+1}}, \cdots, \mathtt{u_j}]$ où $0 \leqslant i \leqslant j < n$.

On cherche une tranche de `L` croissante et de longueur maximale.

Par exemple, une tranche croissante de longueur maximale de`[0,1,0,1,2,3,4,0,1,2]` est `[0,1,2,3,4]`, correspondant aux indices $i = 2$ à $j = 6$.

3. Écrire une fonction `LC` de deux arguments, une liste `L` et un entier p, qui renvoie l'entier d tel que : la liste $[\mathtt{u_p}, \cdots, \mathtt{u_{p+d}}]$ est croissante et soit $p+d = n-1$, soit $\mathtt{u_{p+d}} > \mathtt{u_{p+d+1}}$.
4. Écrire une fonction `maxCroissante` d'argument une liste `L` qui renvoie la plus longue tranche croissante de `L`. S'il n'y a pas unicité, on renvoie la première trouvée.

Exercice d'algorithmique

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

Soit n un entier naturel impair et non multiple de 5. On admet qu'un de ses multiples noté N s'écrit en base 10 avec seulement le chiffre 1 (de la forme $111\cdots111$). Le but de cet exercice est de trouver pour un n vérifiant ces conditions le plus petit multiple N de cette forme, et de déterminer le nombre de 1 nécessaires.

1. Écrire une fonction `QueDesUn` d'un argument n qui renvoie le premier multiple N de n ne s'écrivant qu'avec des 1, si n est impair non multiple de 5, et `'erreur'` sinon.
2. Pour n allant de 1 à 150, pour les n impairs et non multiples de 5 afficher les couples (n, N).
3. Écrire une fonction Longueur(k), donnant le nombre de chiffres de l'entier naturel k. [Indication : la fonction log ne pouvant pas s'appliquer aux entiers longs, on pourra compter le nombre de divisions entières par 10 que l'on peut faire pour obtenir finalement 0.]
4. Pour n allant de 1 à 1000, chercher le(s) couple(s) (n, N) tel que le nombre de 1 de N soit le plus grand. (En cas d'égalité, donner toutes les solutions.)

Exercice d'algorithmique

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

Soit la suite $(u_n)_{n\in\mathbb{N}}$ définie par

$$u_0 = N \text{ (entier naturel non nul)} \quad \text{et} \quad \forall n \in \mathbb{N}\ ,\ u_{n+1} = \begin{cases} u_n \ / \ 2 \text{ si } u_n \text{ est pair} \\ 3\,u_n + 1 \text{ si } u_n \text{ est impair} \end{cases}$$

1. À l'aide notamment des instructions `plot` et `show` de la bibliothèque `matplotlib.pyplot`, représenter graphiquement les 50 premiers termes de la suite (u_n) avec $N = 7$, puis $N = 18$.
2. On conjecture que pour tout entier naturel N non nul, il existe un plus petit entier n tel que $u_n = 1$. Que se passe-t-il si c'est le cas ?

   Si cet entier existe, on l'appelle « *durée de vol* » de la suite (u_n) de valeur initiale N.
3. Écrire une fonction `vol` d'argument un entier N, renvoyant la liste de tous les termes de la suite (u_n) de valeur initiale N jusqu'à la première apparition de la valeur 1, ainsi que la durée de vol et la valeur maximale de la suite.
4. Représenter la durée de vol en fonction du point de départ N, pour les valeurs de N inférieures à 1000.
5. Représenter la distribution des durées de vol à l'aide d'un histogramme (on pourra utiliser la commande `hist` de `matplotlib.pyplot`).

Exercice d'algorithmique

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

Pour un entier naturel $n \geqslant 2$, on appelle **diviseurs propres de n** les entiers naturels strictement inférieurs à n qui divisent n.

Par exemple la liste des diviseurs propres de 100 est `[1,2,4,5,10,20,25,50]`.

On va s'intéresser à la somme de ces diviseurs propres. Pour 100, elle vaut par exemple 117.

