REPUBLIQUE TUNISIENNE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Concours Nationaux d'Entrée aux Cycles de Formation d'Ingénieurs
Session 2020

الجمهورية التونسية
وزارة التعليم العالي والبحث العلمي

المناظرات الوطنية للدخول إلى مراحل تكوين المهندسين
دورة 2020

Concours Mathématiques et Physique, Physique et Chimie et Technologie
Epreuve d'Informatique

Date : Mercredi 22 Juillet 2020	**Heure : 8 H**	**Durée : 2 H**	**Nombre de pages** : 9

Barème : **PROBLEME 1** : 13 points
PROBLEME 2 : 7 points

DOCUMENTS NON AUTORISES
L'USAGE DES CALCULATRICES EST INTERDIT
Il FAUT RESPECTER IMPERATIVEMENT LES NOTATIONS DE L'ENONCE
VOUS POUVEZ EVENTUELLEMENT UTILISER LES FONCTIONS PYTHON DECRITES A L'ANNEXE (PAGE 9)

PROBLEME 1 :

Dans ce problème, on s'intéresse à la simulation numérique de la diffusion thermique (évolution de la température) dans un milieu homogène et isotrope. Le milieu étudié est une tige métallique mince de longueur L et de coefficient de diffusivité thermique α. L'équation de la chaleur, utilisée pour modéliser la conduction de la chaleur dans la tige, est une Equation Différentielle aux dérivées Partielles (**EDP**) donnée par **eq1** :

$$\frac{\partial u}{\partial t} = \alpha . \Delta u \qquad \textbf{eq1}$$

où l'inconnue u est une fonction à deux variables x et t :

- x est la variable de l'espace unidimensionnel, avec $x \in [0, L]$;
- t est la variable temps, avec $t \in [0, T]$, T étant la borne maximale de l'intervalle temporel.

La résolution de **eq1** permet de déterminer la température ($u(x,t)$) en tout point x du milieu à chaque instant t en prenant comme :

- condition initiale à l'instant $t = 0$:
  $u(x,0) = T_{init}(x)$ avec $T_{init}(x)$, fonction qui décrit l'état thermique initial pour tout $x \in [0, L]$
- conditions aux bords de la tige (côté gauche ($x = 0$) et côté droit ($x = L$)) :
  $u(0,t) = u(L,t) = 0$

L'**EDP** à résoudre avec les conditions précédentes est alors donnée par l'équation **eq2** :

$$\begin{cases} \dfrac{\partial u}{\partial t}(x,t) = \alpha \dfrac{\partial^2 u}{\partial x^2}(x,t) & \forall x \in [0,L],\ \forall t \in [0,T] \\ u(x,0) = T_{init}(x) & \forall x \in [0,L] \\ u(0,t) = u(L,t) = 0 & \forall t \in]0,T] \end{cases} \qquad \textbf{eq2}$$

où T_{init} est définie par :

$$T_{init}(x) = \begin{cases} 0 & \text{si } x = 0 \text{ ou } x = L \\ 200 & \forall x \in]0, L[\end{cases} \qquad \textbf{eq3}$$

Hypothèses

Dans la suite, les fonctions demandées seront écrites en langage Python en supposant que :

- le module `math` a été importé par : `import math`
- le module `numpy` a été importé par : `import numpy as np`
- le module `scipy.integrate` a été importé par : `import scipy.integrate as spi`
- les grandeurs L, T et α sont déjà définis.

Partie 1 : Résolution Analytique

La solution analytique de l'équation **eq2** s'écrit :

$$u(x,t) = \sum_{n=1}^{+\infty} D_n \sin\left(\frac{n\pi x}{L}\right) e^{-\frac{n^2\pi^2\alpha t}{L^2}} \qquad \textbf{eq4}$$

où, pour un ***n*** donné :

$$D_n = \frac{2}{L}\int_0^L f_n(x)\,dx \qquad \textbf{eq5}$$

$$f_n(x) = T_{init}(x)\sin\left(\frac{n\pi x}{L}\right) \qquad \textbf{eq6}$$

Travail demandé

1. Ecrire la fonction **Tinitial** qui prend en paramètre **x** et retourne la valeur de l'état thermique initial $T_{init}(x)$ pour tout x dans $[0, L]$.
2. Ecrire la fonction **Fn** qui prend en paramètres **x** et **n**, puis retourne le résultat du calcul exprimé par l'équation **eq6**.
3. En utilisant la fonction `quad` du module `scipy.integrate`, écrire la fonction **Dn** qui prend en paramètre **n** et retourne le résultat du calcul exprimé par l'équation **eq5**.
   NB : `quad(g,a,b,(y,))` calcule l'intégrale de la fonction g dans l'intervalle $[a,b]$ par rapport à x (y étant fixé) et retourne le tuple (A, ε) où :
   - $A = \int_a^b g(x,y)dx$
   - $\varepsilon > 0$ est une estimation de l'erreur absolue.
4. Ecrire la fonction **SolutionAnalytique** qui, à partir des paramètres **x**, **t** et **eps**, calcule et retourne la somme décrite par l'équation **eq4**. Le calcul s'arrête lorsque la valeur absolue d'un terme de la somme est inférieur à **eps**.

