18. Mathématiques-informatique

18.1. Introduction

L'épreuve de mathématiques-informatique est une épreuve de mathématiques utilisant l'outil informatique.

Un ordinateur équipé des environnements de développement Pyzo et Spyder est mis à disposition du candidat. Des fiches d'aide présentant différentes fonctions Python pouvant être utiles sont fournies lors de l'épreuve sous forme papier, ainsi que sous forme d'un fichier pdf présent sur l'ordinateur. Ces fiches sont consultables en ligne sur le site du concours.

Le candidat dispose d'une préparation d'une demi-heure, puis est interrogé pendant 25 minutes environ. L'outil informatique peut être employé, entre autres, pour effectuer des calculs numériques (comme déterminer les premiers termes d'une suite de scalaires) ou matriciels (comme résoudre un système linéaire ou rechercher les éléments propres d'une matrice), des tracés de courbes ou de surfaces ou pour simuler une expérience aléatoire. L'objectif visé est principalement l'émission de conjectures pertinentes. Les énoncés sont calibrés afin de ne pas nécessiter de dextérité déraisonnable en programmation.

Dans cette épreuve, on évalue la capacité du candidat à aborder de manière constructive les notions du programme de mathématiques de la filière PSI. La capacité à s'exprimer et la rigueur de la démarche sont aussi prises en compte dans la notation.

18.2. Analyse globale des résultats

La majorité des candidats a compris le principe de l'épreuve et beaucoup ont pris la peine de se familiariser avec les fiches d'aide disponibles.

Le jury est relativement satisfait des performances des candidats : la moyenne sur l'épreuve est d'environ 11,53 avec un écart type de 3,28. La majorité des candidats a été capable – parfois avec un peu d'aide – de répondre à l'étude numérique proposée et apporter des éléments de preuve mathématique, certains candidats le faisant de manière très autonome. Les excellentes prestations se font cependant plus rares, ainsi que fort heureusement celles complètement catastrophiques.

Notons que le réflexe de tester ses codes informatiques n'est curieusement pas du tout systématique et une très grande partie des candidats n'a pas le réflexe de faire les codes en plusieurs morceaux pour les valider les uns après les autres. Les moins habiles ne savent pas exécuter d'instructions dans la console, ou ignorent qu'il faut fermer la fenêtre graphique avant de relancer l'exécution de leur code.

Il est très rare que l'étudiant soit mutique. En revanche, le jury regrette que de nombreux candidats se contentent de proposer des pistes pour répondre aux questions et n'entament une rédaction qu'à partir du moment ou l'examinateur valide une de ces propositions. Un peu plus d'initiative de la part de ces candidats est attendue.

18.3. Commentaires sur les réponses apportées et conseils aux candidats

Nous allons donner quelques conseils aux étudiants qui préparent le concours Centrale-Supélec. Les futurs candidats sont également invités à lire les rapports précédents dont le contenu reste valable et n'est pas strictement identique à ce qui suit.

Pour bien préparer l'épreuve, il faut tout d'abord travailler le cours, celui de seconde année, comme celui de première année trop souvent oublié, puis les techniques usuelles. Quiconque connaît son cours et sait comment aborder les problèmes classiques est assuré d'avoir une note fort convenable. Toutes les notions du cours de seconde année de PSI, mais aussi du cours de première année (intersection entre les programmes de MPSI et de PCSI), doivent être connues. Les exercices ont été spécifiquement préparés pour la filière PSI et ne demandent pas de connaissances hors programme.

La gestion de la préparation est un point important de l'oral :

- certains candidats s'imaginent devoir passer la totalité de celle-ci à coder et n'abordent pas les questions théoriques. D'autres, au contraire, négligent les codes et tentent de traiter un maximum de questions pour masquer leur manque de maîtrise de Python. Précisons donc que l'immense majorité des énoncés placent les questions de Python dès le début du sujet. Il est recommandé de ne pas dépasser la moitié du temps de préparation sur les codes, quitte à revenir dessus par la suite pendant le passage, l'oral restant une épreuve de mathématiques avant tout ;
- un soin particulier doit être apporté à la qualité des réponses pendant la préparation. Il est fréquent de constater que les candidats négligent les premières questions, proposent des réponses fausses ou mal argumentées, puis se décomposent quand on leur fait remarquer leur erreur ou lors d'une demande de précision. De nombreuses contestations ont pour origine le fait qu'un candidat s'estime lésé parce qu'il n'a pas pu présenter ce qu'il avait fait aux questions de milieu de sujet, justement parce que l'examinateur a décidé d'insister sur les nombreuses erreurs en début d'épreuve. Il convient donc de rappeler :
  - que le jury décide seul de l'ordre de ses questions ;
  - que la meilleure manière d'éviter une telle déconvenue est de ne pas chercher à aller trop vite, mais au contraire à veiller à la qualité de ses arguments plutôt qu'au nombre de questions abordées : cet oral n'est en aucun cas une épreuve de vitesse.

