BANQUE PT
PHYSIQUE & TECHNOLOGIE

Epreuve d'Informatique et Modélisation de Systèmes Physiques

Durée 4 h

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, d'une part il le signale au chef de salle, d'autre part il le signale sur sa copie et poursuit sa composition en indiquant les raisons des initiatives qu'il est amené à prendre.

L'usage de calculatrices est interdit.

La **présentation**, la lisibilité, l'orthographe, la qualité de la **rédaction, la clarté et la précision** des raisonnements entreront pour une **part importante** dans **l'appréciation des copies**. En particulier, les résultats non justifiés ne seront pas pris en compte. Les candidats sont invités à encadrer les résultats de leurs calculs.

L'épreuve comporte différentes parties permettant de mobiliser les compétences du candidat en informatique et en modélisation.

- La partie modélisation est distribuée au début et à la fin du sujet (I-1, I-2 et I-8). Il est conseillé de ne pas y consacrer plus d'une heure et demie.
- La partie informatique est développée au cœur du sujet (de I-3 à I-7) et sa durée de traitement conseillée est de deux heures et demie.

En dernière partie du sujet (II), vous trouverez un aide-mémoire *Python* et les consignes générales à respecter pour les codes exigés.

Tout code fourni en réponse à une question doit être commenté de façon claire.
Les commentaires sont à placer dans le code, ou éventuellement dans la brève introduction qui le précède.

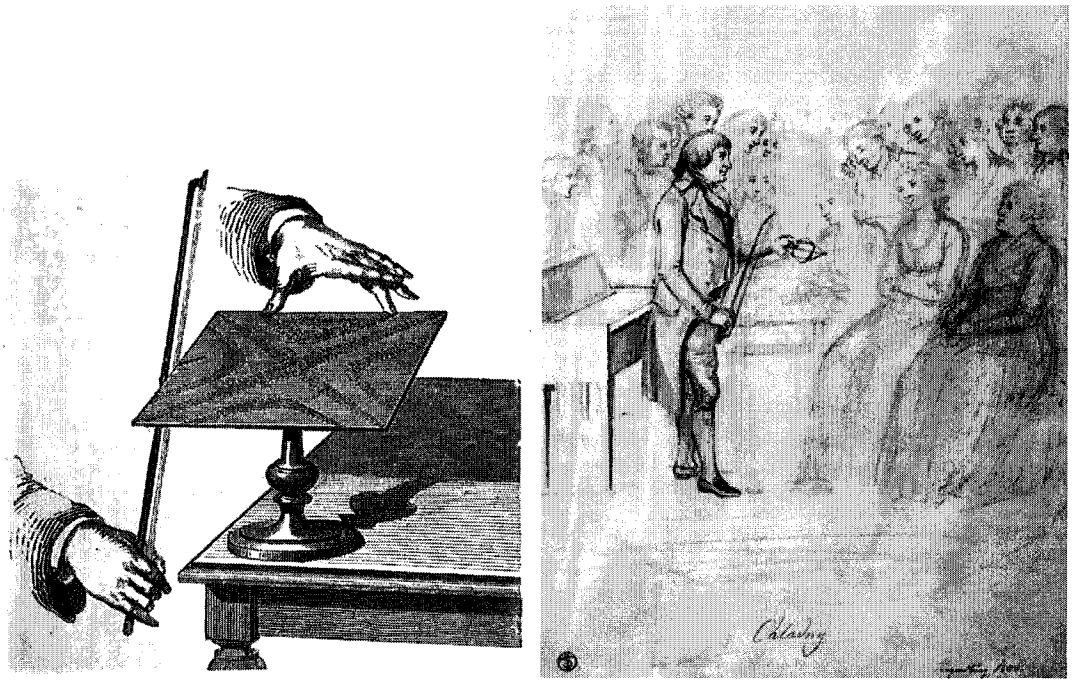
Le poids relatif des parties est indiqué dans le titre de chaque paragraphe.

Tournez la page S.V.P.

I Figures de Chladni

Introduction historique

En 1787, le physicien et musicien, Ernst Florence Friedrich Chladni de Wittenberg, alors qu'il procédait à de nombreuses expériences, fit une intéressante découverte. Il constata que lorsqu'il excitait une plaque de métal avec l'archet de son violon, il la faisait vibrer et il pouvait produire des sons de distinctes tonalités selon l'endroit où il touchait la plaque. La plaque métallique était fixée en son centre, il eut l'idée d'y disperser de la poussière. Lors des phases vibratoires, pour chaque tonalité, la poussière s'arrangeait et se distribuait selon les lignes nodales des vibrations, de magnifiques figures géométriques apparaissait alors. Expérimentateur chevronné, il prit soin de cataloguer et recenser ces différentes figures, après s'être assuré de leur caractère reproductif.