Partie 2 : Résolution numérique

Dans cette partie, on propose une résolution numérique d'un Système d'Equations Différentielles Ordinaires (**SEDO**) associé à l'**EDP** de l'équation **eq2**.

Après la discrétisation de l'intervalle spatial $[0, L]$, la résolution numérique du **SEDO** consiste à approcher les valeurs de la fonction $u(x,t)$ en utilisant la méthode des différences finies.

On considère :

- l'intervalle $[0, L]$ découpé en $N+1$ points équidistants $x_0 = 0 < x_1 < x_2 < \cdots < x_N = L$;
- le pas de découpage $\delta x = x_{i+1} - x_i$ $(0 \le i \le N-1)$.

Pour chaque point $x_i \in \{x_1, x_2, \ldots, x_{N-1}\}$, on approche la dérivée seconde $\frac{\partial^2 u}{\partial x^2}(x_i,t)$ par la différence finie donnée par :

$$\frac{\partial u}{\partial t}(x_i,t) = \alpha \frac{\partial^2 u}{\partial x^2}(x_i,t) \approx \frac{\alpha}{(\delta x)^2}\left(u(x_{i+1},t) - 2u(x_i,t) + u(x_{i-1},t)\right) \qquad \textbf{eq7}$$

En tenant compte de la condition initiale, le **SEDO** s'écrit :

$$\begin{cases} \dfrac{\partial u}{\partial t}(x_i,t) = \alpha \dfrac{\partial^2 u}{\partial x^2}(x_i,t) \approx \dfrac{\alpha}{(\delta x)^2}\left(u(x_{i+1},t) - 2u(x_i,t) + u(x_{i-1},t)\right) & \forall i \in \{1,\ldots,N-1\} \\ u(x_i,0) = T_{init}(x_i) & \forall i \in \{0,\ldots,N\} \\ u(0,t) = u(L,t) = 0 & \forall t \in]0,T] \end{cases} \qquad \textbf{eq8}$$

La forme linéaire du **SEDO** s'écrit finalement :

$$\frac{\partial u}{\partial t}(t) = \frac{\alpha}{(\delta x)^2} A\, u(t) \qquad \textbf{eq9}$$

où :

A est une matrice carrée tridiagonale d'ordre $N+1$ telle que :

$$A = (a_{ij}) \quad \forall (i,j) \in \{0,1,\ldots,N\} \ \text{ où } \ a_{ij} = \begin{cases} -2 & \text{si } i = j \text{ et } i \notin \{0,N\} \\ 1 & \text{si } i \in \{j+1, j-1\} \text{ et } i \notin \{0,N\} \\ 0 & \text{sinon} \end{cases}$$

$$u(t) = \begin{pmatrix} u(x_0,t) \\ u(x_1,t) \\ \vdots \\ u(x_{N-1},t) \\ u(x_N,t) \end{pmatrix} \qquad \text{avec } \ u(0) = \begin{pmatrix} 0 \\ T_{init}(x_1) \\ \vdots \\ T_{init}(x_{N-1}) \\ 0 \end{pmatrix}$$

Après la discrétisation de l'intervalle $[0,T]$ en $M+1$ points équidistants, la résolution numérique du **SEDO** donne la matrice U telle que :

$$U_{i,j} \approx u(x_i,t_j) \quad \forall\, 0 \le i \le N \quad \forall\, 0 \le j \le M \qquad \textbf{eq10}$$

Travail demandé

5. Ecrire la fonction **GenererA** qui prend un entier **N** en paramètre, construit et retourne la matrice **A**.
6. Exprimer en fonction de **N** l'ordre de grandeur (en O) de la complexité asymptotique des calculs effectués par la fonction **GenererA**. Justifier brièvement votre réponse.
7. Ecrire la fonction **SolutionNumerique** qui, à partir des paramètres **L**, **T**, **N**, **M** et **alpha**, permet de résoudre l'équation **eq9** en appliquant les étapes suivantes :
   - Déterminer, dans le vecteur **vx**, le résultat du découpage de l'intervalle $[0,L]$ en **N**+1 points équidistants ;
   - Déterminer **dx**, le pas du découpage de l'intervalle $[0,L]$;

- Déterminer, dans le vecteur **vt**, le résultat du découpage de l'intervalle de temps $[0,T]$ en **M+1** points équidistants ;
- Définir, dans **u0**, le vecteur $u(0)$;
- Générer la matrice **A** ;
- Déterminer la solution du **SEDO** de l'équation **eq9** dans la matrice **U**. Utiliser `odeint` du module `scipy.integrate` dont le résultat est la matrice transposée de **U**.

