

Épreuve orale de « *Mathématiques et algorithmique* » de la Banque PT – Rapport 2024

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. Même si les programmes ont changé depuis la session 2023 de la Banque PT, notamment avec l'introduction des *dictionnaires*, de la *programmation dynamique* et de la *manipulation des graphes*, et des exigences moindres en simulation numérique, l'esprit de l'épreuve reste le même avec un fort niveau d'exigence sur les savoirs et savoir-faire de base. Nous suggérons aux futurs candidats de consulter le [site de la Banque PT](#), où ils trouveront le mémento *Python* fourni lors de l'oral ainsi que les exercices types d'informatique, comme 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 (les semestres 1 et 2 ainsi que le paragraphe 3.2 du semestre 3) 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.

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 semestres 1 et 2 ainsi que le paragraphe 3.2 du semestre 3 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* et ses principales bibliothèques (dont **numpy**, **scipy**, **matplotlib**, **random**, aides incluses), 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** 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). Nous ne pouvons que conseiller de se placer dans les conditions de passage de l'oral tout au long des deux années de préparation, en installant par exemple l'environnement virtuel Python 3 dédié, construit avec la distribution Miniconda (voir [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 – À propos de l'oral 2024

Cette dernière session s'est déroulée dans les locaux de l'*École Nationale d'Arts et Métiers*, 155 boulevard de l'Hôpital, Paris (13^e). Comme lors des sessions précédentes, la plupart des candidats semblaient bien préparés à cette épreuve. Cependant, nous avons remarqué comme en 2023 des **lacunes importantes en mathématiques sur les calculs, même très simples**. Les candidats doivent être à l'aise avec des développements ou simplifications d'expressions algébriques, calculs de déterminants, de dérivées, d'intégrales etc. On voit trop souvent des candidats passer une partie importante de l'oral sur un calcul simple dans une question qui n'aboutit au résultat qu'après de nombreuses erreurs du candidat et indications de l'examineur.

4 – Conseils généraux

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

- Lire attentivement le sujet et bien écouter une question dans le détail permet de répondre à la question effectivement posée ; même si c'est de moins en moins dans l'air du temps, cette exigence de précision est indispensable ; il ne sert à rien de se précipiter dans un calcul ou l'écriture d'un code sans s'être assuré d'avoir lu et compris l'intégralité de la question, éventuellement en demandant une confirmation à l'interrogateur.
- 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. 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 commentaire ; 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) et les coordonnées de son milieu, 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 \vec{AB} et \vec{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 .

5 – Conseils pour l'exercice de mathématiques

5.1 – 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 triviale, **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 au programme.
- 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 devra se poser la question : « *la méthode que je veux mettre en œuvre est-elle bien adaptée au problème que je veux résoudre ? En particulier, les hypothèses nécessaires sont-elles bien satisfaites ?* » ; par exemple, lorsque j'écris l'équation caractéristique pour une suite récurrente ou une équation différentielle, suis-je bien dans le cas d'un problème linéaire à coefficients constants ? De même, lorsque je veux étudier la convergence d'une suite ou d'une série, ai-je bien recensé ses propriétés élémentaires (positivité, monotonie, type répertorié ou non, etc.) avant de choisir telle ou telle méthode (majoration par une suite ou série convergente, minoration par une suite ou série divergente, critère de d'Alembert, etc.) ? Ou encore, lorsqu'on me demande d'étudier la continuité ou la dérivabilité de $x \mapsto \int_0^x f(t) dt$ ou de $\theta \mapsto \int_{-\cos(\theta)}^{\cos(\theta)} f(t) dt$, suis-je bien dans le cas d'application des théorèmes sur les intégrales à paramètre ?
- On attend par conséquent d'un candidat qu'il soit capable d'identifier et de décrire précisément le type de problème à résoudre.
- Et ensuite qu'il maîtrise les techniques de calcul adaptées 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.

- 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, ou de confondre équation cartésienne et représentation paramétrique.