1. Écrire une fonction `LDP` d'argument un entier naturel n qui renvoie la liste de ses diviseurs propres. La tester pour $n = 100$.
2. Écrire une fonction `SDP` d'argument un entier naturel n qui renvoie la somme de ses diviseurs propres. La tester pour $n = 100$.
3. On dit qu'un entier naturel est **parfait** s'il est égal à la somme de ses diviseurs propres. Écrire une fonction `parfaits` d'argument un entier naturel K qui renvoie la liste des entiers p parfaits inférieurs ou égaux à K, après avoir affiché au fur et à mesure le message « p *est parfait* ». La tester pour $K = 2000$.
4. On dit que deux entiers sont **amicaux** si chacun est égal à la somme des diviseurs propres de l'autre. Écrire une fonction `amicaux` d'argument un entier naturel K qui renvoie la liste de tous les couples (p, q) d'entiers amicaux tels que $p < q \leqslant K$, après avoir affiché les couples trouvés au fur et à mesure. Tester `amicaux` pour $K = 5000$.

Exercice d'algorithmique

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

Dans la liste des entiers naturels non nuls, on barre un nombre sur 2 en commençant par barrer le deuxième :

$$1\ ,\ \cancel{2}\ ,\ 3\ ,\ \cancel{4}\ ,\ 5\ ,\ \cancel{6}\ ,\ 7\ ,\ \cancel{8}\ ,\ 9\ ,\ \cancel{10}\ ,\ 11\ ,\ \cancel{12}\ ,\ 13\ ,\ \cdots$$

Puis dans la liste restante, on barre un nombre sur 3 en commençant par barrer le troisième :

$$1\ ,\ 3\ ,\ \cancel{5}\ ,\ 7\ ,\ 9\ ,\ \cancel{11}\ ,\ 13\ ,\ \cdots$$

puis on barre un nombre sur 4, un nombre sur 5, etc.

Et ceci à l'infini pour obtenir « *la liste des nombres de type J* ».

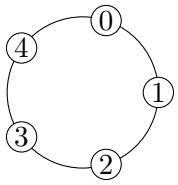
1. Écrire une fonction `enlever` de deux arguments, une liste L et un entier naturel i, qui renvoie une liste S construite en ne gardant dans la liste L que les éléments dont le rang n'est pas multiple de i. Par exemple, `enlever([1,3,5,7,9,11,13],3)` doit donner `[1,3,7,9,13]`.
2. Écrire une suite d'instructions donnant la liste des nombres de type J inférieurs ou égaux à 100.
3. Écrire une fonction `LJ` d'argument n renvoyant la liste des nombres de type J inférieurs ou égaux à n.
4. Écrire une fonction `U` d'argument n renvoyant u_n, le nombre de nombres de type J inférieurs ou égaux à n.
5. Vers quelle limite ℓ semble tendre $4\,n/u_n^2$ quand n tend vers l'infini ?
6. Déterminer le premier n pour lequel la différence en valeur absolue entre $4\,n/u_n^2$ et ℓ est inférieure à 10^{-3}.

Exercice d'algorithmique

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

---

n personnes numérotées de 0 à $(n-1)$ se mettent en cercle, comme le montre la figure suivante pour $n = 5$:



En commençant par la personne numéro 1 et en tournant dans le sens des numéros croissants (sens horaire sur la figure), on retire une personne sur deux, en ne prenant en compte que les personnes restant dans le cercle. Par exemple, pour $n = 5$, on retirera successivement les personnes 1, 3, 0 puis 4 ; il restera la personne 2.

Pour simuler cette procédure d'élimination progressive, on décrit le cercle par une liste de n booléens : la valeur numéro i est `True` si la personne i est éliminée, et `False` si elle est encore dans le cercle.

Au départ, la liste ne contient que des `False` puis ses valeurs passent progressivement à `True`, jusqu'à ce qu'il ne reste plus qu'un seul `False`.

Ainsi, pour $n = 5$, la liste passe par les états successifs :

Liste de booléens `E`	Rang `p` du dernier éliminé
`[ False , True , False , False , False ]`	1
`[ False , True , False , True , False ]`	3
`[ True , True , False , True , False ]`	0
`[ True , True , False , True , True ]`	4

Il reste 2

1. Écrire une fonction `suivant` de deux arguments, une liste `E` de n booléens et un entier p entre 0 et $(n-1)$ qui renvoie la position q du premier `False` rencontré en partant de la position juste après la position p, en parcourant la liste de façon circulaire.

   Par exemple :

   `suivant([True,True,False,True,False],0)` donne 2 ;
   `suivant([True,True,False,True,False],2)` donne 4 ;
   `suivant([True,True,False,True,False],4)` donne 2.