Dispositif et démonstration de Chladni

Ces figures, désormais dénommées figures de Chladni, en hommage à leur découvreur, attirèrent l'attention de nombreuses personnalités de l'époque. Scientifiques et puissants se pressaient aux nombreuses démonstrations du musicien pour admirer le phénomène. Sa compréhension échappait toutefois à l'entendement des contemporains de Chladni.

L'empereur Napoléon Bonaparte fut enthousiasmé par ces démonstrations permettant d'entendre et "voir" le son. En 1809, il invita l'académie des sciences à proposer un prix pour expliquer le phénomène et il finança la traduction en français du traité majeur d'acoustique de Chladni.

La première à proposer une explication pour ces observations fut la mathématicienne Sophie Germain dans une correspondance privée à partir de 1811. Elle publia quelques années plus tard un résumé de ces études qui constitua le premier modèle mathématique pour la déformation d'une plaque sous une contrainte de force extérieure, son travail "Recherche sur la Théorie des surfaces élastiques" sera couronné par l'Académie des Sciences en 1816. Lagrange et Poisson corrigèrent et améliorèrent ce premier modèle. Il fallut toutefois attendre l'intervention de Kirchoff pour aboutir à une modélisation permettant d'approcher de façon satisfaisante le comportement d'une plaque. Il en profita pour résoudre le problème des figures de Chladni sur une plaque circulaire où les nombreuses symétries permettent de réduire la complexité des solutions.