5.2 – Algèbre linéaire

- En algèbre comme ailleurs, on doit veiller à utiliser un vocabulaire précis et à éviter les confusions. Nous avons pu déplorer trop souvent une confusion incompréhensible entre matrice inversible et matrice diagonalisable.
- 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 $\dim(\ker(A)) \geq 1$ 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 ».
- Rappelons également que la détermination des valeurs propres d'une matrice triangulaire ne nécessite pas le calcul du polynôme caractéristique.
- Le théorème du rang est souvent cité de manière approximative.
- Les applications linéaires d'un espace autre que \mathbb{R}^n (espaces de matrices, de polynômes ou de fonctions) posent des difficultés aux candidats.
- 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.
- Rappelons enfin que la notation \mathcal{A}^\top pour la transposée de la matrice \mathcal{A} est la seule admise dans le cadre des nouveaux programmes. Elle permet de caractériser simplement la notion de matrice symétrique avec la relation $\mathcal{A}^\top = \mathcal{A}$.

5.3 – 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 $\lim_{x \rightarrow a} f(g(x)) = f\left(\lim_{x \rightarrow a} g(x)\right)$ doit être justifiée clairement, même si la fonction f est une fonction usuelle.
- Les candidats ne pensent pas à utiliser les théorèmes des valeurs intermédiaires et de la bijection et connaissent mal les hypothèses.

5.4 – 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.

- Pour montrer que deux intégrales sont égales, l'intégration par parties n'est pas systématique ; il faut penser aussi à des changements de variables simples du type $x = \pi/2 - t$ ou $x = 1/t$.
- De trop nombreux candidats mélangent le *Théorème fondamental du calcul intégral* et les théorèmes sur les intégrales dépendant d'un paramètre.
- De plus en plus de candidats ne connaissent pas les sommes de Riemann, qui sont très utiles dans de nombreux exercices.

5.5 – 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.
- Les séries géométriques doivent être parfaitement maîtrisées, ce qui n'est pas toujours le cas. Leur somme ainsi que leur somme partielle doivent être connues. Il faut aussi penser à les utiliser lorsque la raison est dans \mathbb{C} , comme par exemple pour calculer $\sum_{k=1}^n e^{ik\theta}$.
- Le critère de d'Alembert ne fonctionne pas toujours ; il doit parfois être adapté intelligemment, par exemple pour les séries où les termes de rangs pairs (ou impairs) sont tous nuls.
- L'écriture $\lim_{n \rightarrow \infty} f(u_n) = f\left(\lim_{n \rightarrow \infty} u_n\right)$ doit être justifiée clairement, même si la fonction f est une fonction usuelle.
- La recherche de solution développable en série entière d'une équation différentielle fait partie des attendus de cette épreuve.

5.6 – 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'examineur 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} .

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

Pour la géométrie dans le plan et dans l'espace, la distinction entre équation cartésienne et représentation paramétrique doit être claire pour tout candidat, ainsi que le passage aux éléments géométriques de la

courbe ou de la surface (vecteur directeur, vecteur normal, droite et plan tangents, etc.). Cela s'applique en particulier aux éléments géométriques de base que sont les droites, les cercles, les ellipses, les plans, les cylindres ou les sphères. Nous avons remarqué cette année que ces compétences de géométrie de base font de plus en plus défaut. 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 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 de représentation paramétrique.

5.8 – 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, ainsi que la méthode de la variation de la constante pour trouver une solution particulière non triviale.
- Les solutions des équations différentielles $y''(t) \pm y(t) = 0$ devraient être connues et ne nécessitent pas le calcul de l'équation caractéristique.

5.9 – Probabilités

- Encore plus qu'ailleurs, il faut lire attentivement l'énoncé et être précis dans son vocabulaire ; un minimum de formalisme est attendu.
- Les lois de probabilités usuelles (uniforme, Bernoulli, géométrique, binomiale, Poisson) et leurs caractéristiques sont connues mais pas toujours de manière précise.
- 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. Savoir prononcer le terme « *système complet d'évènements* » ou « *formule des probabilités totales* » est bien, mais il est nettement mieux d'être en mesure de détailler de quoi il s'agit.

6 – Conseils pour l'exercice d'algorithmique

Les candidats sont en général bien formés et le nombre d'excellents candidats est toujours en augmentation. Cependant, quelques candidats étaient cette année en grande difficulté, sachant à peine faire la distinction entre la console et l'éditeur dans l'environnement **Idlex**. 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.