2. Se servir de la fonction `suivant` pour simuler le cas $n = 5$ décrit dans le tableau ci-dessus.
3. Écrire une fonction `reste` d'argument un entier naturel non nul n qui renvoie le numéro de la dernière personne restante parmi n personnes. Tester `reste` pour $n = 16$, $n = 31$, $n = 65$.
4. Pour $2 \leqslant n \leqslant 140$, afficher n suivi du numéro du dernier restant.

   En observant le résultat, que peut-on conjecturer sur le calcul du dernier restant ?
5. En supposant que la conjecture est exacte, deviner sans utiliser le programme Python quel sera le numéro du dernier restant pour les valeurs de n : 256, 513, 1023, 1041.
6. Écrire une fonction `reste2` mettant en œuvre cette conjecture et comparer les résultats pour $n = 5104$.

Exercice de simulation numérique

Cet exercice pourra être fait avec le langage Python (et ses bibliothèques numpy, scipy, matplotlib) ou avec le logiciel Scilab. À chaque question, les instructions ou les fonctions écrites devront être testées.

On définit les suites $(a_n)_{n\in\mathbb{N}}$ et $(b_n)_{n\in\mathbb{N}}$ par :

$$\left\{\begin{array}{rcl} a_0 &=& u \\ b_0 &=& v \end{array}\right. \quad \text{et} \quad \forall n \in \mathbb{N}\,; \left\{\begin{array}{rcl} a_{n+1} &=& \sqrt{a_n\, b_n} \\ b_{n+1} &=& \dfrac{2}{1/a_n \;+\; 1/b_n} \end{array}\right. ,$$

où u et v sont deux réels strictement positifs donnés.

1. Créer une fonction `iterer` d'un seul argument p, où p représente le couple (a_n, b_n), et qui renvoie le couple (a_{n+1}, b_{n+1}). Tester cette fonction en calculant `iterer([3,2])`.
2. Créer une fonction *non récursive* `suite` de trois arguments u, v et n qui renvoie le couple (a_n, b_n).
3. Tester cette fonction en calculant `suite(3,2,2)` puis `suite(3,2,5)` ; que peut-on conjecturer ?
4. On admet que pour deux réels strictement positifs donnés u et v, les suites $(a_n)_{n\in\mathbb{N}}$ et $(b_n)_{n\in\mathbb{N}}$ convergent vers une même limite L qu'elles encadrent (*i.e.* pour $n \geqslant 1$, $a_n \geqslant L \geqslant b_n$). Créer une fonction `moyenne` de deux arguments u et v qui renvoie une valeur approchée à 10^{-10} près par défaut de cette limite L.
5. Créer une fonction *récursive* `suiterec` de trois arguments u, v et n qui renvoie le couple (a_n, b_n). Tester cette fonction par `suiterec(3,2,5)` ;
6. Faire tracer `moyenne(1,x)` pour `x` variant de 1 à 10 avec un pas de 0.1.

Exercice de simulation numérique

Cet exercice pourra être fait avec le langage Python (et ses bibliothèques numpy, scipy, matplotlib) ou avec le logiciel Scilab. À chaque question, les instructions ou les fonctions écrites devront être testées.

---

1. Soit f la fonction définie sur $[-5,5]$ par $f(x)=\dfrac{1}{1+x^2}$. Tracer la courbe de f.

> Il existe différentes méthodes pour approcher une fonction h par un polynôme.
>
> L'une d'entre elles, dite « *interpolation de Lagrange* », consiste à construire un polynôme de degré $(n-1)$ coïncidant en n points avec la fonction h.
>
> Plus précisément, si h est définie sur un segment contenant les valeurs distinctes $[a_0, a_1, \cdots, a_{n-1}]$ regroupées dans une liste $\mathbf{a}$, alors le polynôme :
>
> $$P_{\mathbf{a},h}(x)=\sum_{i=0}^{n-1} h(a_i)\ \left(\prod_{k=0,k\neq i}^{n-1}\frac{x-a_k}{a_i-a_k}\right)$$
>
> est le seul polynôme de degré inférieur ou égal à $(n-1)$ tel que
>
> $$\forall i\in\mathbb{N}\,,\ 0\leqslant i<n\,,\quad P(a_i)=h(a_i).$$

