Université de Carthage Institut Préparatoire aux Études d'Ingénieur de Nabeul Département : Mathématiques	IPEIN المعهد التحضيري للدراسات الهندسية بنابل Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul	Année universitaire : 2021/2022 Filière : SM/SP/ST Niveau d'étude : 2ème année Semestre : 2 Nombre de pages : 6 pages Date : 14/05/2022 Durée : 2h00

Documents non autorisés

Exercice 1. (3 points)

On cherche à calculer une valeur approchée de l'intégrale d'une fonction f sur le segment où elle est définie. La méthode des rectangles nous permettait d'obtenir une valeur approchée de $\int_a^b f(t)dt$ avec f une fonction continue sur $[a,b]$.
De la même façon, si (a_k) désigne une subdivision de $[a,b]$ telle que :

$$a = a_0 < a_1 < \ldots < a_{n-1} < a_n = b$$

On peut approcher la fonction f sur chacun des segments $[a_k, a_{k+1}]$ par des rectangles de hauteur $f(m_k)$ avec pour tout $k \in [0..n-1]$, m_k le milieu du segment $[a_k, a_{k+1}]$. On parle alors de *la méthode du point milieu.*

1. Donner sous forme explicite l'aire $A_n(f)$ des n rectangles ainsi obtenus sur le segment $[a,b]$.
2. On considère la fonction `sin` sur le segment $[a,b]$ et on note (a_k) la subdivision à pas constant définie par :
$$\forall k \in [0,n], a_k = a + k\frac{(b-a)}{n}$$
Construire la fonction **ptmilieu** qui pour a, b, n donnés, renvoie l'aire $A_n(f)$ des rectangles obtenus par la méthode du point milieu.
3. Avec le langage Python, définir $f : t \mapsto sin(t)$ et caculer numériquement son intégrale sur $[0;\pi]$.

Problème 1. (7 points)

Partie 1.

En analyse numérique, *la méthode de décomposition LU*, est un algorithme de factorisation LU d'une matrice A donnée. La factorisation consiste à écrire la matrice A comme le produit de deux autres matrices L et U : $A = LU$.
L est une matrice trianglaire inférieure ayant des 1 sur la diagonale et U une matrice supérieure.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1j} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2j} & \ldots & a_{2n} \\ \ldots & \ldots & & & & \\ a_{i1} & a_{i2} & \ldots & a_{ij} & \ldots & a_{in} \\ \ldots & \ldots & & & & \\ a_{n1} & a_{n2} & \ldots & a_{nj} & \ldots & a_{nn} \end{pmatrix}$$

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & \dots & 0 & \dots & 0 \\ l_{21} & 1 & 0 & 0 & \dots & 0 \\ \dots & \dots & & & & \\ l_{i1} & \dots & \dots & 1 & 0 & 0 \\ \dots & \dots & & & 1 & 0 \\ l_{n1} & \dots & \dots & \dots & l_{n(n-1)} & 1 \end{pmatrix} \quad ; \quad \mathbf{U} = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1j} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2j} & \dots & u_{2n} \\ \dots & 0 & & & & \\ 0 & 0 & 0 & u_{ij} & \dots & u_{in} \\ \dots & \dots & & & & \\ 0 & 0 & \dots & 0 & 0 & u_{nn} \end{pmatrix}$$

Présentation de la méthode :
On considère les matrices suivantes :

- $L = I_n$: matrice identité (ou matrice unité).
- U : matrice nulle.