6.1 – 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 intrinsèques `sum`, `max`, `min`, `sorted` entre autres, 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.
- Lorsqu'un candidat crée ou manipule un objet élémentaire, il doit être capable de préciser son type (entier, flottant, complexe, booléen, chaîne de caractères, liste, tuple, vecteur, matrice, etc.), les opérations et fonctions spécifiques qui peuvent lui être appliquées, et pouvoir justifier son choix de type d'objet ; les confusions *entier-flottant*, *liste-vecteur*, et *liste de listes-matrice* sont encore trop fréquentes.
- En règle générale, le travail demandé est à écrire dans l'éditeur de programme, qu'il s'agisse de définir puis de tester une fonction, de créer directement des objets ou de faire tracer une courbe ou un nuage de points ; quelques candidats n'écrivaient dans leur programme que des fonctions, même lorsque ce n'était pas demandé, et tout le reste dans la console, ce qui montre une méconnaissance de ce qu'est réellement un programme informatique.
- Ne pas hésiter cependant à utiliser la console (l'interpréteur) pour effectuer des vérifications ou des tests complémentaires, contrôler la nature et l'état d'un objet, ou consulter l'aide.
- 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.
- Il est indispensable de savoir utiliser l'instruction `help` : 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. Cela fait partie des compétences évaluées.
- Il faut savoir mettre en œuvre une démarche en cas d'erreur : faire des tests élémentaires dans la console, insérer des `print` dans un programme pour contrôler pas à pas son 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.
- 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 ; le quotient `//` et le reste `%` de la division euclidienne sont en général bien maîtrisés, même si la confusion avec la division flottante / subsiste parfois.
- Le traitement 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), en particulier lorsqu'il s'agit du nom ou du chemin d'un fichier ; la connaissance des méthodes `split`, `strip`, `replace`

peut s'avérer utile pour la lecture de données structurées dans un fichier ASCII ; nous avons pu déplorer trop souvent cette année le mauvais réflexe de convertir une chaîne en liste dans l'idée de se ramener à un type mieux connu, sans avoir conscience des complications que cela peut engendrer.

- 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 (`nom[a:b]`) et la concaténation des chaînes de caractères comme des listes permettent de gagner du temps, 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, `nom[:-3]` pour tout prendre sauf les trois derniers, etc.).

Attention – Une erreur trop fréquemment rencontrée dans le cas d'un tableau `T` à deux indices : les écritures `T[:,j]` et `T[:,j]` ne sont absolument pas équivalentes, contrairement à `T[i,j]` (écriture compatible avec une représentation en liste de listes) et `T[i,j]` ; écrire `T[:,j]` revient à extraire la ligne d'indice `j` alors que `T[:,j]` permet d'extraire la colonne d'indice `j`.

- La vérification de la conformité des paramètres d'une fonction (par `assert` et/ou des tests) ainsi que le rajout d'un `docstring` ne sont en général pas demandés et font perdre un temps précieux dans le cadre de cet oral en temps limité, même s'ils peuvent être dans un tout autre contexte légitimement préconisés (développement logiciel).
- 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` 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).
- Dans le cadre des nouveaux programmes, où des exigences moindres en simulation numérique sont demandées, **il sera toujours nécessaire de savoir tracer la courbe représentative d'une fonction sur un intervalle, une courbe paramétrée ou un nuage de points**. Cette compétence était mieux maîtrisée comparé aux années précédentes. Si l'on souhaite par exemple tracer la courbe représentative de $x \mapsto \sin(x^2)$ sur l'intervalle $[0, 5]$ avec 200 points on pourra écrire :

```
def f(x): return np.sin(x**2)
Vx = np.linspace(0,5,200)
plt.plot(Vx,[f(t) for t in Vx])
```

6.2 – 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 techniques de *slicing* peuvent être utilisées : « `U[debut : fin : pas]` » pour une séquence (liste, chaîne de caractères, etc.).
- 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 :

<pre>L = [] for i in range(9) : L.append(i)</pre>	ou	<pre>L = [i for i in range(9)]</pre>	au lieu de	<pre>L = list(range(9))</pre>
---	----	--	------------	-------------------------------

<pre>L = [] for x in np.linspace(-1.2, 3.2, 441) : L.append(x) V = np.array(L)</pre>	au lieu de	<pre>V = np.linspace(-1.2, 3.2, 441)</pre>
--	------------	--

- Même si les listes en compréhension ne sont pas exigibles, leur utilisation maîtrisée permet de gagner en efficacité et en lisibilité ; de nombreux candidats les ont utilisées.
- Certains candidats utilisent la fonction **zeros** du module **numpy** avec des arguments erronés : **np.zeros(n)** crée un tableau unidimensionnel de taille **n** tandis que **np.zeros((n,n))** permet de créer un tableau bidimensionnel de taille **n** dans chaque direction (très utilisé pour initialiser une matrice).
- 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.
- 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 :

<pre>B = test(a,b)</pre>	et non pas	<pre>if test(a,b) == True : B = True else : B = False</pre>
--------------------------	------------	---

De même, si **B** désigne un booléen, on privilégiera l'écriture « **not B** » à « **B == False** » ou encore « **B != True** ».