En début d'épreuve, la lecture, la copie quasi intégrale au tableau de l'énoncé, la présentation générale à l'oral du sujet constituent une perte de temps ; les membres du jury interrogent toujours en ayant l'énoncé de l'exercice et le candidat est invité à entrer d'emblée dans le vif du sujet.

Il est attendu d'un admissible qu'il fasse preuve de rigueur et de précision. Le recours très fréquent à des expressions comme « c'est continu » ou « ça converge » est à bannir au profit de phrases comportant un sujet précis : la fonction est continue, la suite converge ou la série converge. Attention à ne pas tomber dans l'excès inverse et à donner trop d'éléments hors sujet, voire absurdes. Lors des études de suites récurrentes de la forme $u_{n+1} = f(u_n)$, on a déjà vu des candidats affirmer que la suite $(u_n)_{n\in N}$ était croissante sur son intervalle de définition. La lucidité étant essentielle, il faut savoir faire le lien entre les résultats théoriques et ceux constatés sur l'ordinateur et ne pas tenter de justifier des propriétés en contraste évident avec ceux observées par la machine.

Utilisation du logiciel

Dans l'ensemble, la syntaxe de base du langage Python est bien maîtrisée, ainsi que les rudiments d'algorithmique nécessaires pour l'épreuve, ce qui est un point positif. C'est moins vrai pour l'utilisation des outils d'ingénierie numérique. Sur ce point, le jury avait pour consigne d'être particulièrement disposé à faire preuve de pédagogie. Les conseils des années précédentes n'ont pas pris une ride et sont reformulés ci-après.

- Il convient de se familiariser avec l'environnement Pyzo ou Spyder avant de passer l'épreuve : télécharger le logiciel, repérer où sont l'éditeur et la console, comment les utiliser, être à même de n'exécuter qu'une partie de son script pour corriger une erreur ou obtenir de nouveaux résultats, savoir faire des allers-retours dans l'emploi de l'éditeur et de la console (l'usage de `print` n'est pas une fatalité).
- Il est recommandé d'être plus vigilant aux messages renvoyés par le logiciel lors de l'exécution d'un script : ils peuvent permettre de corriger de nombreuses erreurs. Il convient de prêter une attention toute particulière aux parenthésages : de nombreux candidats demeurent démunis devant une syntaxe invalide ou ne comprennent pas les messages « `not subscriptable` » et « `not callable` ». Il faut faire attention à ne pas commettre de fautes de frappe dans les imports si l'on emploie ceux mentionnés dans l'aide.
- Les feuilles d'aide sont disponibles sur le site du concours et peuvent permettre tout au long de l'année de préparation d'illustrer de manière concrète le cours de mathématiques. La différence est nette entre les candidats connaissant les fiches d'aide proposées par le concours et ceux les découvrant pendant la demi-heure de préparation.
- Il faut être vigilant sur les bornes dans les `range`, sur les initialisations des variables avant les boucles, ainsi que sur les terminaisons des boucles `while`. Il faut aussi faire attention aux indentations et à la façon de tester une égalité. D'une manière générale, les candidats doivent avoir une idée de la complexité de leurs calculs.
- Fait notable et apprécié : les candidats étaient dans leur grande majorité sensibilisés aux problèmes liés à l'utilisation de nombres à virgule flottante. La plupart sait dorénavant que lorsque la machine calcule une valeur flottante d'un ordre de grandeur inférieur à 10^{-10}, il faut conjecturer que la valeur exacte doit être nulle.
- La programmation des suites définies par une relation de récurrence est généralement bien menée. On notera cependant la présence persistante de candidats qui ont utilisé des fonctions récursives de complexité exponentielle, alors même que le principe de mémoïsation ne leur est pas étranger.
- Quand on demande une valeur numérique avec une certaine précision, il faut être capable de justifier que le résultat proposé respecte cette précision. C'est surtout le cas si l'on essaie de donner une estimation de la somme d'une série numérique (ce qui implique alors de majorer un reste).
- Les fonctions `quad` et `solve` ne s'emploient qu'avec des fonctions d'une variable. Si l'on veut les employer avec des fonctions dépendant d'autres paramètres, il faut alors les utiliser en définissant une fonction à l'intérieur d'une fonction. Cela a pu surprendre certains candidats, mais un exemple – qui concerne une intégrale à paramètre – est donné dans l'aide.
- Les tracés sont globalement maîtrisés. Les erreurs les plus fréquentes sur ce point sont d'employer la commande `plot` avec des listes n'ayant pas le même nombre de termes ou de confondre abscisse et ordonnée. La commande `show()` permet de faire afficher plusieurs tracés sur une même figure : attention le résultat peut être affiché dans une fenêtre en arrière-plan et bloquer le reste de l'exécution d'un script.