Etape 1 On crée une boucle pour i variant de 1 jusqu'à $(n-1)$:

Etape 2 Pour chaque valeur de i on applique les formules de récurrences suivantes :

Etape 2.1 pour j variant de i jusqu'à n :

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}.u_{kj} \qquad j \in \{i, i+1, \dots, n\}$$

Etape 2.2 pour j variant de $i+1$ jusqu'à n :

$$l_{ji} = \frac{1}{u_{ii}} \left(a_{ji} \sum_{k=1}^{i-1} l_{jk}.u_{ki} \right) \qquad j \in \{i+1, i+2, \dots, n\}$$

Etape 3 On termine par le calcul du terme u_{nn}

$$u_{nn} = a_{nn} \sum_{k=1}^{n-1} l_{nk}.u_{kn}$$

Questions :

1. Créer une fonction **saisie_mat(n,m)** permettant de saisir et retourner une matrice d'ordre (n,m). Saisir la matrice $A_{(n,n)}$, avec $n = 4$, .
2. Créer la matrice identité $L = I_n$.
3. Créer la matrice nulle U d'ordre n.
4. Créer une fonction **somme (L,U, p, i, j)** qui calcule et renvoie la somme suivante : $\sum_{k=1}^{p-1} L_{ik}.U_{kj}$
5. Programmer en Python les trois étapes de la méthode *Décomposition LU*.
6. Vérifier que $A = LU$.

Partie 2.

Il s'agit de développer un programme Python de résolution d'un système matriciel $A.X = B$ où A et B sont deux matrices données.
On commence par déterminer un système équivalent :

$$L.U.X = B$$

On résout alors successivement les deux systèmes suivants :

$$\begin{cases} L.Y &= B \\ U.X &= Y \end{cases}$$

On utilise les notations suivantes :

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1j} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2j} & \dots & a_{2n} \\ \dots & \dots & & & & \\ a_{i1} & a_{i2} & \dots & a_{ij} & \dots & a_{in} \\ \dots & \dots & & & & \\ a_{n1} & a_{n2} & \dots & a_{nj} & \dots & a_{nn} \end{pmatrix} \quad ; \quad \mathbf{X} = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_i \\ \dots \\ x_n \end{pmatrix} \quad ; \quad \mathbf{B} = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_i \\ \dots \\ b_n \end{pmatrix}$$

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & \dots & 0 & \dots & 0 \\ l_{21} & 1 & 0 & 0 & \dots & 0 \\ \dots & \dots & & & & \\ l_{i1} & \dots & \dots & 1 & 0 & 0 \\ \dots & \dots & & & 1 & 0 \\ l_{n1} & \dots & \dots & \dots & l_{n(n-1)} & 1 \end{pmatrix} \quad ; \quad \mathbf{U} = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1j} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2j} & \dots & u_{2n} \\ \dots & 0 & & & & \\ 0 & 0 & 0 & u_{ij} & \dots & u_{in} \\ \dots & \dots & & & & \\ 0 & 0 & \dots & 0 & 0 & u_{nn} \end{pmatrix}$$

Présentation de la méthode :

Etape 1 Résolution du système $L.Y = B$

On utilise les formules de récurrences suivantes :

$$\begin{cases} y_1 &= \dfrac{b_1}{l_{11}} \\ y_i &= \dfrac{1}{l_{ii}}\left(b_i - \displaystyle\sum_{j=1}^{i-1} l_{ij}.y_i\right) \quad i \in \{2,3,\dots,n\} \end{cases}$$

Etape 2 Résolution du système $U.X = Y$

On utilise les formules de récurrences suivantes :

$$\begin{cases} x_n &= \dfrac{y_n}{u_{nn}} \\ x_i &= \dfrac{1}{u_{ii}}\left(y_i - \displaystyle\sum_{j=i+1}^{n} u_{ij}.x_j\right) \quad i \in \{n-1, n-2, \dots, 1\} \end{cases}$$

Questions :

1. Créer une fonction **sommeU (U,X,i,n)** qui calcule et renvoie la somme suivante : $\displaystyle\sum_{j=i+1}^{n} U_{ij}.X_j$
2. Créer une fonction **sommeL (L,Y,i)** qui calcule et renvoie la somme suivante : $\displaystyle\sum_{j=1}^{i-1} L_{ij}.Y_j$
3. Saisir une matrice vecteur $B_{(n,1)}$.
4. Créer les matrices vecteur de zéros $X_{(n,1)}$ et $Y_{(n,1)}$.
5. Résoudre le système $L.Y = B$
6. Résoudre le système $U.X = Y$
7. Afficher les matrices A, B et X.