6.3 – Algorithmique

Des thématiques autour de la manipulation de graphes et de la programmation dynamique ont été introduites dans les nouveaux programmes depuis la session 2023 et les candidats ont été interrogés sur ces thèmes cette année. Le niveau de connaissances sur les graphes était très hétérogène : certains candidats maîtrisaient les notions et les algorithmes du programme, d'autres ne connaissaient pas le vocabulaire de base (arête, sommet, chemin, etc.). Concernant les exercices sur le thème de la programmation dynamique, il est nécessaire de faire la distinction entre *algorithme récursif* et *algorithme itératif*, ce qui n'est hélas pas

toujours le cas. Dans l'écriture d'une fonction récursive, un soin particulier doit être porté à la condition d'arrêt. La banque d'exercices d'algorithmique continuera à être alimentée sur ces nouveaux thèmes dans les futures sessions, il est donc important de bien s'y préparer (nous invitons les candidats à consulter les exercices types d'algorithmique qui sont disponibles sur le [site de la Banque PT](#)), ainsi que les exercices publiés dans le présent rapport.

- Avant toute chose, respecter impérativement le principe de base : « **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 ; ce défaut a pu être observé hélas chez des candidats qui par ailleurs avaient une bonne pratique de la programmation. Un exemple caricatural à ne surtout pas suivre, en supposant que la fonction `points` renvoie une liste de couples de coordonnées :

```
X, Y = [], []
for i in range( len( points(50, -1.2) ) ) :
    X.append( points(50, -1.2)[i][0] )
    Y.append( points(50, -1.2)[i][1] )
plt.plot(X, Y, "sr")
```

où l'on répète inutilement le même calcul de `points(50, -1.2)`, au lieu d'écrire plus simplement et en ne faisant qu'une seule fois le calcul :

```
P = points(50, -1.2)
plt.plot([p[0] for p in P], [p[1] for p in P], "sr")
```

ou encore, sans utiliser de liste en compréhension :

```
P = points(50, -1.2)
X, Y = [], []
for xi,yi in P :
    X.append( xi )
    Y.append( yi )
plt.plot(X, Y, "sr")
```

- Il faut maîtriser des algorithmes simples, comme par exemple : l'extraction d'éléments d'une liste ou d'une chaîne selon un critère donné ; la subdivision d'un intervalle $[a, b]$ en n sous-intervalles de même longueur ; l'extraction de toutes les sous-listes ou sous-chaînes de taille k ; l'extraction des éléments distincts d'un objet itérable ; la détermination du rang de la première répétition dans un objet itérable ; calcul de moyenne et extraction de fréquences d'apparition à partir d'une liste de valeurs tirées au hasard (module `random`) ; ou encore l'extraction à partir d'un entier de la liste de ses chiffres en écriture décimale.
- L'utilisation d'une boucle `while` nécessite de : s'assurer de sa terminaison, par l'évolution d'au moins une variable de boucle, en rajoutant éventuellement un contrôle fixant le nombre maximum d'itérations ; de bien faire la distinction entre la condition de poursuite de la boucle et la condition d'arrêt, sa négation ; et de penser enfin à reculer d'un cran en sortie de boucle dans les cas qui l'exigent. Nous avons pu déplorer chez quelques rares candidats un mélange étrange de `while` et de `for` sans queue ni tête.

- Il faut être à l'aise avec les différentes manières de représenter les graphes, orientés comme non orientés : matrice d'adjacence, listes d'adjacence, ou encore dictionnaire. Il convient aussi d'être familier avec les notations mathématiques concernant les graphes telles que le graphe $G = (S, A)$, l'arête $\{u, v\}$, l'arc (u, v) .
- Lorsqu'un énoncé d'algorithmique propose une formule de récurrence, comme c'est le cas dans la question 4 de l'exercice ALGO 2 en annexe, il faut savoir interpréter cette formule ainsi qu'écrire une fonction l'évaluant. Si l'énoncé ne le précise pas, le candidat a la liberté d'utiliser l'approche qui lui semble la plus appropriée, récursive ou itérative.

