Épreuve orale de « *Mathématiques et algorithmique* » de la Banque PT – Rapport 2021

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 consulter le **site de la Banque PT**, où ils pourront trouver le mémento *Python* fourni lors de l'oral, les exercices types d'informatique, ainsi que les rapports des années antérieures comportant à la fois des informations complémentaires en regard du présent rapport et des exercices qui ont été posés lors de sessions antérieures, à titre d'exemples.

1 – 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), sans oublier celle des connaissances de base du programme des classes du lycée (seconde, première, terminale).

Cette épreuve permet aussi d'examiner :

- l'aptitude du candidat à lire attentivement un sujet et à répondre précisément à la question posée ;
- son aisance à exposer clairement ses idées avec un vocabulaire précis ;
- sa capacité d'initiative et son autonomie et, en même temps, son aptitude à écouter l'interrogateur, à prendre en compte ses indications, à lui demander des précisions si besoin ;
- 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.

2 – Modalités de cette épreuve

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

Il comporte deux exercices de durées comparables :

- l'un porte sur le programme de mathématiques des deux années de la filière PTSI/PT (algèbre, analyse, géométrie et probabilités) et se déroule au tableau ;
- l'autre exercice porte sur les items 2, 3 et 5 du programme d'informatique et se déroule sur ordinateur. Pour ce deuxième exercice, les candidats disposent d'un ordinateur (Windows 10, clavier français Azerty) dans lequel sont installés *Python* 3.6 et ses principales bibliothèques (dont `numpy`, `scipy`, `matplotlib`, `random`, aides incluses) [1], d'un mémento plastifié en couleurs au format A3, et de feuilles de brouillon, qu'il ne faut pas hésiter à utiliser. **L'environnement de développement** est *IDLE*, comme annoncé depuis 2014, muni de l'extension ***IDLEX*** qui permet notamment d'afficher plus clairement les numéros de ligne, de faire exécuter une partie d'un programme seulement (F9 au lieu de F5), ou de rappeler dans la console une commande déjà saisie (flèches montante et descendante). Quelques candidats ont avoué avoir préparé l'oral avec *Spyder*, *Pyzo* ou autre, ce qui est un peu surprenant. Nous ne pouvons que conseiller de se placer dans les conditions de passage de l'oral tout au long des deux années de préparation.

1. Distribution Anaconda (voir par exemple **Formations Python 3 Arts et Métiers**).

Pendant chaque exercice, alternent des phases de réflexion et d'écriture du candidat et des phases d'interaction avec l'interrogateur, par le biais éventuel d'une feuille de brouillon pour l'exercice sur ordinateur si cela facilite les échanges.

3 – Organisation

Cette dernière session s'est déroulée dans des conditions particulières en raison de la pandémie, dans le respect des consignes sanitaires. 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}). Au final, les conditions de passage étaient très voisines de celles des oraux jusqu'en 2019.

4 – À propos de l'oral 2021

Comme lors des sessions précédentes, la plupart des candidats semblaient bien préparés à cette épreuve. Cependant, l'équipe d'interrogateurs a pu observer cette année des lacunes très inhabituelles de trop nombreux candidats, en particulier en algèbre linéaire (rang d'une application linéaire, sous-espace vectoriel, définition des valeurs et vecteurs propres), sur la manipulation des nombres complexes, sur la mise en place d'une démonstration par récurrence et sur l'étude des courbes paramétrées. D'autres candidats semblaient également ne pas posséder de bases en algorithmique et programmation. La cause majeure en est sans doute l'accumulation des perturbations significatives engendrées par la pandémie des deux dernières années. Espérons que tout sera rentré dans l'ordre dès les prochaines sessions.

5 – Conseils généraux

Lors d'une épreuve orale, le candidat doit être extrêmement vigilant :

- Lire attentivement le sujet et bien écouter une question permet de répondre à la question effectivement posée ; lire une phrase dans son intégralité (du premier mot au point final) peut s'avérer extrêmement profitable ; trop souvent, c'est pour n'avoir pas vu un mot, un seul, que l'on passe à côté d'une question.
- Écouter les consignes de l'interrogateur est en général utile ; il vaut mieux attendre qu'il ait terminé avant de répondre ; de même, une consigne du style « *je vous laisse continuer* » signifie que la phase d'échanges est terminée et que le candidat doit poursuivre sa réflexion.
- Lorsqu'une indication est donnée pour aider le candidat, il faut savoir l'écouter et réagir à celle-ci, par exemple en la reformulant pour vérifier qu'on l'a bien comprise.
- La capacité du candidat à s'exprimer clairement avec un vocabulaire précis est évidemment un critère important d'évaluation.

Ces capacités d'attention, d'écoute et de réaction sont des éléments d'évaluation. De manière générale, la passivité, l'attentisme, le mutisme, ou l'obstination dans une voie infructueuse sont déconseillés lors de l'oral.

Les exercices posés sont tous issus de banques d'exercices sur lesquelles l'équipe d'interrogateurs travaille tout au long de l'année, notamment en faisant le bilan de chaque session d'oraux. Ces exercices sont de longueurs variables et assez souvent trop longs. Il est donc important de rappeler que l'objectif poursuivi est l'évaluation par l'interrogateur des capacités de chaque candidat grâce à l'exercice proposé, et non pas que le candidat termine nécessairement l'exercice.

L'oral, contrairement à une « *colle* », ne sert qu'à évaluer les capacités du candidat et non plus à participer à sa formation ; des indications seront en général données par l'interrogateur si le candidat reste bloqué trop longtemps, ou si celui-ci demande de l'aide par des questions dont il reconnaît implicitement ignorer la réponse (exemples : « *Est-ce que je peux utiliser tel théorème ?* », ou « *Pourquoi la figure ne s'affiche-t-elle pas ?* »).

Il est évidemment préférable, lorsqu'on sollicite de l'aide, d'expliquer les pistes envisagées et les raisons pour lesquelles elles ne semblent pas déboucher, plutôt que de se contenter de dire « *Je ne vois pas.* » ou « *Ça ne marche pas.* ».

Contrairement à une « *colle* », le candidat ne doit pas s'attendre à ce qu'on lui donne la solution à la fin de l'épreuve ni que l'on émette de commentaires ; le respect strict des horaires, pour garantir l'égalité de traitement entre les candidats, peut entraîner l'arrêt d'un exercice d'une manière abrupte, ou que l'on demande à un candidat de se dépêcher, sans que cela puisse donner sujet à interprétation sur l'évaluation elle-même.

Quelques détails utiles en mathématiques comme en informatique :

- Une bonne maîtrise des nombres complexes, de leurs différentes représentations (tant mathématique qu'informatique) et de leur manipulation est requise ; leur utilisation et leur manipulation en tant qu'affixes de points du plan, permettant d'éviter de revenir systématiquement aux coordonnées, peut s'avérer très efficace (exemples : affixe du milieu de deux points, distance entre deux points) ; les interprétations géométriques du module, de l'argument, des parties réelles et imaginaires, du conjugué d'un nombre complexe doivent donc être connues.
- En géométrie dans le plan, on doit être capable de construire et/ou de manipuler les coordonnées de points et de vecteurs, de calculer la longueur d'un segment (en repère orthonormé), les coordonnées des sommets d'un polygone usuel – en vue par exemple de faire tracer les côtés de ce polygone à l'écran –, l'aire de polygones usuels (triangle, trapèze, carré, rectangle, parallélogramme) ; le rôle du déterminant de deux vecteurs $\overrightarrow{AB}$ et $\overrightarrow{BC}$ du plan (et aussi, dans l'espace, de leur produit vectoriel) est trop souvent méconnu pour caractériser l'alignement des 3 points, la colinéarité des vecteurs, l'aire du parallélogramme $ABDC$ et, conséquemment, celle du triangle ABC.

6 – Conseils pour l'exercice de mathématiques

6.1 – Généralités

- L'oral n'est pas un écrit sur tableau ; les justifications et commentaires doivent être donnés au moment où l'on est interrogé ; le temps étant limité, il est inutile d'écrire de longues phrases, notamment pour justifier une linéarité ou une continuité triviales, et encore moins lire à voix haute voire de recopier l'énoncé que l'interrogateur et le candidat connaissent tous les deux.
- Le candidat doit être précis dans ses propos, et, en particulier lorsqu'il énonce une définition, une propriété ou un théorème au programme de mathématiques, il doit énoncer l'ensemble des hypothèses sans en oublier ; le jury attend d'un candidat qu'il connaisse les résultats de cours.
- Un exercice de mathématiques ne peut se résumer à l'application d'une recette toute faite ; au lieu de se précipiter vers l'utilisation d'un théorème, d'une règle ou d'une technique, chaque candidat pourra se poser la question : « *L'application de la définition ou un calcul élémentaire ne suffisent-ils pas à fournir une solution ?* » (Exemples : $\phi(\mathbf{v}) = \lambda\,\mathbf{v}$ pour la recherche d'une valeur propre λ et d'un vecteur propre $\mathbf{v}$ pour l'application linéaire ϕ, calcul de la somme partielle pour étudier une série, calcul de l'intégrale dépendant d'un paramètre, dérivation de $x \mapsto \int_a^x f(t)\,\mathrm{d}t$).
- On attend également d'un candidat qu'il maîtrise les techniques de calcul en connaissant les concepts sous-jacents ; par exemple, maîtriser le procédé de calcul puis de recherche des racines du polynôme caractéristique ne dispense pas de connaître les définitions de valeur propre et de sous-espace propre ; lorsque plusieurs procédés de calcul sont possibles, par exemple pour la résolution d'un système linéaire ou la détermination du rang d'une matrice (méthode du pivot, substitution, combinaisons linéaires, etc.), le candidat peut utiliser celui qu'il préfère à condition d'être efficace.