Problème 2. (10 points)

Une institut veut gérer son parc informatique à l'aide d'une base de données qui décrit les installations de logiciels sur des postes de travail. Un poste de travail est une machine sur laquelle sont installés certains logiciels, gratuits ou payants.
Pour chaque installation on enregistre la date d'installation (sous la forme AAAA/MM/JJ) et la durée de la licence (en mois). Lorsque le logiciel est gratuit, on saisit la valeur «infini» pour la durée de licence.
On a établi le modèle relationnel ci-dessous, dans lequel les données sont organisées sous forme de relations :

- Logiciel (id, nom, version, prix)
- Installer (#id_poste, #id_logiciel, date_installation, duree_licence)
- Poste(id, nom, #id_salle)
- Salle(id, nom, nb_prises_reseau)

Les attributs de la relation **Logiciel** sont :

- **id** : identifiant, entier naturel non nul, s'incrémente automatiquement.
- **nom** : attribut non nul de type chaine de caractères.
- **version** : attribut non nul de type chaine de caractères, dans la mésure où la version d'un logicel s'écrit sous la forme de nombres séparés par un ou plusieurs points, comme 5.4.1 par exemple.
- **prix** : un nombre décimal avec deux chiffres après la virgule. Un logiciel gratuit a un prix égale à 0.

Les attributs de la relation **Installer** sont :

- **id_poste** : a le même domaine que l'attribut `id` de la relation `poste` . En effet, c'est une clé étrangère qui fait référence à la clé primaire `id` de la relation `poste`.
- **id_logiciel** : a le même domaine que l'attribut `id` de la relation `logiciel` . En effet, c'est une clé étrangère qui fait référence à la clé primaire `id` de la relation `logiciel`.
- **date_installation** : est un attribut non nul de type `DATE`.
- **duree_licence** : est un attribut non nul, de type chaine de caractères dans la mesure où on saisit la valeur «`infini`» lorsque le logiciel est gratuit et un nombre entier naturel de mois sinon.

Les attributs de la relation **Poste** sont :

- **id** : identifiant, entier naturel non nul, s'incrémente automatiquement.
- **nom** : attribut non nul de type chaine de caractères.
- **id_salle** : a le même domaine que l'attribut `id` de la relation `salle` . En effet, c'est une clé étrangère qui fait référence à la clé primaire `id` de la relation `salle`.

Les attributs de la relation **Salle** sont :

- **id** : identifiant, entier naturel non nul, s'incrémente automatiquement.
- **nom** : attribut non nul, unique, de type chaine de caractères.
- **nb_prises_reseau** : entier naturel.

Afin de mieux manipuler la base de données de l'institut, on proposera les classes Python suivantes :

— une classe `Logiciel` définie par :

(a) une méthode constructeur définissant les attributs
- `nom` pour le nom du logiciel
- `version` pour la version du logiciel
- `prix` pour son prix d'achat.

(b) une méthode **enregistrer(self)** permettant d'enregistrer un logiciel dans la table `Logiciel`.

(c) une méthode **chercherId(self)** permettant de chercher un logiciel dans la table `Logiciel` par son nom et sa version et retourne son identifiant.

(d) une méthode **__str__(self)** pour visualiser un logiciel (nom, version, prix).

— une classe `Poste` définie par :

(a) une méthode constructeur définissant les attributs
- `nom` pour le nom du poste.
- `liste_logiciels` pour la liste des logiciels installés sur un poste. Cette liste sera initialement vide.

(b) une méthode **chercherId(self, idSalle)** permettant de chercher un poste dans la table `Poste` par son nom et la salle où il est installé et retourne son identifiant.