  NB : `odeint(g,u0,vt)`, permet de résoudre numériquement le SEDO de l'équation **eq9** $\frac{\partial u}{\partial t}(t) = g(u,t)$ avec la condition initiale le vecteur **u0** et le domaine d'intégration **vt**.
- Retourner le tuple (**U, vx, vt**).

Partie 3 : Programmation Orientée Objet

La solution de la résolution numérique de l'équation de la chaleur permet d'approcher la température en tout point $x_i \in \{x_0, x_1, \ldots, x_N\}$, à tout instant $t_i = \{t_0, t_1, \ldots, t_M\}$. En utilisant l'interpolation bilinéaire, on peut estimer la température en tout point $x \in [0,L]$ et à n'importe quel instant $t \in [0,T]$.

Dans cette partie, on s'intéresse à une représentation orientée objet de la résolution numérique du **SEDO** par la classe **EqChaleur**, puis à l'interpolation bilinéaire de la solution résultante pour des points quelconques par la classe **InterpolationBilineaire**.

Description de la classe EqChaleur

Cette classe modélise la résolution numérique de l'équation de la chaleur.

- Attributs :
  - **L**, longueur de la tige.
  - **T**, durée totale de la simulation.
  - **N**, un entier tel que **N+1** est le nombre de points équidistants de l'espace discrétisé $[0,L]$.
  - **M**, un entier tel que **M+1** est le nombre de points équidistants de l'intervalle temporel discrétisé $[0,T]$.
  - **alpha**, coefficient réel de la diffusivité de la tige.

- Méthodes :
  - **__init__(...)** : permet l'initialisation des attributs de la classe **EqChaleur**, à partir des paramètres **L, T, N, M** et **alpha**.
  - **SolveEq(...)** : permet de résoudre l'équation de la chaleur en appelant la fonction **SolutionNumerique** écrite en partie 2.

Description de la classe InterpolationBilineaire

Cette classe permet d'interpoler la solution de la résolution numérique de l'équation de la chaleur.

- Attributs :
  - **U**, une matrice représentant l'approximation des valeurs de la fonction $u(x,t)$.
  - **bornes_max**, un tuple contenant la plus grande valeur du vecteur **vx** et celle de **vt**.
  - **pas_v**, un tuple contenant le pas de découpage du vecteur **vx** et celui de **vt**.

- Méthodes :
  - **__init__(...)** : permet l'initialisation des attributs de la classe **InterpolationBilineaire**, à partir des trois paramètres **U**, **vx** et **vt**.
  - **__str__(...)** : permet de retourner une représentation textuelle de l'instance courante conformément au format suivant :

    `"F : [0, xmax] × [0,tmax] --> [umin, umax]"`

    où :

    - xmax et tmax représentent la valeur maximale de **vx** et celle de **vt** ;
    - umin et umax représentent respectivement la valeur minimale et la valeur maximale de U.

  - **__contains__(...)** : qui à partir d'un tuple de réels **tup**, représenté par (x,t), retourne `True` si $x \in [0, \text{xmax}]$ et $t \in [0, \text{tmax}]$, `False` sinon.
  - **get (...)** : qui à partir d'un tuple de réels **tup**, représenté par (x,t), détermine et retourne un tuple de réels représenté par (p, q) tels que :
    - $p = \frac{x}{\text{dx}}$, avec dx le pas de découpage du vecteur **vx** ;
    - $q = \frac{t}{\text{dt}}$, avec dt le pas de découpage du vecteur **vt**.
  - **get_low(...)** : qui à partir d'un tuple de réels **tup**, retourne un tuple d'entiers en appliquant la fonction `floor` du module `math` pour chaque élément du tuple résultat de l'appel de la méthode get.
  - **get_up(...)** : qui à partir d'un tuple de réels **tup**, retourne un tuple d'entiers en appliquant la fonction `ceil` du module `math` pour chaque élément du tuple résultat de l'appel de la méthode get.
  - **interpolate(...)** : qui à partir d'un tuple de réels **tup**, représenté par (x,t), permet d'interpoler $u(x,t)$ par la méthode bilinéaire si $x \in [0, \text{xmax}]$ et $t \in [0, \text{tmax}]$. Gérer l'erreur dans le cas contraire.