- L'utilisation de recettes toutes faites n'est pas toujours la panacée ; l'interrogateur sera particulièrement attentif à la précision et la complétude des hypothèses nécessaires à leur application ; citons par exemple la règle de d'Alembert qui est quasiment systématiquement utilisée pour déterminer la convergence d'une série même dans les cas où cette règle n'est pas utile, en particulier si la série est une série usuelle qui devrait être bien connue.
- Les candidats doivent s'attendre à être interrogés sur la nature des objets qu'ils manipulent ; ils doivent pouvoir dire s'ils manipulent un nombre, une fonction, un vecteur ; par exemple, il n'est pas acceptable à ce niveau de confondre intégrale et primitive.

6.2 – Polynômes à coefficients réels et complexes

- On n'est pas obligé de passer par un calcul de discriminant pour résoudre $x^2 + 4 = 0$ ou $x^2 - 2\,x + 1 = 0$; cela permettra de disposer de plus de temps pour traiter les autres questions.
- Les racines n-ièmes de l'unité et leurs propriétés, notamment leur somme et leur représentation géométrique, doivent être connues.
- La manipulation de familles de polynômes considérés comme vecteurs de l'espace vectoriel des polynômes fait partie des attendus en algèbre linéaire.

6.3 – Algèbre linéaire

- En algèbre comme ailleurs, on doit veiller à utiliser un vocabulaire précis et à éviter les confusions.
- Les notions liées aux sous-espaces vectoriels (s.e.v. supplémentaires, s.e.v. engendrés par une famille de vecteurs, etc.) doivent être mieux connues.
- Les liens entre les notions de valeur propre, de rang, de noyau, gagneraient en général à être mieux assimilés ; par exemple, les équivalences entre $\det(A) \neq 0$ et $\ker(A) = \{\mathbf{0}_E\}$, entre $\dim(\ker(A)) \geqslant 1$ et « 0 *est valeur propre de* A », entre « le vecteur non nul $\mathbf{u}$ est invariant par l'endomorphisme f » et « $\mathbf{u}$ *est vecteur propre de* f *pour la valeur propre* 1 ».
- Le calcul littéral sur les matrices et les vecteurs doit être maîtrisé, pour caractériser par exemple une matrice symétrique, une matrice orthogonale, un vecteur propre d'une matrice et la valeur propre associée, un produit scalaire associé à une matrice ; l'écriture générale sous forme de somme du produit d'une matrice par un vecteur doit être connue.

6.4 – Analyse

- Les candidats qui pensent à utiliser un développement limité à bon escient, notamment lorsqu'un simple équivalent ne suffit pas, sont en général positivement évalués ; il est par conséquent conseillé de connaître les développements limités usuels (comme celui de $x \mapsto (1+x)^\alpha$ au voisinage de 0, par exemple).
- L'écriture $\lim\limits_{x\to a} f(g(x)) = f\left(\lim\limits_{x\to a} g(x)\right)$ doit être justifiée clairement, même si la fonction f est une fonction usuelle.

6.5 – Intégration

- Lorsqu'on étudie l'intégrabilité d'une fonction sur un intervalle, penser à regarder en premier lieu si celle-ci est continue sur l'intervalle fermé ou, à défaut, sur l'intervalle ouvert, avant de détailler les problèmes éventuels aux bords.
- De trop nombreux candidats mélangent le *Théorème fondamental du calcul intégral* et les théorèmes sur les intégrales dépendant d'un paramètre.

6.6 – Suites et séries

- Pour l'étude de la convergence d'une suite, bien penser à regarder la monotonie et à rechercher des minorants et majorants éventuels.
- Les suites récurrentes doivent être maîtrisées, ce qui est heureusement souvent le cas mais pas toujours.
- Les séries géométriques doivent être parfaitement maîtrisées, ce qui est heureusement très souvent le cas.
- L'écriture $\lim_{n\to\infty} f(u_n) = f\left(\lim_{n\to\infty} u_n\right)$ doit être justifiée clairement, même si la fonction f est une fonction usuelle.

6.7 – Géométrie dans le plan

- De nombreux sujets de géométrie sont posés, y compris parmi les exercices d'informatique. C'est une particularité de la filière PT. Il est plus que conseillé de faire un dessin lisible ; cela permet de mieux comprendre le sujet, et est très apprécié par les examinateurs.
- 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.
- Trop peu de candidats ont réussi à mener à bien l'étude d'une courbe paramétrée, vraisemblablement par manque de pratique ; la réduction du domaine d'étude et la mise en évidence de symétries doivent être maîtrisées, ainsi que l'étude des points singuliers, ce qui est fort heureusement assez fréquent.
- Il sera apprécié qu'un candidat sache paramétrer simplement une conique définie par son équation cartésienne réduite.
- Comme indiqué en préambule, il en sera de même pour la signification géométrique du déterminant de deux vecteurs $\overrightarrow{AB}$ et $\overrightarrow{AC}$.

6.8 – Fonctions de plusieurs variables et géométrie des courbes et surfaces

Liées aux notions de champs, de courbes et de surfaces, les fonctions de plusieurs variables sont indispensables, notamment en ingénierie mécanique. En particulier, il est nécessaire de :

- savoir étudier leur continuité (ou plus généralement leur régularité $\mathcal{C}^1$) ;
- connaître la définition de ses dérivées partielles et savoir les calculer ;
- savoir utiliser la *règle de la chaîne* (dans le programme PT : « *Calcul des dérivées partielles d'ordres 1 et 2 de* $(u,v) \mapsto f(x(u,v), y(u,v))$ ») ;
- savoir passer en coordonnées polaires (changement de variables) ;
- savoir déterminer les points critiques et leur nature ;
- savoir déterminer la tangente et la normale à une courbe ainsi que le plan tangent à une surface, à partir d'équations cartésiennes ou paramétriques.

En revanche, cela ne dispense pas d'être capable de donner des représentations cartésiennes et paramétriques d'éléments géométriques de base comme les droites, les plans, les cylindres ou les sphères.

6.9 – Équations différentielles linéaires

- La résolution d'équations différentielles linéaires à coefficients constants avec second membre doit être maîtrisée, ce qui est heureusement très souvent le cas.
- Les équations différentielles linéaires du premier ordre sans second membre et à coefficients non constants doivent aussi être maîtrisées.

6.10 – Probabilités

- Encore plus qu'ailleurs, il faut lire attentivement l'énoncé et être précis dans son vocabulaire ; un minimum de formalisme est attendu.
- 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 évènements, ou de deux variables aléatoires. Savoir prononcer le terme « *système complet d'évènements* » est bien, mais il est nettement mieux d'être en mesure de détailler de quoi il s'agit.

7 – Exercice d'algorithmique/simulation numérique

Les candidats sont en général bien formés et le nombre d'excellents candidats est toujours en augmentation. Cependant, comme évoqué plus haut, nous avons pu observer une recrudescence de candidats en grande difficulté alors que leur nombre était très faible en 2019. De trop nombreux candidats, qui semblaient pourtant maîtriser les bases, ont avoué spontanément qu'ils ne connaissaient pas la manipulation des chaînes de caractères, ce qui est pour le moins surprenant.

L'effort sur l'aspect « *simulation numérique* » et plus particulièrement sur l'utilisation des tableaux (dont vecteurs et matrices) de la bibliothèque **`numpy`** doit toujours être poursuivi.

7.1 – Conseils généraux

- Lire attentivement l'énoncé ; il arrive très souvent que plusieurs phrases introductives présentent le contexte de l'exercice ; ne pas hésiter à solliciter l'interrogateur si on a le moindre doute, pour clarifier le problème et éviter tout contresens qui pourrait induire des réponses « *hors sujet* ».
- Sauf indication contraire de l'énoncé, **toutes les fonctionnalités de Python 3 sont permises** (fonctions `sum`, `max`, `min`, `sorted`, ..., l'instruction « `x in L` » qui donne un booléen indiquant si l'objet `x` est dans l'itérable `L` ou pas, etc.) ; cela ne dispense pas le candidat d'être capable de répondre s'il est interrogé sur un algorithme de base.
- 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 ; expliquer un code n'est pas le lire mot à mot mais décrire globalement ce qu'il fait et à quoi il sert.
- Ne pas hésiter à utiliser le brouillon mis à disposition avant de se jeter trop rapidement dans la programmation ou pour décrire l'ébauche d'un algorithme à l'interrogateur.
- 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 le mémento, surtout si le conseil en est donné par l'interrogateur.
- Ne pas hésiter à utiliser la console (l'interpréteur) pour effectuer des vérifications élémentaires ou tester les fonctions de Python suggérées par l'énoncé.
- Il est indispensable de savoir utiliser les instructions `help` et **`numpy.info`** : il est normal de ne pas connaître toutes les fonctions apparaissant dans les exercices ; le nom de la fonction à utiliser est très souvent suggéré dans l'énoncé, notamment si cette fonction n'apparaît pas dans le mémento, et il faut donc savoir se renseigner à son sujet et faire des tests élémentaires dans la console.