(c) une méthode **installer (self, L, d , c)** qui permet d'installer un logiciel **L** sur un poste à une date **d** avec une licence **c**.

(d) une méthode **charger_liste_logiciels (self)** permettant de mettre à jour la `liste_logiciels` par les logiciels installés sur un poste à partir de la base de données.

(e) une méthode **__str__(self)** pour visualiser la liste des logiciels installés sur un poste.

— une classe `Salle` définie par :

(a) une méthode constructeur définissant les attributs
- `nom` pour le nom de la salle.
- `liste_postes` pour la liste des postes localisés dans une salle. Cette liste sera initialement vide.

(b) une méthode **chercherId(self)** permettant de chercher une salle dans la table `Salle` par son nom et retourne son identifiant.

(c) une méthode **installer (self, p)** qui permet d'ajouter un poste **p** à une salle.

(d) une méthode **charger_liste_postes (self)** permettant de mettre à jour la `liste_postes` par les postes installés dans une salle à partir de la base de données.

(e) une méthode **__str__(self)** pour visualiser les postes (nom, catégorie) d'une salle.

On supposera pour la suite des questions que l'institut a équipé 3 salles (`I101`, `I102` et `I323`) par 9 postes, 3 postes (`P1`, `P2` et `P3`) pour chacune des salles.
La base de données de l'institut se nomme «**logiciels.db**», on utilisera le module `SQLite` pour sa manipulation.

Questions :

1. Quelle est la requête `SQL` permettant de créer la table **Installer**.
2. Définir une fonction Python **executer** permettant de retourner le résultat d'exécution d'une requête `SQL` de type `SELECT`.
3. Définir la classe **Logiciel**.
4. Créer une fonction **liste_salles** permettant de retourner à partir de la table `Salle`, une liste contenant toutes les salles de l'institut.
5. Définir un objet **scipy** représentant le logiciel «`Scipy 2.3`» en sa version gratuite et l'enregistrer dans la base de données.
6. Enregistrer dans la base de données l'opération d'installation du **scipy** sur tous les postes de l'institut le 04/04/2022.

7. Expliquer brièvement le concept d'**encapsulation** en programmation orientée objet.
8. Changer la protée de l'attribut `prix` de la classe `Logiciel` du publique à privée. Réecrire seulement la/les instructions de la/les méthodes modifiées de la classe `Logiciel`.
9. Ajouter, à la classe `Logiciel`, une méthode publique nommée **getPrix(self)** permettant de retourner le prix d'achat d'un logiciel. Quelles sont les modifications à faire au niveau de la classe `Logiciel` pour garder la cohérence du code (Réecrire seulement la/les instructions modifiées de la classe `Logiciel`).
10. Ajouter, à la classe `Logiciel`, une méthode publique **setPrix(self, nouveauPrix)** permettant de modifier le prix d'un logiciel, dans la quelle on fait tous les traitements/contrôles nécessaires pour préserver l'intégritée des données. Réecrire seulement la/les instructions modifiées de la classe `Logiciel`.
11. Ajouter à la classe `Poste` une méthode **nb_logiciels(self)** permettant de calculer et retourner le nombre de logiciels installés sur un poste .
12. Quelle est la requête `SQL` permettant d'afficher les postes sur les quels le logiciel « Python » n'est pas installé.

Rappels des syntaxes en Python

Tableau à une dimension (liste)	`L=[1,2,3]`
Tableau à une dimension (vecteur)	`v=array([1,2,3])`
Tableau à deux dimensions (matrice)	`M=array(([1,2,3],[3,4,5]))`
Tableau de 0 (2 lignes, 3 colonnes)	`zeros((2,3),float)`
Tableau de 1 (2 lignes, 3 colonnes)	`ones((2,3),float)`
Matrice identité d'ordre 3	`eye(3)`
Produit matriciel $I \times J$	`dot(I,J)`
Produit de deux matrices (élément par élément)	`I*J`

Bon travail**.**