7 – Analyse des résultats

En 2024, 1603 candidats ont passé l'oral de « *Mathématiques et algorithmique* ». Chacun des 12 jours de l'oral, les 8 à 10 jurys se sont efforcés de poser des exercices balayant l'ensemble du programme, tant en mathématiques qu'en algorithmique. Ainsi, 173 exercices différents d'analyse et de probabilités ont été proposés à 820 candidats contre 166 exercices différents de géométrie et d'algèbre proposés à 783 candidats. De plus, 312 exercices différents d'algorithmique ont été posés à 1603 candidats.

Les statistiques sur les notes sont les suivantes (rappelons que seule la note globale est communiquée au candidat) :

Oral 2024	Note (sur 20)	Math. (sur 20)	Algo. (sur 20)
Moyenne	10,78	10,71	10,84
Écart-type	4,17	5,04	5,06
Minimum	1	0	0
Maximum	20	20	20

Thomas MILCENT et Roland GRAPPE, Coordonnateurs de l'épreuve orale de « *Mathématiques et algorithmique* » de la Banque PT, Le 12 juillet 2024.

8 – Exercices sortis de la banque

Nous proposons pour finir ce rapport de publier 3 exercices issus de la banque qui sont représentatifs de ce qui peut être posé aux candidats.

ALGO 1

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 **U** de deux arguments, un entier naturel N et un nombre a , qui renvoie le vecteur $(1^a, 2^a, \dots, N^a)$.
2. Faire tracer sur un même graphique n^a en fonction de n , pour n entier de 1 à 30 et pour a valant 0.0, -0.5 et -1.0 .
3. Observer dans la console ce que donnent les instructions :

```
>>> import numpy as np
>>> np.eye(4)
>>> np.diag([1,2,3],k=-1).
```

En déduire une fonction **M** d'argument un vecteur $V = (v_1, v_2, \dots, v_{n-1}, v_n)$ qui renvoie la matrice :

$$\begin{pmatrix} 1 & -v_1 & 0 & \cdots & 0 & 0 & 0 \\ v_1 & 1 & -v_2 & \ddots & \ddots & \ddots & \vdots \\ 0 & v_2 & \ddots & \ddots & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 & -v_{n-2} & 0 & \vdots \\ \vdots & \ddots & 0 & v_{n-2} & 1 & -v_{n-1} & 0 \\ \vdots & \ddots & \ddots & 0 & v_{n-1} & 1 & -v_n \\ 0 & 0 & \cdots & 0 & 0 & v_n & 1 \end{pmatrix}$$

En utilisant la fonction **det** du module **numpy.linalg**, faire afficher le déterminant de cette matrice pour $\mathbf{V}=\mathbf{U}(n,a)$, pour $a = -0.5$ et $a = -1.0$ et pour n prenant ses valeurs dans $\{10, 50, 200, 1000\}$.

4. Nous admettrons que si l'on pose $a_n = \det(\mathbf{M}((v_1, v_2, \dots, v_{n-1}, v_n)))$, la suite ainsi définie vérifie la récurrence :

$$a_0 = 1, \quad a_1 = 1 + v_1^2 \quad \text{et} \quad \forall n \in \mathbb{N}, \quad n \geq 2, \quad a_n = a_{n-1} + v_n^2 a_{n-2},$$

et cette suite est toujours croissante et à valeurs supérieures à 1.

En déduire une fonction **A** d'argument un vecteur $V = (v_1, v_2, \dots, v_{n-1}, v_n)$ qui renvoie la valeur de a_n . Vérifier que l'on retrouve bien numériquement les mêmes valeurs qu'à la question 3.

5. Lorsque pour tout entier naturel non nul n , $v_n = 1/\sqrt{n}$, déterminer le plus petit entier n_{\min} tel que, pour tout entier naturel n supérieur à n_{\min} , $a_n \geq 1000$.
Faire tracer a_n/n en fonction de n pour n variant de 1 à 10000. Quelle conjecture peut-on faire lorsque n tend vers l'infini ?
6. Vérifier numériquement sur les premières valeurs de n que :

- si $v_n = 1$ pour tout n , alors $a_n = \left(1 + \frac{2}{\sqrt{5}}\right) \left(\frac{1 + \sqrt{5}}{2}\right)^{n-1} + \left(1 - \frac{2}{\sqrt{5}}\right) \left(\frac{1 - \sqrt{5}}{2}\right)^{n-1}$;
- si $v_n = n$ pour tout n , alors $a_n = (n + 1)!$.