- Il faut savoir mettre en œuvre une démarche en cas d'erreur : faire des tests élémentaires dans la console, insérer des `print` pour contrôler pas à pas une exécution, lire attentivement et savoir utiliser les messages d'erreurs (lecture de bas en haut, savoir par exemple que « `...index out of range` » est lié à un problème de numérotation dans un objet indexé, que « `...object is not callable` » indique un problème de parenthèses et que « `...object is not subscriptable` » indique un problème de crochets), etc. Il s'agit d'une compétence valorisée par le jury.
- Bien faire la distinction entre les entiers et les flottants, et maîtriser les conséquences induites, dont la différence entre l'opérateur `//` (division entière) et l'opérateur `/` (division flottante).
- La manipulation des entiers est indispensable en informatique et il est essentiel de connaître la numération en bases 10 et 2, ainsi que le passage de l'une à l'autre.
- La manipulation des chaînes de caractères fait aussi partie des capacités exigibles, avec une distinction claire entre `ma_chaine` (nom d'un objet) et `'ma_chaine'` (chaîne de caractères) et aussi la connaissance des méthodes `split`, `strip`, `replace` qui peuvent être utiles pour la lecture de données structurées dans un fichier ASCII.
- L'effort doit être poursuivi dans la lecture d'un fichier texte se trouvant dans un sous-répertoire du répertoire courant ; le plus souvent, le candidat aura à extraire des données numériques à partir de ce fichier ; dans le cas où le fichier contient un texte comportant des lettres accentuées, il est systématiquement encodé selon la norme internationale et multiplateforme UTF-8 ; le rajout de l'option « `encoding='UTF8'` » lors de l'ouverture du fichier est alors en général indiqué dans l'énoncé ou, à défaut, par l'interrogateur ; ce détail ne peut entraîner aucune pénalité.
- La numérotation des éléments, le découpage et la concaténation des chaînes de caractères comme des listes doivent être aussi maîtrisés, dont l'utilisation de l'indexation négative qui ne nécessite pas de connaître le nombre d'éléments (`nom[-1]` pour le dernier élément, `nom[-2]` pour l'avant-dernier, `nom[-3:]` pour les trois derniers, etc.).
- Il n'est pas nécessaire de définir systématiquement une fonction pour chaque tâche demandée, et, plus généralement, il n'y a aucun style de programmation imposé ; le candidat est évalué sur la maîtrise des outils mis à sa disposition et non sur le respect dogmatique de telle ou telle règle ou interdiction le plus souvent arbitraire ; en particulier, la vérification de la conformité des paramètres d'une fonction (par `assert` et/ou des tests) n'est en général pas demandée et fait perdre un temps précieux.
- En revanche, **une fonction doit toujours être testée** de façon appropriée, soit dans l'éditeur (F5 ou F9), soit dans la console, comme cela est spécifié dans l'en-tête de chaque énoncé.
- Préférer une boucle `for` à un `while` quand le nombre d'itérations est connu à l'avance. Préférer également une boucle non indexée « `for objet in iterable` » à une boucle indexée « `for i in range(len(iterable))` » lorsque la connaissance de l'indice i ne sert à rien ; les interruptions de boucle par `return ...` [2] ou même par `break` sont autorisées, à condition de bien faire attention à l'indentation et de pouvoir justifier celles-ci sur le plan algorithmique.
- Comme on le fait en général en mathématiques, réserver les noms `i`, `j`, `k`, `m`, `n` à des entiers et en particulier à des indices, et par conséquent éviter d'écrire « `for i in L` » si `L` ne désigne pas une séquence d'entiers ; ce dernier point est parfois révélateur d'une **confusion** encore trop souvent observée **entre l'objet** (sa valeur s'il s'agit d'un nombre) **et son indice** (sa position dans la séquence).
- La distinction entre une liste (type `list`) et un vecteur (tableau `numpy.ndarray` à un seul indice) doit être parfaitement comprise ; les avantages et les inconvénients de ces deux types complémentaires doivent être connus, ainsi que les fonctions de conversion pour passer de l'un à l'autre (la fonction `numpy.array` et la méthode `tolist`).

2. Noter que `return` est bien un mot-clef du langage Python et non pas une fonction. Lors de la session 2021, de nombreux candidats ont écrit « `return(a)` » au lieu de « `return a` », ce qui n'est pas faux mais très inhabituel. Lorsque la fonction renvoie plusieurs objets, les écritures « `return (a, b, ...)` » et « `return a, b, ...` » sont équivalentes.

7.2 – Gestion du temps

Quelques candidats perdent un temps considérable avec des pratiques peu adaptées pour une épreuve de 30 minutes :

- Il est bon de connaître et de savoir utiliser par exemple les fonctions intrinsèques `min`, `max`, `sum`, `sorted`, les méthodes `append`, `extend`, `sort`, `index` pour les listes, les méthodes `min`, `max`, `argmin`, `argmax`, `sum`, `mean`, `std`, `transpose`, `conjugate`, ... pour les tableaux `numpy.ndarray` (`T.real` et `T.imag` aussi pour un tableau de complexes) ; ces méthodes existent aussi sous forme de fonctions dans le module `numpy`.
- Les techniques de *slicing* peuvent être utilisées :
  - ◇ « `U[debut:fin:pas]` » pour une séquence (liste, chaîne de caractères, vecteur, etc.) ;
  - ◇ « `M[Ldeb:Lfin:dL,Cdeb:Cfin:dC]` » pour une matrice (tableau à 2 indices).

  Ce dernier point particulièrement important fait l'objet d'un encadré spécifique dans le mémento fourni aux candidats.
- Il a encore été observé cette année un abus de la méthode `append` pour créer des séquences très simples. Des exemples caricaturaux observés plusieurs fois :

```
L = []
for i in range(10) :
    L.append(i)
```

au lieu de

```
L = list(range(10))
```

```
L = []
for x in np.linspace(-2.5,2.5,51) :
    L.append(x)
V = np.array(L)
```

au lieu de

```
V = np.linspace(-2.5,2.5,51)
```

- Même si les listes en compréhension ne sont pas exigibles, leur utilisation maîtrisée permet de gagner en efficacité et en lisibilité ; comme en 2019, de nombreux candidats les ont utilisées en 2021.
- Ne pas hésiter à réutiliser les fonctions créées dans les questions précédentes, ou même à créer de petites fonctions intermédiaires si cela peut être utile ; les exercices sont très souvent structurés dans cet esprit.
- Dans le rapport 2018, il était indiqué : « *L'écriture systématique de commentaires et d'en-têtes (`"docstring"`) pour les fonctions est déconseillée pour l'oral ; même si elle est légitimement préconisée en génie logiciel, elle fait perdre un temps précieux ; les explications peuvent être données oralement par le candidat.* ». Comme en 2019, ce point a été respecté en 2021.
- L'effort pour éviter les écritures redondantes contenant des booléens doit être poursuivi ; par exemple, si une fonction `test` a été définie précédemment et que `test(a,b)` donne un booléen, on écrira :

```
t = test(a,b)
```

et non pas

```
if test(a,b) == True :
    t = True
else :
    t = False
```

7.3 – Algorithmique

- Les algorithmes du cours et leurs coûts de calcul doivent être connus (algorithmes de tri, méthodes par dichotomie, de Newton, d'Euler, des trapèzes, pivot de Gauss, algorithme d'orthonormalisation de Gram-Schmidt, algorithme d'Euclide, etc.). Leur connaissance est fréquemment évaluée.
- Cela ne suffit pas ; en préambule, il faut maîtriser des algorithmes simples (comme par exemple l'extraction des éléments distincts d'un objet itérable, la détermination du rang de la première répétition dans un objet itérable, ou l'extraction à partir d'un entier de la liste de ses chiffres en écriture décimale), et aussi respecter quelques règles de base dont la plus importante d'entre elles : « **Ne pas appeler plusieurs fois la même fonction avec les mêmes arguments** » ; son non-respect montre une mauvaise compréhension de l'algorithmique et de la programmation de la part du candidat.

  Un exemple caricatural à ne surtout pas suivre :

```
for i in range(1, len(ma_fonction(1, 2, 3)[0]) ) :
    X = ma_fonction(1, 2, 3)[:, 0]
    Y = ma_fonction(1, 2, 3)[:, i]
    plt.plot(X, Y, ".", label="cas n°{}".format(i))
```

  au lieu d'écrire plus simplement et de façon autrement plus efficace :

```
T = ma_fonction(1, 2, 3)
for i in range(1, T.shape[1] ) :
    plt.plot(T[:,0], T[:,i], ".", label="cas n°{}".format(i))
```

- La distinction claire entre *algorithme récursif* et *algorithme itératif* doit être acquise ; dans l'écriture d'une fonction récursive, un soin particulier doit être porté à la condition d'arrêt.

8 – Analyse des résultats

En 2021, 1597 candidats ont passé l'oral de « *Mathématiques et algorithmique* ». Chacun des 12 jours de l'oral, les 6 à 9 jurys se sont efforcés de poser des exercices balayant l'ensemble du programme, tant en mathématiques qu'en algorithmique et simulation numérique.

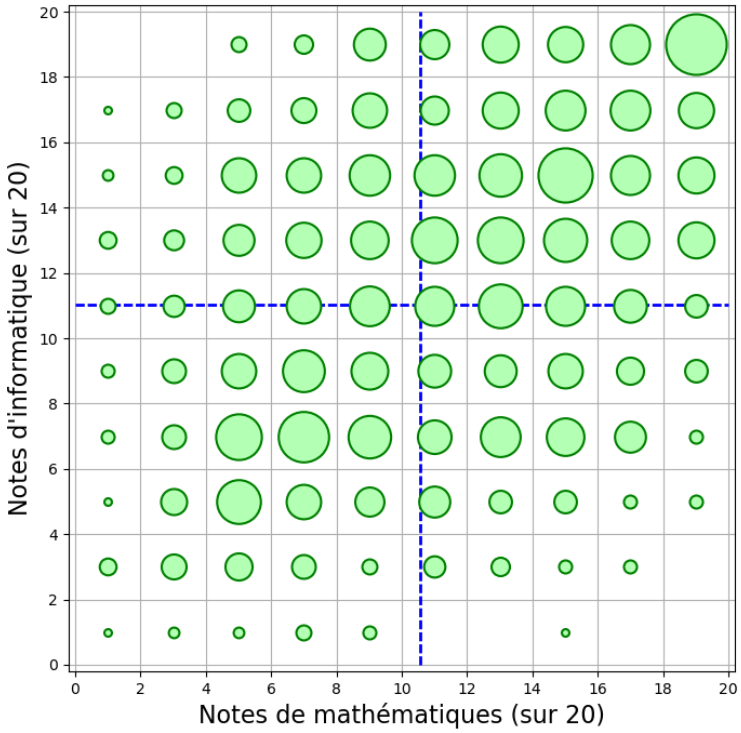
Ainsi, 213 exercices différents d'analyse et de probabilités ont été proposés à 860 candidats contre 166 exercices différents de géométrie et d'algèbre proposés à 737 candidats.