2. Écrire une fonction `interpoler` de trois arguments, une fonction h, un réel x et une liste $\mathbf{a}$ qui renvoie $P_{\mathbf{a},h}(x)$.
3. Écrire une fonction `affiche` d'argument j et donnant la représentation graphique de f et de $P_{\mathbf{a},f}$, $\mathbf{a}$ correspondant aux $(j+1)$ valeurs équiréparties sur l'intervalle $[-5,5]$.
4. Tester la fonction précédente pour $j=5,10,15,20$. Qu'observe-t-on ?
5. Cette fonction particulière f montre que l'interpolation par des points équirépartis n'est pas toujours adaptée pour obtenir une bonne approximation. Une façon de traiter ce problème consiste à ne plus faire l'interpolation de Lagrange sur des points équirépartis sur le segment, mais sur les abscisses dites « *de Tchebychev* » :
$$a_i=5\cos\left(\frac{(2\,i+1)\,\pi}{2\,n}\right)\,,\quad i\in\mathbb{N}\,,\ 0\leqslant i<n.$$
Visualiser le résultat pour différentes valeurs de n.

Exercice de simulation numérique

Cet exercice pourra être fait avec le langage Python (et ses bibliothèques numpy, scipy, matplotlib) ou avec le logiciel Scilab. À chaque question, les instructions ou les fonctions écrites devront être testées.

L'objet de cette exercice est d'étudier numériquement la série $\sum \frac{(-1)^{n+1}}{n}$.

1. On admet que cette série est convergente et que $\left|\sum_{k=n}^{+\infty} \frac{(-1)^{k+1}}{k}\right| < \frac{1}{n}$.

   Donner une valeur approchée à 10^{-6} près de la somme de cette série. Vérifier que cette valeur est proche de `log(2)` où `log` désigne le logarithme népérien.
2. Représenter graphiquement les cent premières sommes partielles de cette série.
3. On modifie l'ordre des termes de la série en prenant alternativement 2 termes positifs et 3 termes négatifs comme suit :

   $$\left(1+\frac{1}{3}\right)+\left(-\frac{1}{2}-\frac{1}{4}-\frac{1}{6}\right)+\left(\frac{1}{5}+\frac{1}{7}\right)+\left(-\frac{1}{8}-\frac{1}{10}-\frac{1}{12}\right)+\cdots$$

   Écrire une fonction `SPD` d'argument un entier naturel n non nul et qui renvoie la liste des n premières sommes partielles de cette série réordonnée.
4. Représenter graphiquement les 120 premières valeurs de cette série réordonnée. Que peut-on conjecturer sur la somme de cette série ?
5. Examiner ce qui se passe si l'on prend un terme positif, puis un négatif, puis deux positifs, puis un seul négatif, puis trois positifs, puis un seul négatif, puis quatre positifs, etc.

Exercice de simulation numérique

Cet exercice pourra être fait avec le langage Python (et ses bibliothèques numpy, scipy, matplotlib) ou avec le logiciel Scilab. À chaque question, les instructions ou les fonctions écrites devront être testées.

---

1. Soit le système différentiel avec condition initiale suivant :

$$\begin{cases} x'(t) &= y(t) \\ y'(t) &= \left(1 - x(t)^2\right) y(t) - x(t) \\ x(0) &= 2 \\ y(0) &= 0 \end{cases} \tag{1}$$

   Pour un pas de discrétisation $h = \frac{1}{4}$, puis $h = \frac{1}{8}$, représenter, en fonctions du temps t, les solutions approchées de cette équation obtenues par la méthode d'Euler, pour t dans l'intervalle $[0, 8]$.

2. En utilisant la fonction `ode` de *Scilab* ou la fonction `odeint` du module `scipy.integrate` de *Python*, résoudre numériquement le problème (1). On prendra garde à définir soigneusement les arguments de cette fonction en lisant attentivement l'aide en ligne.

   Représenter sur la figure précédente la solution numérique trouvée.

3. Représenter les différentes solutions trouvées dans le plan xy.

4. Représenter sur l'intervalle $[0, 8]$ une solution approchée de l'équation : $x''(t) = (1 - x(t)^2)\, x'(t) - x(t)$, avec $x(0) = 0$ et $x'(0) = 1$.