    Les étapes de la méthode d'interpolation bilinéaire sont :
    - déterminer dans (**x_low**, **t_low**), le résultat de l'appel de la méthode **get_low** ;
    - déterminer dans (**x_up**, **t_up**), le résultat de l'appel de la méthode **get_up** ;
    - construire le vecteur $\boldsymbol{v} = \begin{pmatrix} 1 \\ x \\ t \\ x \times t \end{pmatrix}$;
    - calculer les valeurs x_l , x_u , t_l et t_u telles que :

      $x\_l = x\_low \times dx$; $x\_u = x\_up \times dx$; $t\_l = t\_low \times dt$; $t\_u = t\_up \times dt$;

    - construire la matrice $B = \begin{pmatrix} 1 & x\_l & t\_l & x\_l \times t\_l \\ 1 & x\_l & t\_u & x\_l \times t\_u \\ 1 & x\_u & t\_l & x\_u \times t\_l \\ 1 & x\_u & t\_u & x\_u \times t\_u \end{pmatrix}$;

- construire le vecteur $b = \begin{pmatrix} u_{x\_low,t\_low} \\ u_{x\_low,t\_up} \\ u_{x\_up,t\_low} \\ u_{x\_up,t\_up} \end{pmatrix}$;
- déterminer le vecteur ***y*** solution de la résolution du système linéaire $B.y = b$;
- retourner la valeur du produit scalaire des vecteurs ***y*** et ***v***.

Travail demandé

En se basant sur les descriptions ci-dessus, répondre aux questions suivantes :

8. Construire la classe **EqChaleur** :
- **8.1** Ecrire la méthode **__init__**.
- **8.2** Ecrire la méthode **SolveEq**.

9. Construire la classe **InterpolationBilineaire** :
- **9.1** Ecrire la méthode **__init__**.
- **9.2** Ecrire la méthode **__str__**.
- **9.3** Ecrire la méthode **__contains__**.
- **9.4** Ecrire la méthode **get**.
- **9.5** Ecrire la méthode **get_low**.
- **9.6** Ecrire la méthode **get_up**.
- **9.7** Ecrire la méthode **interpolate**.

10. En supposant disposer des valeurs de **L**, **T**, **N**, **M** et **alpha**, écrire le script Python permettant de :
- créer une instance **chal** de la classe **EqChaleur** ;
- déterminer dans **(U,vx,vt)** la solution numérique de **chal** ;
- créer une instance **bil** de la classe **InterpolationBilineaire** ;
- saisir deux réels **x** et **t** en respectant la condition suivante : **x** n'appartient pas à **vx** et **t** n'appartient pas à **vt** ;
- donner la température du point **x** à l'instant **t**.

PROBLEME 2

Un concours national d'entrée aux cycles de formation d'ingénieurs réunit chaque année des milliers de candidats qui passent un ensemble d'épreuves spécifiques aux sections MP (Mathématiques et Physique), PC (Physique et Chimie), T (Technologie) et BG (Biologie et Géologie). On propose de concevoir en partie une application de gestion de certains résultats des épreuves d'un concours.

On considère une partie d'une base de données schématisée par les relations décrites ci-dessous. Bien que se rapprochant de la réalité, il est à noter que le schéma a été élaboré pour les besoins de l'énoncé.

- **Etablissement** (idEtab, nom, adresse, gouv)

  La table **Etablissement** décrit tous les établissements universitaires assurant un cycle préparatoire ainsi qu'un établissement fictif (service concours) pour les candidats libres.
  - idEtab : identifiant de l'établissement contenant le nom abrégé (chaîne de caractères), *clé primaire* (l'établissement fictif a comme identifiant la valeur 'Libre') ;
  - nom : nom complet de l'établissement (chaîne de caractères) ;
  - adresse : adresse de l'établissement (chaîne de caractères) ;
  - gouv : nom du gouvernorat où l'établissement est situé (chaîne de caractères).

- **Candidat** (idC, nom, date_naissance, sexe, nationalite, adresse, section, idEtab)

  La table **Candidat** décrit tous les candidats inscrits au concours.
  - idC : identifiant du candidat (entier), *clé primaire* ;
  - nom : nom et prénom du candidat (chaîne de caractères) ;
  - date_naissance : date de naissance du candidat (chaîne de caractères en format 'aaaa/mm/jj' avec 'aaaa' : année, 'mm' : mois et 'jj' : jour) ;
  - sexe : sexe du candidat (un caractère, 'F' pour féminin et 'M' pour masculin) ;
  - nationalite : nationalité du candidat (chaîne de caractères) ;
  - adresse : adresse du domicile du candidat (chaîne de caractères) ;
  - section : section à laquelle le candidat appartient (chaîne de caractères, 'MP', 'PC', 'T', 'BG') ;
  - idEtab : identifiant de l'établissement d'origine du candidat (chaîne de caractères), *clé étrangère* qui fait référence à la table **Etablissement**.

  Tous les candidats non inscrits à un cycle préparatoire sont considérés comme candidats libres et affectés à l'établissement fictif dont l'attribut idEtab est 'Libre'.