160 exercices différents d'informatique à dominante algorithmique ont été posés à 968 candidats, contre 155 exercices à dominante « *simulation numérique* » pour 629 candidats.

Les statistiques sur les notes sont les suivantes [3] :

Oral 2021	**Note (sur 20)**	Math. (sur 10)	Algo. (sur 10)
Moyenne	**10,79**	5,28	5,52
Écart-type	**4,12**	2,58	2,42
Minimum	1	0	0
Maximum	20	10	10

3. Rappelons que seule la note globale est communiquée au candidat.



Distribution des notes 2021

Éric Ducasse, Coordonnateur de l'épreuve orale de « *Mathématiques et algorithmique* » de la Banque PT,
Le 12 juillet 2021.
eric.ducasse@ensam.eu

Annexe 1 – À propos du mémento pour l'oral

La version actuelle du mémento de l'oral date d'août 2018 (voir **site de la Banque PT**). Elle est destinée uniquement au passage de l'oral. Ce mémento est à disposition du candidat sous forme d'une seule feuille plastifiée au format A3 recto-verso.

Un mémento à vocation pédagogique, plus fourni, est disponible sur l'espace numérique de travail *Arts et Métiers* **https://savoir.ensam.eu/moodle/course/view.php?id=1428** destiné aux étudiants des Arts et Métiers. Des supports de référence sont également disponibles sur ce site.

Annexe 2 – Exemples d'exercices d'informatique

Ces exercices ayant été posés de nombreuses fois au cours des dernières sessions, ils sont communiqués aux futurs candidats à titre d'exemples. Attention : il ne sont pas forcément représentatifs de tous les exercices se trouvant dans la banque d'exercices.

D'autres exercices publiés sont joints aux rapports 2015, 2016, 2018 et 2019.

2021.1 – **Exercice à dominante algorithmique**

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

En Biélosyldavie, la monnaie est le Zlotly. La banque centrale de Biélosyldavie émet uniquement des billets de 57, 62 et 72 Zlotlys.

1. Est-il possible de payer (exactement) 400 Zlotlys ? 600 Zlotlys ?

2. Créer une fonction `comptage` qui à un nombre n associe le nombre de manières (*i.e.* de combinaisons de billets possibles) pour payer (exactement) n Zlotlys.

3. Un enfant achète un croissant coûtant 4 Zlotlys. On sait qu'il a au plus 600 Zlotys sur lui, qu'il paie avec au moins deux types de billets différents et que le pâtissier lui rend la monnaie exacte. Combien cet enfant a-t-il donné au pâtissier ?

2021.2 – **Exercice à dominante algorithmique**

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

Dans une liste ou une chaîne de caractères, on appelle « *composante* » une sous-liste ou sous-chaîne correspondant à la répétition d'un même élément ou caractère, précédée et suivie par rien ou un autre élément ou caractère.

Les composantes de `[1,1,0,0,1,1,1,0,1,1,1]` sont `[1,1]` suivie de `[0,0]`, `[1,1,1]`, `[0]`, et `[1,1,1]` ; celles de `'aaaaggaadaaa'`, `'aaaa'` suivie de, `'gg'`, `'aa'`, `'d'` et `'aaa'`.

Une composante peut ensuite se coder sous la forme d'un couple `[n,c]`, où `n` désigne le nombre de répétitions de l'élément ou du caractère `c`.

1. Écrire une fonction **`coupures`** dont l'argument est une liste ou une chaîne de caractères S, qui renvoie la liste des indices i tels que `S[i]` $\neq$ `S[i - 1]`, à la fin de laquelle on rajoute la longueur de S.

   `coupures([1,1,0,0,1,1,1,0,1,1,1])` donne `[2,4,7,8,11]` et `coupures('aaaaggaadaaa')`, `[4,6,8,9,12]`.

2. Écrire une fonction `CP` de même argument S qui renvoie une liste résultant de la mise bout-à-bout des couples codant les composantes de S.

   Ainsi, `CP([1,1,0,0,1,1,1,0,1,1,1])` donne `[2,1,2,0,3,1,1,0,3,1]` et `CP('aaaaggaadaaa')`, `[4,'a',2,'g',2,'a',1,'d',3,'a']`.

3. Écrire une fonction **`nbits`** dont l'argument est un entier naturel n et qui renvoie le plus petit entier a tel que $n \leqslant 2^a$.

4. Écrire une fonction **`nbelem`** dont l'argument est une liste ou une chaîne de caractères S et qui renvoie le nombre d'éléments ou caractères distincts contenus dans S. `nbelem([1,1,0,0,1,1,1,0,1,1,1])` donne 2 et `nbelem('aaaaggaadaaa')` donne 3.

5. En déduire une fonction **`nbtotalbits`** qui renvoie le nombre minimum de bits nécessaires pour coder S une fois connu l'ensemble des éléments ou caractères distincts contenus dans S (par exemple, dans `'aaaaggaadaaa'`, chaque lettre peut être codée sur 2 bits avec `'a'`:`00`, `'d'`:`01`, `'g'`:`10`).

6. Si S est une liste composée de 45 zéros suivis de 76 uns, quelle solution peut-on trouver pour minimiser le stockage de S ?

2021.3 – **Exercice à dominante algorithmique**

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

---

1. Rappeler le théorème de la division euclidienne dans $\mathbb{N}$.

2. Soient a et b deux entiers naturels avec $b \neq 0$. Pour calculer le reste et le quotient de la division euclidienne de a par b, on peut retrancher b à a tant que $a \geqslant b$.

   En déduire une fonction **`DE1`** de deux arguments a et b renvoyant le couple **`(quotient,reste)`** de la division euclidienne de a par b.

3. Écrire une fonction **`W`** de deux arguments a et b qui renvoie le plus petit nombre w supérieur ou égal à a de la forme $w = 2^k\, b$ où k est un entier naturel.

4. On considère le code *Python* suivant :

```
def DE2(a,b) :
    w, q, r = W(a,b), 0, a
    while w != b :
        w //= 2
        q *= 2
        if w <= r :
            q += 1
            r -= w
    return q,r
```

   a) Justifier que cette boucle se termine.

   b) À chaque fin d'itération, que peut-on dire de la quantité $w\,q + r$?

   c) Comment se traduisent en binaire les instructions **`q*=2`** et **`w//=2`** ?

   d) Finalement, que fait ce code ?

5. Laquelle des deux fonctions **`DE1`** et **`DE2`** vous paraît la plus rapide ?

2021.4 – **Exercice à dominante 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 **`petitsdiviseurs`** d'argument un entier naturel n et renvoyant la liste des diviseurs de n dont le carré est inférieur ou égal à n. Tester cette fonction sur 47, 49 et 60.

2. Écrire une fonction **`estPremier`** d'argument un entier naturel n renvoyant un booléen indiquant si n est premier ou pas.

3. Écrire une fonction **`nbp`** d'argument n renvoyant le nombre de nombres premiers compris entre 2 et n.

4. Écrire une fonction **`premapres`** d'argument n renvoyant le plus petit nombre premier strictement supérieur à n. Tester **`premapres`** pour $n = 1000$.

5. Écrire une fonction **`premavant`** d'argument n renvoyant le plus grand nombre premier inférieur ou égal à n. Tester **`premavant`** pour $n = 1000$.

2021.5 – **Exercice à dominante algorithmique**

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

---

1. Que font les fonctions **`L2int`** et **`int2L`** suivantes ?

```
def L2int(L) :
    # L est une liste d'entiers naturels
    ch = ""
    for e in L :
        ch = ch + str(e)
    return int(ch)
def int2L(n) :
    # n est un entier naturel
    L = []
    for car in str(n) :
        L.append(int(car))
    return L
```

Soit la fonction ϕ de $\mathbb{N}$ dans $\mathbb{N}$ telle que, pour tout entier naturel n, $\phi(n)$ est le nombre composé des chiffres décrivant le nombre n. Par exemple :

1112	$\mapsto$	3112	(trois uns, un deux)
29	$\mapsto$	1219	(un deux, un neuf)
333	$\mapsto$	33	(trois trois)
1211	$\mapsto$	111221	(un un, un deux, deux uns)

Soit une suite d'entiers naturels $(u_n)_{n\in\mathbb{N}}$ définie par : $u_0 = a$ et $\forall n \in \mathbb{N}$, $u_{n+1} = \phi(u_n)$.
On suppose que a ne comporte pas plus de 9 chiffres identiques à la suite, de sorte que le nombre de chiffres consécutifs dans u_n est toujours un chiffre.

2. Écrire une fonction **`lire`** d'argument un nombre n et renvoyant $\phi(n)$.

3. Afficher les dix premiers termes de la suite u_n avec comme valeur initiale $a = 44440$.

4. Existe-t-il une valeur de a inférieure à 10000 telle que la suite u_n soit constante ?

5. Définir une fonction **`ecrire`** d'argument k qui renvoie la valeur de n telle que $\phi(n) = k$ si elle existe, et -1 sinon.

6. Déterminer la valeur de a telle que a ne s'écrive pas $\phi(b)$ et que **`31123113`** soit un terme de la suite $(u_n)_{n\in\mathbb{N}}$ ayant comme valeur initiale a.