Le jury regrette que les commentaires sur les graphiques obtenus soient aussi pauvres : c'est dommage car l'interprétation d'un graphique peut donner lieu à de nombreuses conjectures. Il faut que les candidats pensent à regarder les échelles sur les axes lors des sorties graphiques et pensent à les utiliser.

– Le jury est globalement satisfait de l'utilisation de la commande `odeint` pour les tracés de solution d'équation différentielle. Un point technique reste problématique : le premier élément du tableau de temps `T` est celui sur lequel porte la condition initiale. Cela peut perturber quand on demande d'effectuer le tracé d'une solution d'une équation différentielle sur un intervalle I lorsque la condition initiale est prise en un temps situé à l'intérieur de I. Le jury a parfaitement conscience de cette difficulté et a systématiquement aidé le candidat a la surmonter.

– La manipulation des tableaux `numpy` est globalement satisfaisante. Il est recommandé de savoir extraire des lignes ou des colonnes de tels tableaux. Certains candidats ignorent que le produit matriciel ne s'effectue pas grâce au symbole `*` ou `**` pour les puissances.

– L'utilisation du logiciel en algèbre linéaire demeure souvent délicate. L'utilisation du rang d'une matrice n'est pas souvent faite alors qu'elle permet de répondre simplement à de nombreuses questions. Rappelons que les vecteurs propres d'une matrice se lisent dans les colonnes (et non les lignes) de la seconde matrice renvoyée par la commande `eig` du module `numpy.linalg` et qu'un exemple montrant comment extraire un tel vecteur figure dans l'aide.

  D'autre part, cette commande renvoie toujours un résultat même lorsqu'une matrice n'est pas diagonalisable. Elle ne permet pas donc de répondre simplement à la question de la diagonalisabilité d'une matrice connaissant ses valeurs propres ; il faut en plus étudier la dimension des sous espaces propres (ce qui est assez simple en utilisant des rangs) ou encore utiliser un polynôme annulateur scindé à racines simples (et là encore, le logiciel peut faire le calcul).

– Le procédé d'orthonormalisation de Gram-Schmidt pose problème à une proportion non négligeable de candidats. Certains se lancent dans des calculs au tableau ou sur feuille, forcément fastidieux, alors que l'outil informatique est particulièrement indiqué dans ce cas. On peut conseiller au candidat pour arriver au résultat de bien décomposer les étapes de l'algorithme et d'avoir défini au préalable des fonctions calculant le produit scalaire et la norme euclidienne associée.

– En probabilités, les simulations numériques sont généralement bien menées. Cependant, peu de candidat pensent à citer la loi faible des grands nombres (ou Bienaymé-Tchebychev) pour justifier le fait qu'une moyenne de variables aléatoires indépendantes de même loi donne un résultat proche de l'espérance avec une grande probabilité. On entend trop souvent que la moyenne est « plus ou moins » la définition de l'espérance.

18.4. Conclusion

Le jury est, cette année encore, assez satisfait des résultats. L'épreuve semble bien être installée dans la préparation des candidats qui, s'ils ne sont pas tous parfaitement à l'aise, ne semblent pas déstabilisés par son contenu et son déroulement. Il convient cependant d'insister sur la maîtrise du cours, notamment de première année, pour réussir cet oral. Le jury encourage tous les futurs candidats à utiliser de manière régulière l'outil informatique pour appréhender de manière plus concrète les notions théoriques étudiées en cours de mathématiques.