- **Epreuve** (idEpr, nomEpr, section, dateEpr, heure, duree, coeff)

  La table **Epreuve** décrit toutes les épreuves du concours.
  - idEpr : identifiant de l'épreuve (entier), *clé primaire* ;
  - nomEpr : nom de l'épreuve (chaîne de caractère) ;
  - section : section à laquelle l'épreuve est associée (chaîne de caractères, 'MP', 'PC', 'T', 'BG') ;
  - dateEpr : date de l'épreuve (chaîne de caractères en format 'aaaa/mm/jj' avec 'aaaa' : année, 'mm' : mois et 'jj' : jour) ;
  - heure : horaire de passage de l'épreuve (chaîne de caractères en format 'hh : mm' avec 'hh' : heures et 'mm': minutes) ;
  - duree : durée de l'épreuve en nombre d'heures (entier) ;
  - coeff : coefficient de l'épreuve (entier).

- **Evaluation** (idC, idEpr, note)

  La table **Evaluation** décrit les notes des candidats aux épreuves passées.
  - idC : identifiant du candidat (entier), *clé étrangère* qui fait référence à la table **Candidat** ;
  - idEpr : identifiant de l'épreuve passée (entier), *clé étrangère* qui fait référence à la table **Epreuve** ;
  - note : la note du candidat à une épreuve donnée (un réel compris entre 0 et 20).

  Le couple (idC, idEpr) représente la *clé primaire* de la table Evaluation.

Partie 1 : Algèbre relationnelle

Exprimer en algèbre relationnelle les requêtes permettant de donner :

1. Les noms des candidats libres de la section 'PC'.
2. Les identifiants des candidats qui n'ont passé aucune épreuve du concours.

Partie 2 : SQL

Dans la suite, en supposant que toutes les tables sont créées et remplies en respectant les règles d'intégrités, exprimer en SQL les requêtes suivantes permettant de :

3. Modifier à 2 heures la durée de l'épreuve ayant le nom 'Informatique' de la section 'T'.
4. Déterminer les noms des établissements assurant un cycle préparatoire pour la section 'BG'.
5. Déterminer, pour chaque section, le total des coefficients des épreuves en triant les totaux par ordre décroissant.
6. Déterminer les identifiants des candidats ayant obtenu des notes supérieures à 15, au moins à 3 épreuves.

Partie 3 : sqlite3

Une épreuve est dite discriminante si elle a le plus d'influence sur les résultats du concours. La comparaison entre les écarts types des notes des épreuves est un moyen permettant de déterminer l'épreuve la plus discriminante du concours.

L'écart type des notes d'une épreuve modélise le degré de dispersion des notes des candidats autour de la moyenne. Ainsi, plus les notes sont largement distribuées autour de la moyenne, plus l'écart type est élevé et plus l'épreuve est discriminante. Dans le cas contraire, les notes seraient concentrées autour de la moyenne et l'épreuve n'aurait donc pas d'influence significative sur les scores des candidats.

Pour une épreuve donnée, considérons n candidats ayant obtenu les notes x_i $(1 \leq i \leq n)$. La moyenne des notes à cette épreuve est notée $\overline{x}$.

L'écart type σ des notes de cette épreuve s'écrit : $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2}$.

Travail demandé

Dans la suite, les fonctions demandées doivent être écrites en Python en désignant par **cur**, le curseur d'exécution de requêtes.

7. Ecrire la fonction **Notes** qui prend en paramètres **cur** et l'identifiant **id** d'une épreuve, puis retourne une liste contenant les notes de tous les candidats qui ont passé cette épreuve.
8. Ecrire la fonction **EcartType** qui prend en paramètres **cur** et l'identifiant **id** d'une épreuve puis calcule et retourne la valeur de l'écart type des notes de l'épreuve.
9. Ecrire la fonction **Epreuves** qui prend en paramètres **cur** et le nom **s** d'une section, puis retourne un ensemble contenant les identifiants des épreuves de **s**.
10. Ecrire la fonction **EcartsTypesEpreuves** qui prend en paramètres **cur** et le nom **s** d'une section, puis retourne un dictionnaire où chaque clé est le nom d'une épreuve de **s** et chaque valeur est l'écart type des notes pour cette épreuve.
11. Ecrire la fonction **discriminante** qui prend en paramètres **cur** et le nom **s** d'une section, puis retourne le nom de l'épreuve la plus discriminante.

ANNEXE – Quelques fonctions/Méthodes Python

Sans aucune obligation, les fonctions suivantes pourraient vous être utiles.

Module math

- **floor**(r) retourne la partie entière d'un réel r.
- **ceil**(r) retourne la partie entière d'un réel r +1.

Module scipy.integrate

- **quad**(g, a, b, args) calcule l'intégrale de la fonction g dans l'intervalle [a,b] par rapport au premier paramètre de g, args étant le restant des paramètres.
- **odeint**(f, y0, t) permet de résoudre numériquement un système d'équations différentielles ordinaires avec la condition initiale y0 et t le domaine d'intégration.