2021.6 – **Exercice à dominante 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 $(T_n)_{n\in\mathbb{N}}$ à valeurs dans $\{0,1\}$ définie par (« $a \equiv b \mod c$ » signifie que a et b sont égaux *modulo* c) :

$$\forall n \in \mathbb{N} \quad T_n \equiv \left(\sum_{i=0}^{+\infty} b_i(n)\right) \mod 2\,, \text{ où les } b_i(n) \text{ sont les chiffres de l'écriture en base 2 de } n.$$

Par exemple (le code binaire est surligné), comme $0 = \overline{0}$, $1 = \overline{1}$, $2 = \overline{10}$, et $3 = \overline{11}$, les quatre premiers termes de la suite sont : $0, 1, 1, 0$.

1. Écrire une fonction **`binaire`** d'argument un entier n qui revoie la liste des chiffres de l'écriture de n en base 2.

2. En déduire une fonction calculant T_n pour $n \in \mathbb{N}$.

3. Montrer que pour tout $n \in \mathbb{N}$, T_{2n} et T_{2n+1} s'expriment simplement en fonction de T_n.

   Programmer une fonction récursive calculant T_n.

4. Soit un entier naturel N.

   On pose $I = \left\{i \in \mathbb{N}\,,\ 0 \leqslant i \leqslant 2^N-1\,,\ T_i = 0\right\}$ et $J = \left\{j \in \mathbb{N}\,,\ 0 \leqslant j \leqslant 2^N-1\,,\ T_j = 1\right\}$.

   On peut démontrer que : $\forall k \in \mathbb{N}\,,\ 0 \leqslant k \leqslant N-1\,,\ \sum_{i\in I} i^k = \sum_{j\in J} j^k$.

   Vérifier cette propriété pour $N = 5$.

2021.7 – **Exercice à dominante algorithmique**

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

---

1. Soit $k \in \mathbb{N}^*$; on note $m(k)$ le plus grand nombre qui est une puissance de 2 et qui divise k ; par exemple, $m(12) = 4$. Définir la fonction `m`.

2. Soit la suite $(c_n)_{n\in\mathbb{N}^*}$ définie par : $\forall n \in \mathbb{N}^* \quad c_n = \dfrac{1}{2}\left(1 + \dfrac{n}{m(n)}\right)$.

   Montrer que les termes de cette suite sont des entiers naturels, puis écrire une fonction `c1` d'argument un entier naturel n et renvoyant l'entier c_n.

   Donner les valeurs de c_n pour $1 \leq n \leq 16$.

3. Quelle relation simple existe-t-il entre c_{2n} et c_n pour $n \geq 1$? Que vaut c_{2n+1} ?

   En déduire une fonction `c2` renvoyant c_n après l'avoir calculé sans utiliser la fonction m.

4. Une façon équivalente de calculer c_n consiste en :
   — écrire le nombre n en base 2 ;
   — supprimer tous les zéros consécutifs à partir de la droite ;
   — ajouter 1 au nombre binaire obtenu ;
   — supprimer le 0 de droite ainsi obtenu.
   On obtient ainsi l'écriture binaire de c_n.

   Après avoir vérifié à la main sur un exemple que l'algorithme fonctionne, écrire une fonction `c3` qui calcule c_n selon l'algorithme indiqué puis renvoie sa valeur. On pourra utiliser la fonction `bin` pour obtenir l'écriture binaire d'un entier.

   Comment prouver la validité de cet algorithme ?

2021.8 – **Exercice à dominante algorithmique**

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

---

1. Pour tout entier naturel n, on pose $t_n = 0 + 1 + 2 + \cdots + n = \sum_{k=0}^{n} k$. Rappeler la valeur de t_n en fonction de n. Les nombres t_n sont appelés « *nombres de type* $\mathcal{T}$ ».

   Construire la liste des nombres de type $\mathcal{T}$ inférieurs ou égaux à 100.

2. Écrire une fonction **`sommes`** de trois paramètres, deux listes L_1 et L_2 d'entiers naturels et un entier naturel n, renvoyant la liste strictement croissante des nombres inférieurs ou égaux à n qui s'écrivent comme somme d'un élément de L_1 et d'un élément de L_2.

3. Utiliser la fonction précédente pour obtenir la liste strictement croissante des entiers compris entre 0 et 100 qui s'écrivent comme somme de trois nombres de type $\mathcal{T}$. Que constate-t-on ? Que peut-on conjecturer ?

4. Écrire une fonction **`decompositions`** d'argument un entier naturel n, renvoyant la liste de tous les triplets croissants de nombres de type $\mathcal{T}$ dont la somme vaut n. Chacun de ces triplets est une *décomposition de* n.

   Déterminer alors le plus petit entier naturel N ayant au moins 10 décompositions différentes.

5. Évaluer le coût de calcul de la fonction **`decompositions`**.

2021.9 – **Exercice à dominante algorithmique**

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

---

1. Créer une fonction **`parcours`** d'argument une liste $L = [a_1, a_2, \cdots, a_n]$ qui *modifie* cette liste en permutant a_k et a_{k+1} si $a_{k+1} < a_k$, pour k variant de 1 à $n-1$, et qui renvoie le nombre de permutations effectuées.

   Par exemple, si $L = [5, 4, 1, 6]$, alors **`parcours(L)`** renvoie 2 et la liste L devient $L = [4, 1, 5, 6]$.

2. Que peut-on dire du dernier élément d'une liste à laquelle on a déjà appliqué une fois la fonction **`parcours`** ?

3. On veut utiliser cette fonction pour trier par ordre croissant une liste selon le principe suivant : Pour une liste L donnée en entrée on répète l'application de la fonction **`parcours`** jusqu'à ce que L devienne invariante par cette fonction.

   Écrire une fonction **`tri1`** qui ordonne une liste donnée en entrée selon ce principe et qui renvoie le nombre d'itérations effectuées.

4. Expliquer pourquoi la fonction **`tri1`** donne bien le résultat voulu en une durée finie. Évaluer son coût de calcul dans le meilleur des cas et dans le pire des cas.

5. Écrire une fonction **`tri2`** comme une amélioration de la fonction **`tri1`** en évitant les comparaisons inutiles.

2021.10 – **Exercice à dominante algorithmique**

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

On considère un jeu de 32 cartes. Il est formé de couples de 8 valeurs ordonnées, `valeurs=["7", "8", "9", "10", "V", "D", "R", "A"]`, et de 4 « *couleurs* », `couleurs=["T", "K", "C", "P"]` (Trèfle, Carreau, Coeur, Pique). On distribue au hasard une « *main* », c'est-à-dire 5 cartes distinctes, et on s'intéresse à des mains particulières :

— les « *couleurs* » (5 cartes de même « *couleur* ») ;
— les « *quintes* » (5 cartes de valeurs qui se suivent) ;
— les « *quintes floches* » (5 cartes de même « *couleur* » et de valeurs qui se suivent).

1. Construire l'ensemble **cartes** des 32 cartes à partir des deux listes **valeurs** et **couleurs**, chaque carte étant représentée par la paire $[valeur, couleur]$. Vérifier que le nombre de cartes obtenu est correct.

2. Écrire une fonction **tirerMain** sans argument qui renvoie une liste de 5 cartes distinctes tirées au hasard. On pourra utiliser la fonction **sample** du module **random**.

3. Écrire une fonction **estCouleur** d'argument une main et qui renvoie un booléen indiquant si cette main est une « *couleur* » ou pas.

4. Déterminer la liste **quintes** de toutes les suites possibles de 5 valeurs d'une quinte. En déduire une fonction **estQuinte** d'argument une main et qui renvoie un booléen indiquant si cette main est une « *quinte* » ou pas. [Indication : pour comparer deux listes de valeurs indépendamment de leur ordre, on pourra utiliser la fonction intrinsèque `sorted`].

5. À partir de 50000 tirages aléatoires de mains, estimer la probabilité d'obtenir une « *couleur* », celle de tirer une « *quinte* » et celle de gagner une « *quinte floche* ».

6. Comparer ces estimations avec les probabilités obtenues par dénombrement. On pourra utiliser la fonction **comb** du module **scipy.special**, ou faire autrement.

2021.11 – **Exercice à dominante algorithmique**

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

---

1. Observer et expliquer ce que fait le code *Python* suivant, n désignant un entier naturel non nul :

```
def progX(n):
    if n == 1:
        res = False
    else:
        d = 2
        res = True
        while d*d <= n and res :
            res = ( n%d != 0 )
            d += 1
    return res
```

2. Améliorer la fonction précédente en remarquant que, pour $d \geqslant 3$, d peut progresser de 2 en 2.

3. Montrer que $2\,k^2 + 29$ est premier pour tout entier k vérifiant $0 \leqslant k \leqslant 27$.

4. Combien y-a-t-il de nombres premiers la forme $2\,k^2 + 29$, pour k vérifiant $0 \leqslant k \leqslant 100$?

5. Écrire une fonction **`tester`** d'argument un entier n et qui renvoie un booléen. Ce booléen vaudra `True` si et seulement si :

$$2\,k^2 + n \text{ est premier pour tout entier } k \text{ vérifiant } 0 \leqslant k \leqslant n-2. \tag{1}$$

6. Trouver tous les entiers n inférieurs à 100 vérifiant la propriété (1).

2021.12 – **Exercice à dominante algorithmique**

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

N'importe quel ensemble fini Ω d'entiers naturels peut être représenté par une liste P de booléens de la façon suivante : Ω est l'ensemble des entiers i pour lesquels `P[i]` est vrai.

Par exemple, la liste `[False,True,False,False,True,True,False]` représente l'ensemble $\{1, 4, 5\}$.