ALGO 2

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

On considère le problème de minimisation suivant : on dispose d'une séquence de n nombres réels a_0, \dots, a_{n-1} de signes quelconques, et l'on souhaite trouver, parmi toutes les sous-séquences de termes consécutifs de cette séquence, celle dont la somme est minimum.

La séquence en entrée sera représentée par une liste de nombres.

Entrée: $\mathbf{A} = [a_0, \dots, a_{n-1}]$

Sortie: $\mathbf{T} = \min_{0 \leq i \leq j \leq n-1} \sum_{k=i}^j a_k$

Par exemple, pour la séquence $[7, -5, 3, -4, 5]$, la sous-séquence de somme minimum est $[-5, 3, -4]$ et sa somme vaut $\mathbf{T}=-6$ (ce qui correspond à $i = 1$ et $j = 3$ dans l'expression ci-dessus).

1. Écrire une fonction **brutal** qui prend en entrée une liste de nombres \mathbf{A} et qui renvoie \mathbf{T} .
2. Discuter de la complexité de votre fonction **brutal**.

On souhaite résoudre ce problème en utilisant les graphes. Étant donnés $a_0, \dots, a_{n-1} \in \mathbb{R}$, soit G le graphe orienté pondéré défini par :

- $n + 2$ sommets : n sommets numérotés de 0 à $n - 1$, un sommet de départ s et un terminal t ,
- les arcs (s, k) de poids 0, pour $k = 0, \dots, n - 1$,
- les arcs (k, t) de poids a_k , pour $k = 0, \dots, n - 1$,
- les arcs $(k, k + 1)$ de poids a_k , pour $k = 0, \dots, n - 2$.

On notera $p(u, v)$ le poids d'un arc (u, v) .

3. Dans G , à quel problème de théorie des graphes correspond la recherche de \mathbf{T} ?
4. Pour tout sommet $v \in V$, on note $\ell(v)$ le poids d'un plus court chemin de s à v , avec $\ell(s) = 0$, et on admet la relation :

$$\ell(v) = \min \{ \ell(u) + p(u, v) : u \text{ prédécesseur de } v \}.$$

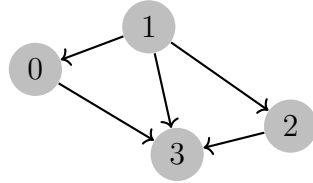
En utilisant cette relation, écrire une fonction **malin** qui prend en entrée une liste de nombres \mathbf{A} et qui renvoie \mathbf{T} .

5. Discuter de la complexité de votre approche.

ALGO 3

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

Dans un graphe orienté $G = (S, A)$ sans boucles, une *source universelle* est un sommet u tel que, pour tout $v \in S \setminus \{u\}$, on a $(u, v) \in A$ et $(v, u) \notin A$. Le but de cet exercice est la détection de sources universelles, selon la représentation du graphe.



Par exemple, le sommet 1 est une source universelle dans le graphe ci-dessus.

1. Peut-il exister plusieurs sources universelles ?
2. Écrire une fonction **source_mat** qui prend en entrée un graphe représenté par sa matrice d'adjacence et renvoie **True** si le graphe contient une source universelle, et **False** sinon.
3. Discuter de la complexité de votre fonction.
4. Écrire une fonction **source_list** qui prend en entrée un graphe représenté par listes d'adjacence et renvoie **True** si le graphe contient une source universelle, et **False** sinon.
5. Discuter de la complexité de votre fonction.

Un graphe orienté est un *océan* s'il contient une source universelle v et que $G \setminus v$ est encore un océan. Ici $G \setminus v$ est le graphe obtenu à partir de G en supprimant le sommet v ainsi que tous les arcs qui lui sont incidents.

6. Écrire une fonction **supprime** qui prend en entrée un graphe G , représenté de la manière la plus appropriée, et un sommet v de G , et qui renvoie le graphe $G \setminus v$.
7. Écrire une fonction **ocean** qui prend en entrée un graphe, représenté de la manière la plus appropriée, et qui renvoie **True** si le graphe est un océan, et **False** sinon.
8. Combien de graphes orientés sont des océans ?
Proposer une autre méthode pour tester si un graphe est un océan.