Module numpy

- M.**shape** ou **shape**(M) retourne un tuple formé par le nombre de lignes et le nombre de colonnes d'une matrice M.
- **linspace**(a,b,n) retourne un vecteur de n points équidistants dans l'intervalle [a,b].
- **zeros**(...) retourne un tableau initialisé par des 0.
- **ones**(...) retourne un tableau initialisé par des 1.
- **diag**(L) retourne la matrice diagonale dont les éléments diagonaux sont ceux de L.
- M.**copy**() retourne une copie de M.
- **identity**(N) ou **eye**(N) retourne la matrice identité d'ordre N (avec $N \neq 0$).
- M.**min**() ou **min**(M) et M.**max**() ou **max**(M) retournent respectivement la valeur minimale et la valeur maximale d'une matrice M.
- **transpose**(M), M.**T** ou M.**transpose**() retourne la matrice transposée de la matrice M.
- **dot**(A,B) ou A.**dot**(B) retourne le produit matriciel de A par B ou la valeur du produit scalaire si A et B sont deux vecteurs.
- g=**vectorize**(f) retourne la fonction g vectorisée de la fonction f qui s'applique terme à terme aux éléments d'un tableau v. g(v) retourne le tableau dont les éléments sont les éléments images de v par g.
- **fromfunction**(f, shape,...) retourne la matrice résultat de l'application de la fonction f aux indices du tableau. Le type des éléments est celui de f.

Module numpy.linalg

- **solve**(A,b) retourne le vecteur solution du système linéaire A.x=b.
- **det**(A) retourne le déterminant de A.

Opérations sur les itérables (str, tuple, list, dict, etc.)

- **len**(it) retourne le nombre d'éléments de l'itérable it.
- **range**(d,f,p) retourne la séquence des valeurs entières successives comprises entre d et f, f exclu, par pas=p.
- **sum**(it) retourne la somme des éléments de l'itérable it.
- x **in** it vérifie si x appartient à it.
- **sorted**(it) retourne une liste contenant les éléments de it dans l'ordre croissant.
- lst.**sort**() trie la liste lst dans l'ordre croissant.
- lst.**append**(val) ajoute val à la fin de la liste lst.
- ch.**format**(paramètres) retourne une chaîne de caractères obtenue en substituant dans l'ordre chaque '{}' dans ch par un objet de paramètres.
- d.**values**() retourne un itérable formé par les valeurs du dictionnaire d.
- d.**items**() retourne un itérable de couples (k,v) ou k est une clé du dictionnaire d et v est la valeur associée.

REPUBLIQUE TUNISIENNE

Ministère de l'Enseignement Supérieur,
de la Recherche Scientifique

Concours Nationaux d'Entrée
aux Cycles de Formation d'Ingénieurs
Session 2020

الجمهورية التونسية

وزارة التعليم العالي والبحث العلمي

المناظرات الوطنية للدخول
إلى مراحل تكوين المهندسين
دورة 2020

Alternative de Correction Concours Mathématiques et Physique, Physique et Chimie et Technologie
Epreuve d'Informatique

Barème sur 100
PROBLEME 1 (65 pts)
Partie 1 :

1. 1.25 pt

Version 1:

```
Tinitial= lambda x: 0 if x==0 or x==L else 200
```

Version 2:

```
def Tinitial(x):
    return 0 if x in (0,L) else 200
```

Version 3:

```
def Tinitial(x):
   assert 0<=x<=L , 'longueur depassant la tige'
   if x==0 or x==L :
      return 0
   else:
      return 200
```

2. 1.25 pt

Version 1:

```
def Fn(x,n):
    return Tinitial(x) * np.sin((n*np.pi*x)/L)
```

Version 2:

```
Fn=lambda x,n:Tinitial(x)*np.sin((np.pi*n*x)/L)
```

3. 2.5 pts

```
def Dn(n):
    Q=spi.quad(Fn,0,L,(n,))
    return (2/L)*Q[0]
```

4. 5 pts

```
def SolutionAnalytique(x,t,eps):
    n=1
    s=0
    while True:
      t=Dn(n)*np.sin(n*x*np.pi/L)*np.exp((-n*n*np.pi**2*alpha*t)/L**2)
      s+=t
      n+=1
```

```
if abs(t)< eps:
      return s
```

Partie 2

5. 5 pts

Version 1:

```
def GenererA(N):
     A=np.zeros((N+1,N+1),'int')
     for i in range(1,N+1):
          A[i,i]=-2
          A[i-1,i]=1
          A[i,i-1]=1
     A[0]=A[N]=np.zeros(N+1)
     return A
```

Version 2:

```
def GenererA(N):
    a=np.diag([-2 for i in range(0,N+1)])
    b=np.diag([1 for i in range(0,N)],1)
    c=np.diag([1 for i in range(0,N)],-1)
    A=a+b+c
    A[0,:]=A[N,:]=0
    return A
```