1. Expliquer pourquoi il n'y a pas unicité de la représentation de Ω.

2. Écrire une fonction **`card`** d'argument une liste P de booléens qui renvoie le cardinal de l'ensemble représenté par P.

3. Écrire une fonction **`EtoB`** d'argument une liste L d'entiers naturels, dans le désordre et avec ou sans répétition, qui renvoie la liste de booléeens la plus courte représentant l'ensemble des entiers de L.

   Tester **`EtoB`** sur la liste `[5,1,2,6,2]`.

4. Écrire une fonction **`bunion`** de deux arguments, deux listes de booléeens P et Q, qui renvoie la liste de booléens la plus courte représentant l'union des ensembles représentés par P et Q.

5. Écrire une fonction **`BtoE`** d'argument une liste P de booléens qui renvoie l'ensemble d'entiers représenté par P.

6. En déduire une fonction **`union`** de deux arguments, des listes d'entiers naturels, qui renvoie l'union de ces deux ensembles.

7. On peut coder également un ensemble d'entiers naturels Ω par l'entier $n(\Omega) = \sum_{i \in \Omega} 2^i$.

   Par exemple, $n(\{1, 3\}) = 10$.

   Écrire une fonction **`BtoN`** d'argument une liste P de booléens qui renvoie l'entier codant l'ensemble représenté par P, ainsi qu'une fonction **`NtoB`** d'argument n un entier et renvoyant la liste de booléens la plus courte représentant l'ensemble codé par n.

2021.13 – **Exercice à dominante algorithmique**

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

On sait que l'écriture décimale d'un nombre rationnel strictement positif est périodique à partie d'un certain rang. Par exemple on a :

$$\frac{4741}{280} = 16,\, 932\,142857\,142857\,142857\cdots \quad ,$$

que l'on récrit :

$$\frac{4741}{280} = 16,\, 932\,\overline{142857} \quad .$$

La liste de chiffres $[1, 4, 2, 8, 5, 7]$ est appelée « *partie périodique* » de l'écriture décimale, dans laquelle elle est précédée de la « *partie non périodique* », la liste de chiffres $[9, 3, 2]$.

On appelle donc « *écriture décimale périodique* » d'un nombre rationnel a/b la liste $[n, A, P]$, où n désigne la partie entière de a/b, A la partie non périodique de son écriture décimale, et P sa partie périodique.

1. Écrire une fonction **`PE`** de deux arguments a et b qui renvoie la partie entière de a/b.

2. Écrire une fonction **`decimales`** de trois arguments a, b et k qui renvoie la liste des k premières décimales de a/b, calculées par divisions euclidiennes successives.

3. Écrire une fonction **`EDP`** de deux arguments a et b qui renvoie l'écriture décimale périodique de a/b, sous forme d'une liste $[n, A, P]$. La partie périodique est identifiée dès qu'un reste de division euclidienne réapparaît. Il faut donc stocker les restes successifs.

4. Réciproquement, on peut retrouver la fraction à partir de l'écriture périodique en remarquant sur l'exemple que :

$$y = 0,\, \overline{142857} \quad \Longleftrightarrow \quad y\,10^6 = 142857 + y \quad \Longleftrightarrow \quad y = \frac{142857}{10^6 - 1} \quad .$$

   Écrire la fonction **`FR`** d'argument E, une écriture décimale périodique, qui renvoie le couple (a, b) d'entiers premiers entre eux tel que E est l'écriture décimale périodique de a/b.

   On pourra utiliser la fonction **`gcd`** du module **`fractions`**.

   Tester **`FR`** sur les exemples : $10,\overline{3}$; $2,125\,\overline{0}$; $1,2\overline{9}$.

5. Écrire une fonction **`somme`** de deux arguments **`E1`** et **`E2`**, deux écritures décimales périodiques, qui renvoie l'écriture décimale périodique de la somme des deux fractions rationnelles correspondantes.

2021.14 – **Exercice à dominante algorithmique**

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

Soient un entier naturel non nul n et l'ensemble $\mathbb{E}_n = \{1, 2, \cdots, n\}$. Chaque permutation p de l'ensemble $\mathbb{E}_n$ (bijection de $\mathbb{E}_n$ dans lui-même) peut être représentée par la liste $[\,p(1), p(2), \cdots, p(n)\,]$.

Par exemple, $[1, 2]$ et $[2, 1]$ représentent les deux permutations de $\mathbb{E}_2$.

1. Observer et expliquer ce que fait le code suivant :

```
L=[[1,2],[2,1]]
M=[ ]
for p in L:
    for k in range(3):
        m = p[:]
        m.insert(k,3)
        M.append(m)
print(M)
```

2. Écrire une fonction **permut** d'argument un entier naturel non nul n renvoyant la liste des permutations de l'ensemble $\mathbb{E}_n$. tester cette fonction pour n entre 3 et 6.

3. Soit p une permutation de $\mathbb{E}_n$. On appelle « *point fixe* » de p, tout élément k de $\mathbb{E}_n$ tel que $p(k) = k$.

   Écrire une fonction **nbpf** d'argument L, une liste représentant une permutation p, renvoyant le nombre de points fixes de p.

4. On appelle « *dérangement de* $\mathbb{E}_n$ », toute permutation de $\mathbb{E}_n$ sans point fixe.

   Écrire une fonction **derang** d'argument un entier naturel non nul n renvoyant la liste de tous les dérangements de $\mathbb{E}_n$.

5. On démontre que le nombre de dérangements de $\mathbb{E}_n$, noté $d(n)$, vérifie :

$$d(n) = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!} \quad .$$

   Vérifier cette formule pour n compris entre 2 et 9.

2021.15 – **Exercice à dominante algorithmique**

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

On définit deux suites $(t_n)_{n\in\mathbb{N}^\star}$ et $(E_n)_{n\in\mathbb{N}^\star}$. La suite $(E_n)_{n\in\mathbb{N}^\star}$ se définit par récurrence : $E_1 = \mathbb{N}^\star$ et, pour tout entier naturel $n \geqslant 2$, E_n est l'ensemble des éléments de E_{n-1} de rangs :

- au moins égaux à 2 ;
- non multiples de n (numérotation commençant à 1).

Pour tout n, t_n est le premier élément de E_n.

Les premières itérations donnent, en encadrant t_n et en barrant les autres éléments qui disparaissent à la ligne suivante :

E_1	:	$\boxed{1}$	~~2~~	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	$\cdots$
E_2	:	$\boxed{3}$	5	~~7~~	9	11	~~13~~	15	17	~~19~~	21	23	~~25~~	$\cdots$
E_3	:	$\boxed{5}$	9	11	~~15~~	17	21	23	~~27~~	29	33	35	~~39~~	$\cdots$
E_4	:	$\boxed{9}$	11	17	21	~~23~~	29	33	35	41	~~45~~	47	51	$\cdots$

1. Écrire une fonction **`enlever`** de deux arguments, une liste L et un entier naturel n au moins égal à 2, qui renvoie la liste L privée de ses éléments de rangs multiples de n (numérotation commençant à 1).

   Par exemple, **`enlever([3,5,7,9,11,13,15],3)`** donne **`[3,5,9,11,15]`**.

2. Écrire une fonction **`LT`** d'argument k qui renvoie la liste des t_n inférieurs ou égaux à k. Par exemple, **`LT(20)`** donne **`[1,3,5,9,11,17]`**.

   Faire afficher **`LT(100)`**.

3. Pour k entier naturel non nul, on note u_k le nombre de t_n inférieurs ou égaux à k. Écrire une fonction **`LU`** d'argument K qui renvoie la liste des u_k, pour k variant de 1 à K. Par exemple **`LU(9)`** donne **`[1,1,2,2,3,3,3,3,4]`**.

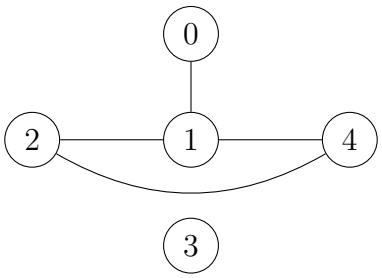
4. Conjecturer la limite ℓ de u_k^2/k lorsque k tend vers l'infini.

5. À l'aide notamment des fonctions **`plot`** et **`show`** du module **`matplotlib.pyplot`**, faire tracer u_k et $\sqrt{\ell\, k}$ en fonction de k, pour k variant de 1 à 10000.

2021.16 – **Exercice à dominante algorithmique**

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

Un *graphe* est un ensemble constitué de n sommets numérotés de 0 à $n-1$, reliés par des arêtes :



Chaque arête est désignée par le couple des numéros de sommets qu'elle relie ; ainsi, l'arête reliant les sommets i et j est désignée par le couple (i, j), avec $0 \leqslant i < j \leqslant n-1$.

Pour décrire un graphe, il suffit de connaître n le nombre de sommets et L la liste de ses arêtes données dans n'importe quel ordre. Par exemple, le graphe représenté ci-dessus contient 5 sommets et la liste de ses arêtes est `[(2,4),(1,4),(1,2),(0,1)]`.

1. Écrire une fonction **`non_isoles`** d'argument la liste L des arêtes d'un graphe, renvoyant la liste de ses sommets non isolés, c'est-à-dire reliés par au moins une arête.

2. Écrire une fonction **`degre`** de deux arguments L et k qui renvoie le degré du sommet k dans le graphe d'arêtes L, c'est-à-dire le nombre d'arêtes dont ce sommet est l'une des extrémités.

   Par exemple, dans le graphe ci-dessus, les degrés respectifs des sommets 0, 1 et 3 sont 1, 3 et 0.