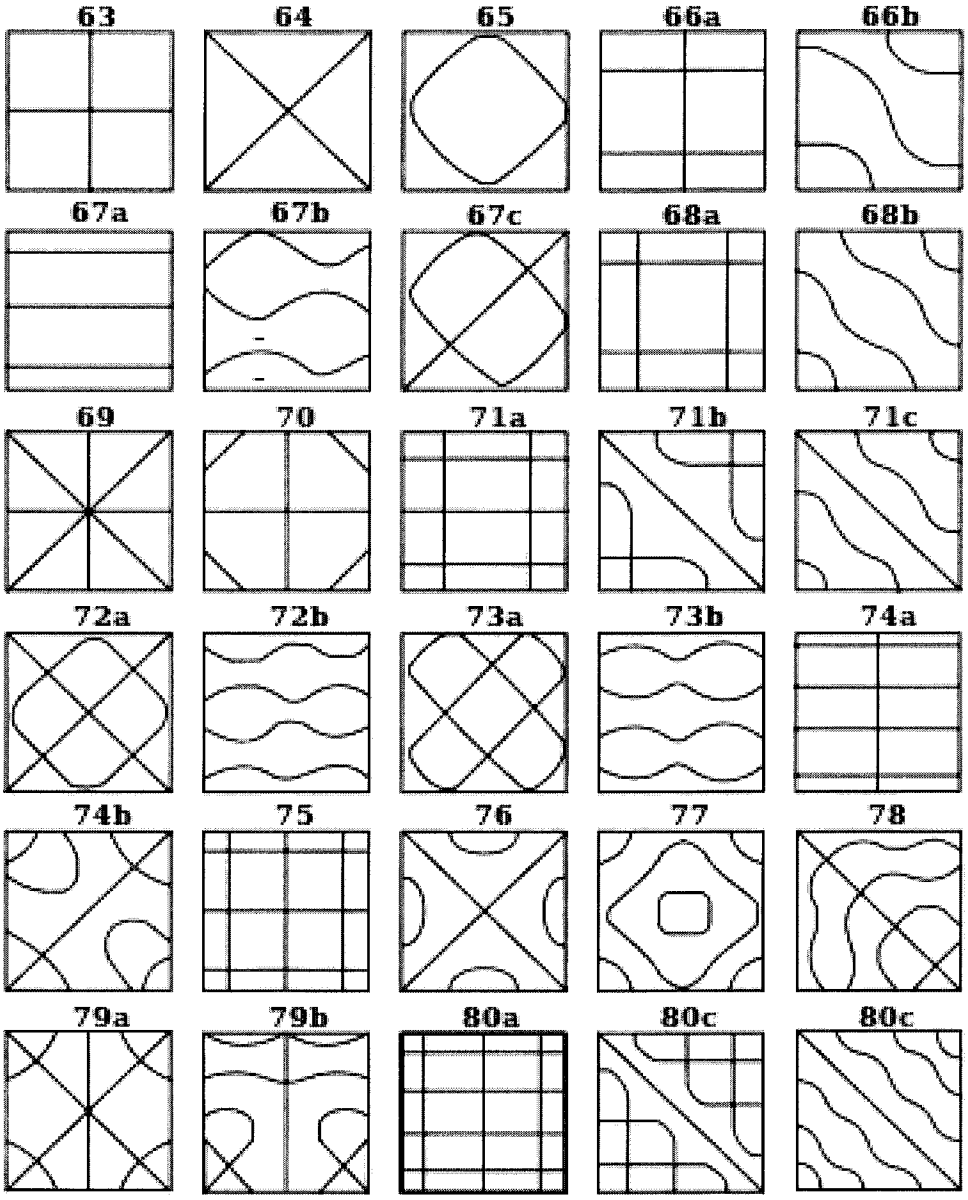


FIGURE 1 – Figures de Chladni

Á l'aube du vingtième siècle, l'expert sur la théorie des sons, John William Strut qui passa à la postérité en tant que Lord Rayleigh résuma la situation dans son traité majeur "La Théorie du Son" : *Le problème de la plaque rectangulaire dont les bords sont libres est d'une extrême difficulté et a pour l'essentiel résisté à toutes les tentatives de résolution.* Ce fut la spectaculaire invention de Walther Ritz qui permit en 1909 d'effectuer le premier calcul précis des vibrations d'une plaque carrée.

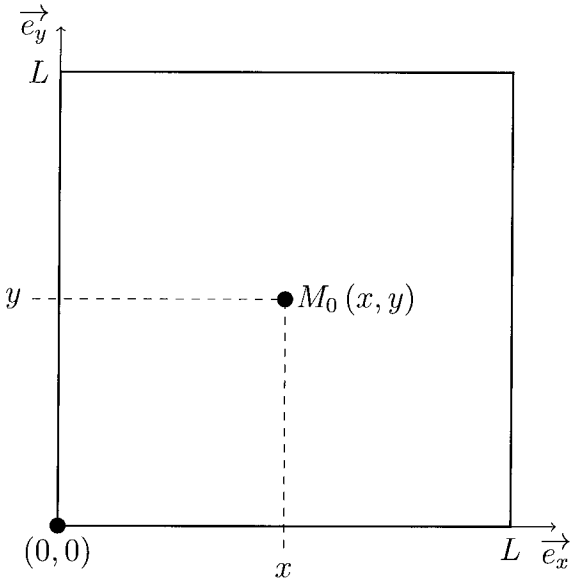
Nous nous proposons dans le problème suivant de reprendre en partie la démarche d'analyse du problème et de mettre en œuvre la conception du code en permettant une résolution numérique.

Tournez la page S.V.P.

I.1 Introduction du modèle physique ($\sim 10\%$)

Nous considérons une plaque métallique carrée de coté L, elle est d'épaisseur e, de module de Young E et sa masse volumique vaut ρ.

Pour suivre les éventuelles déformation de la plaque, nous utilisons un repère cartésien $\mathcal{R}\left(O, \overrightarrow{e_x}, \overrightarrow{e_y}, \overrightarrow{e_z}\right)$. Au repos, la plaque se situe dans un plan horizontal de cote $z = 0$ et chaque point de la plaque au repos est repéré par $M_0\left(x, y\right)$.



Repérage d'un point sur la plaque

Sous l'action d'une sollicitation extérieure, ou d'une contrainte, la plaque se déforme, chacun des points initialement en $M_0\left(x, y\right)$ est alors décrit par le point $M\left(x', y', z, t\right)$. M est une fonction du champ initial de position M_0 et du temps t.

Nous supposons que si les sollicitations sont modérées, nous aurons :

$$\forall t \quad x' = x \quad \text{et} \quad y' = y$$

Dans le cadre de cette approximation, le point M est à tout instant à la verticale de M_0 et sa seule variable d'espace indéterminée est son altitude z. Nous pouvons, dans ce cas, considérer que la position de M est décrite par la fonction :

$$z = f(x, y, t)$$

z est l'amplitude de vibration et $f(x, y, t)$ est la fonction d'onde de vibration.

Une analyse du problème utilisant une approche de déformation élastique, permet de considérer que la fonction d'onde f vérifie l'équation :

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \frac{1}{c^2} \cdot \frac{\partial^2 f}{\partial t^2} \tag{1}$$

où $c > 0$ est la célérité de l'onde.

1°) Quelle est le nom de cette équation d'onde ?
Quelles sont ces principales propriétés ?
Citez au moins deux situations physiques distinctes pour lesquelles il existe une équation analogue.

2°) Pensez-vous que cette équation soit le reflet d'un modèle physique autorisant des pertes d'énergie ? Justifiez votre réponse.

3°) Par analyse dimensionnelle, proposez une expression de c cohérente vis à vis des paramètres physiques de la plaque.

4°) Application numérique : masse volumique $\rho = 2{,}70 \cdot 10^3\,\mathrm{kg\,m^{-3}}$, épaisseur $e = 1{,}00\,\mathrm{mm}$ module de Young $E = 69\,\mathrm{GPa}$ et la longueur du coté de la plaque $L = 25{,}0\,\mathrm{cm}$.
Calculer c.