Version 3:

```
def GenererA(N):
    A=np.zeros((N+1,N+1))
    for i in range(1,N):
        for j in range(N+1):
            if i==j :
                A[i,j]=-2
            elif i==j+1 or i==j-1:
                A[i,j]=1
    return A
```

Version 4:

```
def GenererA(N):
    L=[-2 for i in range(N+1)]
    A=np.diag(L)
    A[0,0]=A[N,N]=0
    for i in range(1,N):
        for j in range(N+1):
            if i==j+1 or i==j-1:
                A[i,j]=1
    return A
```

Version 5:

```
def GenererA(N):
      fill = lambda i,j : -2 if i == j and i not in {0,N} else \
      (1 if i in {j+1, j-1} and i not in {0,N} else 0)
      return np.fromfunction(np.vectorize(fill), (N+1,N+1))
```

6. 2.5 pts

$O(N^2)$, la fonction permet de remplir une matrice d'ordre (N+1)

7. 10 pts

Version 1 :

```
def SolutionNumerique(L,T,N,M,alpha):
    vx=np.linspace(0,L,N+1)
    dx=vx[1]-vx[0]
    vt=np.linspace(0,T,M+1)
    u0=np.zeros(N+1)
    for i in range(1,N+1): #ou bien u0=np.array([Tinitial(i) for i in vx])
      u0[i]=Tinitial(vx[i])
    A=GenererA(N)
    k= lambda u,t: alpha/(dx*dx)*np.dot(A,u)
    U=np.transpose(spi.odeint(k, u0, vt))
    return U,vx,vt
```

Version 2 :

```
def SolutionNumerique(L,T,N,M,alpha):
    vx, dx = np.linspace(0,L,N+1, retstep = True)
    vt= np.linspace(0, T, M+1)
    u0 = np.vectorize(Tinitial)(vx)
    A = GenererA(N)
    k = lambda u, t : (alpha/dx**2)* (A.dot(u))
    U = spi.odeint(k, u0, vt).T # ou bien spi.odeint(k, u0, vt).transpose()
    return (U, vx, vt)
```

Partie 3

8. 2.5 pts + 2.5 pts

```
class EqChaleur:
```

8.1

```
def __init__(self,L,T,N,M,alpha):
      self.L=L
      self.T=T
      self.N=N
      self.M=M
      self.alpha=alpha
```

8.2

```
def SolveEq(self):
      return SolutionNumerique(self.L,self.T,self.N,self.M,self.alpha)
```

9.

```
class InterpolationBilinieaire:
```

9.1 5 pts

```
def __init__(self,U,vx,vt):
   self.U = U.copy() # self.U=U
   self.bornes_max = (vx.max(), vt.max())
   self.pas_v = (vx[1]-vx[0], vt[1]-vt[0])
```

9.2 2.5 pts

Version 1:

```
def __str__(self) :
    motif = "F : [0, {}] x [0, {}] --> [{}, {}]"
    bxmax, btmax  = self.bornes_max
    return motif.format(bxmax, btmax, self.U.min(), self.U.max())
```

Version 2:

```
def __str__(self):
   xmax,tmax= self.bornes_max
```

```
    Umin,Umax=self.U.min(),self.U.max()
    ch= 'F:[0,'+str(xmax)+']*[0,'+ str(tmax)+'] -->\
                          ['+str(Umin)+','+str(Umax)+']'
    return ch
```

9.3 2.5 pts

Version 1:

```
def __contains__(self,tup):
  if 0< tup[0]< self. bornes_max [0] and 0< tup[1]< self. bornes_max [1]:
             return True
  else:
            return False
```

Version 2:

```
def __contains__(self, tup):
   x, t = tup
   xmax, tmax = self.bornes_max
   return 0 < x < xmax and 0 < t < tmax
```

9.4 2.5 pts

```
def get(self,tup):
      return (tup[0]/self.pas_v[0],tup[1]/self.pas_v[1])
```

9.5 1.25 pts

Version 1:

```
def getlow(self,tup):
      t1=self.get(tup)
      t2= math.floor(t1[0]),math.floor(t1[1])
      return t2
```

Version 2:

```
def get_low(self, tup):
      return tuple(math.floor(x) for x in self.get(tup))
```

9.6 1.25 pts

Version 1:

```
def getup(self,tup):
      t1=self.get(tup)
      t2= math.ceil(t1[0]),math.ceil(t1[1])
      return t2
```

Version 2:

```
def get_up(self, tup):
      return tuple(math.ceil(x) for x in self.get(tup))
```