3. On appelle « *liste d'adjacence* » d'un graphe à n sommets, la liste de longueur n dont l'élément d'indice k est lui-même une liste répertoriant l'ensemble des sommets reliés par une arête au sommet k.

   Écrire une fonction **`adjacence`** de deux arguments, le nombre n de sommets d'un graphe et la liste L de ses arêtes, renvoyant la liste d'adjacence du graphe.

   Pour l'exemple ci-dessus, la liste d'adjacence est `[[1],[0,2,4],[1,4],[],[1,2]]`.

4. Écrire une fonction **`degmax`** d'argument une liste d'adjacence A d'un graphe, renvoyant le sommet dont le numéro est le plus grand parmi tous ceux de degré maximum.

5. On dit qu'une liste de sommets $[n_0, ..., n_p]$ est un « *chemin de longueur p* » lorsque, pour tout i compris entre 0 et $(p-1)$, il existe une arête entre les sommets n_i et n_{i+1}.

   Écrire une fonction **`chemin`** de trois arguments, le nombre n de sommets d'un graphe, la liste L de ses arêtes et une liste C de sommets, qui renvoie un booléen indiquant si C est effectivement un chemin du graphe, ou pas.

   Vérifier que `[0,1,2]` et `[0,1,4,2,1]` sont des chemins du graphe ci-dessus, mais pas `[0,2,1]`.

6. Écrire une fonction **`relies`** de deux arguments, une liste L d'arêtes et le numéro k d'un sommet, renvoyant la liste des numéros des sommets reliés au sommet k par au moins un chemin.

2021.17 – **Exercice à dominante 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 **`prod`** de deux arguments, deux listes $L = [\ell_0, \cdots, \ell_{d-1}]$ et $C = [c_0, \cdots, c_{d-1}]$ de mêmes longueurs, renvoyant le produit scalaire $\sum_{i=0}^{d-1} \ell_i \, c_i$ des vecteurs représentés par L et C.

Une matrice carrée d'ordre d est représentée ici par la liste de ses lignes mises bout-à-bout. Par exemple, les listes `[0,1,2,3]` et `[0,1,2,3,4,5,6,7,8]` peuvent représenter les matrices :

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \quad \text{et} \quad \begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}.$$

2. Écrire une fonction **`lig`** de deux arguments, une liste M représentant une matrice carrée et un entier naturel i, qui renvoie la ligne i de la matrice représentée par M, sous forme d'une liste.

   Par exemple `lig([0,1,2,3],0)` et `lig([0,1,2,3],1)` donnent respectivement `[0,1]` et `[2,3]`.

3. Écrire une fonction **`col`** de deux arguments, une liste M représentant une matrice carrée et un entier naturel j, qui renvoie la colonne j de la matrice représentée par M, sous forme d'une liste.

   Par exemple `col([0,1,2,3],0)` et `col([0,1,2,3],1)` donnent respectivement `[0,2]` et `[1,3]`.

4. Écrire une fonction **`mul`** de deux arguments, deux listes représentant des matrices carrées M et N d'ordres identiques, qui renvoie la liste représentant le produit $M\,N$.

5. Écrire une fonction **`puis`** de deux arguments, une liste représentant un matrice carrée M et un entier naturel non nul p, qui renvoie la liste représentant M^p, en tenant compte de la remarque suivante : si $p = 2\,q$ est pair, $M^p = (M^q)^2$, et si $p = 2\,q + 1$ est impair, $M^p = M\ (M^q)^2$.

   Par exemple, `puis([0,1,1,1],2)`, `puis([0,1,1,1],3)` et `puis([0,1,1,1],5)` doivent donner respectivement les listes `[1,1,1,2]`, `[1,2,2,3]` et `[3,5,5,8]`.

2021.18 – **Exercice à dominante simulation numérique**

Cet exercice est prévu pour le langage Python (et ses bibliothèques numpy, scipy, matplotlib). À chaque question, les instructions ou les fonctions écrites devront être testées.

Longueur de ligne brisée et lecture de données dans un fichier

1. Le tableau suivant donne les coordonnées d'un point $M(t)$ (abscisse $x(t)$ et ordonnée $y(t)$) en divers instants t :

t	0.1	0.2	0.3	0.4	0.5
$x(t)$	1	0	-1	0	1
$y(t)$	0	1	0	-1	0

   a) Définir deux listes **`LX`** et **`LY`** contenant les abscisses et les ordonnées du point $M(t)$ aux divers instants t.

   b) Représenter les points aux divers instants t ainsi que la ligne polygonale joignant ces points.

   c) Calculer la longueur de cette ligne polygonale.

2. Les coordonnées du point $M(t)$ sont maintenant stockées dans un fichier CSV (*Comma-separated values*) nommé **`num004-points.csv`** situé dans le répertoire **`data`**, chaque ligne de ce fichier est constituée des données « **`t,x(t),y(t)`** » associées à un point $M(t)$.

   a) À partir de ce fichier, définir trois listes **`LT`**, **`LX`** et **`LY`** contenant les instants t, les abscisses et les ordonnées du point $M(t)$.

   b) Vérifier que la liste **`LT`** est bien ordonnée selon ses valeurs croissantes.

   c) Représenter les points aux divers instants t ainsi que la ligne polygonale joignant ces points.

   d) Calculer la longueur de cette ligne polygonale.

   e) Quelle est la vitesse moyenne sur l'ensemble du parcours ?

2021.19 – **Exercice à dominante simulation numérique**

Cet exercice est prévu pour le langage Python (et ses bibliothèques numpy, scipy, matplotlib). À chaque question, les instructions ou les fonctions écrites devront être testées.

---

1. Observer et expliquer ce que font les instructions suivantes :

```
>>> from numpy.random import rand
>>> LR = rand(5,3,3)
>>> LR[3]
>>> LR<0.5
>>> 1*(LR<0.5)
```

2. En déduire la proportion de matrices inversibles dans une liste de $N = 10\,000$ matrices carrées (3×3) dont chaque coefficient est tiré au hasard selon une loi de Bernoulli de paramètre $1/2$ (valant 0 ou 1). On pourra utiliser l'instruction **`det`** du module **`numpy.linalg`** de *Python*.

3. Écrire une fonction **`listes`** récursive d'argument n renvoyant la liste de toutes les listes de longueur n dont les élements valent 0 ou 1.

4. Utiliser cette fonction pour générer toutes les matrices carrées d'ordre 3 à coefficients dans $\{0,1\}$, et calculer la proportion de matrices inversibles parmi ces matrices. Comparer cette proportion avec la fréquence mesurée à la question 2.

5. Quelles sont les valeurs possibles du déterminant d'une matrice carrée d'ordre 3 à coefficients dans $\{0,1\}$? Établir alors la loi de probabilité du déterminant d'une matrice carrée d'ordre 3 à coefficients dans $\{0,1\}$. Comparer avec les fréquences obtenues par la simulation de la question 2.

2021.20 – **Exercice à dominante simulation numérique**

Cet exercice est prévu pour le langage Python (et ses bibliothèques numpy, scipy, matplotlib). À chaque question, les instructions ou les fonctions écrites devront être testées.

---

1. Faire tracer les courbes des fonctions $x \mapsto x\,\log(x) - 2$, où « log » désigne le logarithme népérien, $x \mapsto x^3 - 7\,x^2 + 11\,x - 1$ et $x \mapsto 0$ sur l'intervalle $]0, 5]$.

2. La méthode de Newton permet de trouver numériquement un zéro d'une fonction g en calculant les premiers termes d'une suite $(x_n)_{n\in\mathbb{N}}$ définie par $x_0 = d$ et $x_{n+1} = \varphi(x_n)$. Rappeler l'expression de la fonction φ en fonction de la variable, de g et de sa dérivée g'.

3. Déterminer la suite de valeurs $[x_0, x_1, \cdots, x_{20}]$ lorsque $x_0 = 3$ et $g(x) = x\,\log(x) - 2$. À partir de quel indice cette suite est-elle numériquement stationnaire ?

4. Recommencer pour $x_0 = 4$ et $g(x) = x^3 - 7\,x^2 + 11\,x - 1$.

5. Écrire une fonction **`indice`** de deux arguments **`phi`** et **`x0`** qui renvoie l'indice à partir duquel la suite $(x_n)_{n\in\mathbb{N}}$ est numériquement stationnaire.

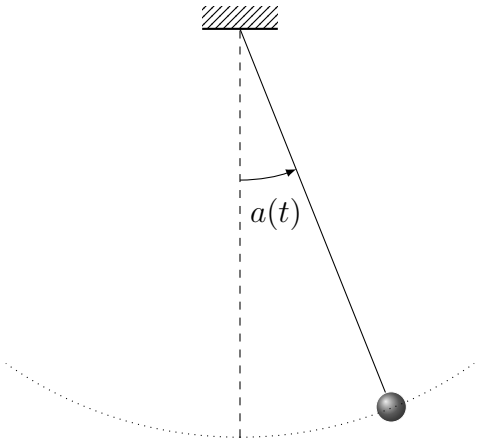
6. On va maintenant utiliser la fonction **`fsolve`** du module **`scipy.optimize`** de *Python*. Lire l'aide sur cette fonction et expliquer à quoi correspondent les deux premiers arguments de cette fonction.

   Utiliser **`fsolve`** pour résoudre à nouveau les deux équations précédentes.

2021.21 – **Exercice à dominante simulation numérique**

Cet exercice est prévu pour le langage Python (et ses bibliothèques numpy, scipy, matplotlib). À chaque question, les instructions ou les fonctions écrites devront être testées.