9.7 10 pts

```
from np.linalg import solve
```

Version 1:

```
def interpolate(self,tup):
     if tup in self:
        xlow,tlow=self.getlow(tup)
        x_up,t_up=self.getup(tup)
        v=np.array([1,tup[0],tup[1],tup[0]*tup[1]])
        xl=xlow*self.pas_v[0]
        xu=x_up*self.pas_v[0]
        tl=tlow*self.pas_v[1]
        tu=t_up*self.pas_v[1]
        c1=np.ones(4,'int')
        c2=np.array([xl,xl,xu,xu])
        c3=np.array([tl,tu,tl,tu])
```

```
            B=np.array([c1,c2,c3,c2*c3])
            B=np.transpose(B)
            b=np.array([self.U[xlow,tlow],self.U[x_up,tlow],\
                                self.U[xlow,t_up],self.U[x_up,t_up]])
            y=solve(B,b)
            return np.dot(y,v)
          else:
            return 'Erreur'
```

Version 2:

```
    def interpolate(self, tup):
      assert tup in self
      x, t  = tup
      dx, dt = self.pas_v
      x_low, t_low = self.get_low(tup)
      x_up, t_up = self.get_up(tup)
      v = np.array([1, x, t, x * t ])
      x_l, x_u, t_l, t_u  = np.array([x_low,x_up,t_low ,t_up]) * [dx,dx,dt,dt]
      B = np.ones((4,4))
      B[0,1] = B[1,1] = x_l
      B[2,1] = B[3,1] = x_u
      B[0,2] = B[2,2] = t_l
      B[1, 2] = B[3,2] = t_u
      B[:,-1] = B[:,-2] * B[:,-3]
      b = np.array([self.U[i,j] for i in (x_low, x_up) for j in (t_low,
t_up)])
      y = solve(B,b)
      return y.dot(v)
```

10. 7.5 pts

```
chal=EqChaleur(L,T,N,M,alpha)
U,vx,vt=chal.SolveEq()
bil=InterpolationBilinieaire(U,vx,vt)
while 1:
    try:
        x=float(input('lire x'))
        t=float(input('lire t'))
        if x not in vx and t not in vt:
            break
    except:
        print('saisir des réels')
print(bil.interpolate((x,t)))
```

PROBLEME 2

Partie 1

1. **2.5 pts**

   $\pi_{nom}(\sigma_{idEtab = 'Libre' \text{ et } section = 'PC'}(Candidat))$

2. **2.5 pts**

   $\pi_{idC}(Candidat)$ - $\pi_{idC}(Evaluation)$

Partie 2

3. **2.5 pts**

   update Epreuve set duree = 2 where nomEpr='Informatique' and section= 'T';

4. **2.5 pts**

   select DISTINCT E.nom from Etablissement E join candidat C on E.idEtab = C.idEtab
   where C.section ='BG' and E.idEtab <> 'Libre'

5. **3.75 pts**
   select section, sum(coeff) s from Epreuve group by section order by s desc;

6. **3.75 pts**
   SELECT IdC FROM Evaluation WHERE note >= 15 GROUP BY IdC Having count(*) >= 3;
   ou
   select idC, count(idEpr) s from evaluation where note >=15 group by idC having s>=3;

Partie 3 :

7. **2.5 pts**

```
def Notes(cur,id):
    cur.execute('select note from Evaluation where idEpr=?', [id])
    L=cur.fetchall()
    return [i[0] for i in L]
```

8. **2.5 pts**

```
def ecart_type(cur,id):
    L=Notes(cur,id)
    m= sum(L)/len(L)
    s=0
    for i in L:
        s+=(i-m)**2
    return math.sqrt(s/len(L))
```

9. **2.5 pts**

```
def Epreuves(cur,s):
    cur.execute('select idEpr from Epreuve where section=?', (s,))
    L=cur.fetchall()
    return {i[0] for i in L}
```

10. **2.5 pts**

```
def ecartypeEpreuves(cur,s):
    req = """
    SELECT nomEpr FROM Epreuve WHERE IdEpr = ?
    """
    L=Epreuves(cur,s)
    d_ecart={}
    for i in L:
        cur.execute(req, [i])
        nom = cur.fetchone()[0]
        d_ecart[nom]=ecart_type(i)
    return d_ecart
```

11. 5 pts

```
def discriminante(cur,s):
     d=ecartypeEpreuves(cur,s)
     m=-1
     for i in d :
          if d[i]>m:
               m=d[i]
               ep=i
     return ep
```

Version 1

```
def discriminante(cur, s):
    d = EcartTypeEpreuves(cur, s)
    return max(d, key = lambda nomEpr : d[nomEpr])
```

version 2

```
def discriminante(cur, s):
    d = EcartTypeEpreuves(cur, s)
    ref = -1
    res = None
    for nomEpr in d:
        if d[nomEpr] > ref:
            ref = d[nomEpr]
            res = nomEpr
    return res
```

Version 3

```
def discriminante(cur, s):
    d = EcartTypeEpreuves(cur, s)
    ref = -1
    res = None
    for nomEpr, stdEpr in d.items():
        if stdEpr > ref:
            ref = stdEpr
            res = nomEpr
    return res
```