Soit un pendule amorti, repéré par son angle $a(t)$ avec la verticale, fonction du temps t exprimé en secondes (s) :



On suppose que la tige est de longueur constante, que le pendule est initialement en position verticale et qu'on le lance avec une vitesse angulaire initiale w_0. L'angle $a(t)$ satisfait alors l'équation différentielle :

$$\begin{cases} a''(t) = -m\,\sin(a(t)) - f\,a'(t) \\ a(0) = 0 \\ a'(0) = w_0 \end{cases} \tag{1}$$

On fixe $m = 1.1\,\mathrm{rad{\cdot}s^{-2}}$ et $f = 0.8\,\mathrm{rad{\cdot}s^{-1}}$.

1. Au brouillon, déterminer une fonction $\phi : \mathbb{R}^2 \to \mathbb{R}^2$ tel que, si l'on pose $\mathbf{u}(t) = (a(t), a'(t))$, le problème (1) soit équivalent à :

$$\mathbf{u}'(t) = \phi(\mathbf{u}(t))\ , \quad \text{avec la condition initiale } \mathbf{u}(0) = (\texttt{a0}, \texttt{w0})\ . \tag{2}$$

2. En utilisant la fonction **`odeint`** du module **`scipy.integrate`** de *Python*, résoudre numériquement le problème (2) pour t dans l'intervalle $[0, 25]$, la vitesse angulaire initiale w_0 valant successivement 1, 2, 4 et $8\,\mathrm{rad{\cdot}s^{-1}}$. On prendra garde à définir soigneusement les arguments de cette fonction en lisant attentivement l'aide en ligne.

3. Faire tracer ces quatres courbes, avec en abscisses l'instant t en secondes et en ordonnées, l'angle $a(t)$ en degrés.

4. Comment expliquer physiquement la différence de comportement asymptotique des solutions obtenues ?

5. Déterminer une valeur approchée à $10^{-4}\,\mathrm{rad{\cdot}s^{-1}}$ près de la valeur de $w_0 \in [2, 4]$ pour laquelle ce changement de comportement asymptotique a lieu. On pourra faire une recherche de type « *dichotomie* ».

2021.22 – **Exercice à dominante simulation numérique**

Cet exercice est prévu pour le langage Python (et ses bibliothèques numpy, scipy, matplotlib). À chaque question, les instructions ou les fonctions écrites devront être testées.

Pour les entiers n et p tels que $n \geqslant 2$ et $p \geqslant 1$, on considère la matrice carrée d'ordre n dont les coefficients entiers s'écrivent $((i-1)\,n+j)^p$ (ligne i, colonne j) :

$$M(n,p) = \begin{pmatrix} 1^p & 2^p & \cdots & (n-1)^p & n^p \\ (n+1)^p & (n+2)^p & \cdots & (n+(n-1))^p & (2\,n)^p \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ ((n-1)n+1)^p & ((n-1)n+2)^p & \cdots & \cdots & (n^2)^p \end{pmatrix}$$

On cherche à étudier numériquement la conjecture suivante : *Pour tout entier naturel non nul p, il existe un entier n au moins égal à 2 tel que* $\det(M(n,p)) = 0$.

1. Écrire une fonction M de deux arguments n et p, renvoyant la matrice $M(n,p)$. Faire afficher $M(4,1)$, $M(4,2)$ et $M(4,3)$.

2. Grâce à la fonction **`det`** du module **`scipy.linalg`** de *Python*, essayer de calculer le déterminant de $M(20,12)$. Grâce à la fonction **`eigvals`** du module **`scipy.linalg`** de *Python*, obtenir la liste des valeurs propres de $M(20,12)$. Commenter.

3. Écrire une fonction N de deux arguments n et p, renvoyant la matrice carrée $N(n,p)$ d'ordre n dont les coefficients s'écrivent $((i-1+j/n)\,/i)^p$ (ligne i, colonne j).

4. Au brouillon, exprimer le déterminant de $N(n,p)$ en fonction de celui de $M(n,p)$, entier.

5. Définir une fonction **`ordreNum`** d'argument p qui renvoie le premier nombre entier n tel que la plus petite en module des valeurs propres de $N(n,p)$ puisse être considérée comme nulle, c'est-à-dire de module inférieur à 10^{-15} fois le plus grand des modules. Tester **`ordreNum`** pour p de 1 à 20.

6. À l'aide du type **`Matrix`** et de la fonction **`det`** du module **`sympy.matrices`** de *Python*, qui permettent de faire du calcul exact et non pas numérique, vérifier que le plus petit entier n tel que $\det(M(n,p)) = 0$ vaut $n = p+2$ pour p de 1 à 20.

2021.23 – **Exercice à dominante simulation numérique**

Cet exercice est prévu pour le langage Python (et ses bibliothèques numpy, scipy, matplotlib). À chaque question, les instructions ou les fonctions écrites devront être testées.

On veut étudier numériquement la courbe Γ décrite par le point $M(t)$ de coordonnées $(\rho(t)\cos(t), \rho(t)\sin(t))$, où $\rho(t) = \sqrt{\mathbb{h}(\cos(2\,t))\,\cos(2\,t)}$ avec $\mathbb{h}$ la fonction qui vaut 1 sur $\mathbb{R}_+$ et 0 sur $\mathbb{R}_-^\star$.

1. Écrire la fonction **`rho`** d'argument t qui renvoie $\sqrt{\mathbb{h}(\cos(2\,t))\,\cos(2\,t)}$.

2. Écrire une fonction **`pts`** d'argument un entier naturel non nul n qui renvoie la liste des couples (x_i, y_i) représentant les coordonnées cartésiennes du point $M(t_i)$, sachant que t_i est la i-ème des $n{+}1$ valeurs résultant de la partition de l'intervalle $[0, 2\,\pi]$ en n intervalles de même longueur.

3. Faire tracer en repère orthonormé et sur un même graphique la ligne polygonale reliant les points de **`pts(`**n**`)`**, pour $n = 50$ et $n = 1000$.

4. Écrire une fonction **`longueur`** d'argument une liste de couples (x_i, y_i) qui renvoie la longueur de la ligne polygonale reliant les points correspondants. Évaluer le longueur de Γ avec 5 chiffres significatifs.

5. Écrire une fonction **`derivee`** d'argument une liste de couples $\mathbf{p}_i = (x_i, y_i)$ qui renvoie une autre liste de couples $\mathbf{p}'_i = (x'_i, y'_i)$ de même longueur $n{+}1$ telle que $\mathbf{p}'_i = 1/(2\,\delta t)\ (\mathbf{p}_{i+1} - \mathbf{p}_{i-1})$, avec $\delta t = 2\,\pi/n$. On remplacera les points manquants grâce à des considérations de périodicité.

   Faire tracer les courbes correspondant à **`derivee(pts(`**n**`))`**, pour $n = 50$, $n = 200$ et $n = 1000$.

2021.24 – **Exercice à dominante simulation numérique**

Cet exercice est prévu pour le langage Python (et ses bibliothèques numpy, scipy, matplotlib). À chaque question, les instructions ou les fonctions écrites devront être testées.

---

1. Écrire une fonction **`trapezes1`** de quatre arguments, f, a, b et n, qui renvoie une valeur approchée de l'intégrale de f sur l'intervalle $[a, b]$, calculée par la méthode des trapèzes en subdivisant le segment $[a, b]$ en n intervalles de même longueur. On suppose que f est une fonction à valeurs réelles continue sur le segment $[a, b]$, que $a < b$ et que $n \geqslant 2$.

   On pourra utiliser la fonction **`linspace`** du module **`numpy`** de *Python*, et également considérer que « *la fonction f est vectorisée* », c'est-à-dire que **`f(u)`** où **`u`** est un vecteur de composantes u_k donne directement le vecteur de composantes **`f(`**u_k**`)`**.

2. Vérifier que **`trapezes1`** fonctionne en calculant numériquement $\displaystyle\int_0^{\pi} \sin(t)\,\mathrm{d}t$. En *Python*, on utilisera les constantes et fonctions mathématiques usuelles de **`numpy`** pour bénéficier de la vectorisation.

3. Écrire une fonction **`trapezes2`** qui améliore la fonction **`trapezes1`** en renvoyant, en plus de la valeur approchée de l'intégrale, une valeur estimée ε d'un majorant de l'erreur numérique :

$$\varepsilon = \frac{b-a}{12} \max_{1 \leqslant k \leqslant n-1} |f(t_{k-1}) - 2\,f(t_k) + f(t_{k+1})|$$

   où l'intervalle $[a, b]$ et discrétisé en $(n+1)$ valeurs t_0, t_1, $\cdots$, t_n régulièrement espacées.

4. On veut appliquer la méthode des trapèzes au calcul de $\displaystyle\int_0^{+\infty} \frac{\exp(-t)}{1+t^2}\,\mathrm{d}t$.

   On peut démontrer que que pour tout réel stritement positif T :

$$0 < \int_0^{+\infty} \frac{\exp(-t)}{1+t^2}\,\mathrm{d}t - \int_0^{T} \frac{\exp(-t)}{1+t^2}\,\mathrm{d}t < \frac{\exp(-T)}{1+T^2} \quad .$$

   faire tracer la courbe de la fonction $t \mapsto \exp(-t)/(1+t^2)$ sur l'intervalle $[0, 4]$ puis déterminer la plus petite valeur $T = i/10$, où i est entier, telle que $\dfrac{\exp(-T)}{1+T^2} < 5{\times}10^{-6}$.

5. Déduire de ce qui précède une valeur approchée de $\displaystyle\int_0^{+\infty} \frac{\exp(-t)}{1+t^2}\,\mathrm{d}t$ à 10^{-5} près.

   On utilisera la fonction **`trapezes2`** en commençant par n=2 et en doublant à chaque étape la valeur de n, jusqu'à obtenir la précision souhaitée.

6. Comparer la valeur numérique de l'intégrale avec le résultat obtenu en utilisant la fonction **`quad`** du module **`scipy.integrate`** de *Python*.