5°) Soit $f_L(x, y, t)$ une solution de l'équation (1) pour une plaque de coté L, $f_L(x, y, t)$ possède une caractéristique temporelle τ.
Nous considérons, dans cette seule question, une plaque faite du même matériau, possédant la même épaisseur, et soumises aux mêmes types de conditions aux limites, mais de coté $L' = \gamma \cdot L$.

a) Déterminer, parmi (α, β), la valeur du coefficient α qui permet à la fonction $g(x, y, t) = f_L(\alpha x, \alpha y, \beta t)$ d'être solution de l'équation (1) pour cette nouvelle plaque.

b) En déduire la caractéristique temporelle τ' correspondante en fonction de γ.

I.2 Modes de vibration ($\sim 5\%$)

En règle générale, les vibrations d'une plaque sont complexes et se décomposent en superposition de *modes*. Chacun de ces modes correspond à une vibration monochromatique (*d'une seule fréquence*).

6°) Pour résoudre l'équation (1) en $z(x, y, t)$, nous allons commencer par rechercher des solutions à variables séparables. C'est à dire des solutions qui seront de la forme :

$$z = f(x, y, t) = u(x, y) \times h(t)$$

a) En déduire les équations différentielles que doivent vérifier les fonctions $u(x, y)$ et $h(t)$.

b) Pourquoi les solutions acceptables pour $h(t)$ sont-elles seulement sinusoïdales ?

c) Les solutions $h(t)$ sont mises sous la forme $h(t) = h_0 \exp(i\omega t)$, exprimez alors l'équation vérifiée par $u(x, y)$. Différerait-elle si nous avions privilégié la forme $h(t) = A \cdot \cos(\omega t) + B \cdot \sin(\omega t)$?

Tournez la page S.V.P.

I.3 Simulation numérique temporelle ($\sim 20\%$)

L'équation différentielle que doit vérifier la fonction temporelle peut s'écrire après adimensionnement sous la forme :

$$\frac{d^2h}{dt^2}(t) = -h(t). \tag{2}$$

C'est une équation différentielle linéaire d'ordre deux classique appelée équation harmonique. Nous l'utiliserons pour tester la pertinence de certaines méthodes numériques.

7°) Méthode d'Euler

a) Rappeler, en quelques lignes concises, le principe de la méthode d'Euler pour résoudre une équation différentielle.

b) Mettre l'équation précédente sous la forme d'un système différentiel linéaire du premier ordre et le présenter sous forme matricielle en caractérisant la matrice A et en exploitant le vecteur $X(t)$ tel que :

$$\frac{dX}{dt}(t) = AX(t) \quad \text{avec} \quad X(t) = \begin{pmatrix} h(t) \\ h'(t) \end{pmatrix}$$

c) A partir de cette forme matricielle, introduire le pas de **discrétisation temporel** τ et mettre en œuvre la méthode d'Euler pour obtenir une équation aux différences correspondant au système initial.
Vous pourrez noter de façon réduite $X(t_n) = X[n]$

8°) Méthode d'Euler à droite.
Pour obtenir $X(t+\tau)$, nous pouvons procéder de la façon suivante :

$$X(t_n+\tau) = X(t_n) + \int_{t_n}^{t_n+\tau} \frac{dX(t)}{dt}\,\mathrm{d}t \approx X(t_n) + \tau \cdot \frac{dX(t_n+\tau)}{dt}$$

C'est la méthode d'Euler dite à *droite*, ou *implicite*, la version classique étant dite à *gauche*, ou *explicite*.

a) Exprimer l'équation liant dans ce cas $X[n+1]$ à $X[n]$.

b) Le code ci-après correspond à l'implémentation de la méthode d'Euler à gauche. Le modifier pour implémenter la méthode d'Euler à droite.

```
# Constantes paramétriques
npoints=2**10 # index maximal des tableaux
tporte=5*2*np.pi # durée d'affichage
X0=[1.,0] # conditions initiales

# Initialisation des tableaux
t=np.linspace(0,tporte,npoints+1) # temps
tau=t[1]-t[0]
X=np.zeros((2,npoints+1),dtype=float) # h et h'

# Algorithme d'Euler
X[:,0]=X0 #initialisation
M=np.array([[1,tau],[-tau,1]]) # matrice de l'equation
for i in range(npoints): # Calcul iteratif
        X[:,i+1]=M.dot(X[:,i])
```

c) La méthode d'Euler à droite est elle plus pertinente que celle à gauche ?

9°) Représentations graphiques

a) A partir du code donné à la question précédente, mettre en place les instructions pour obtenir l'affichage correspondant aux figures présentées ci-après.
Vous veillerez à la présence des légendes des courbes, des axes et des titres.

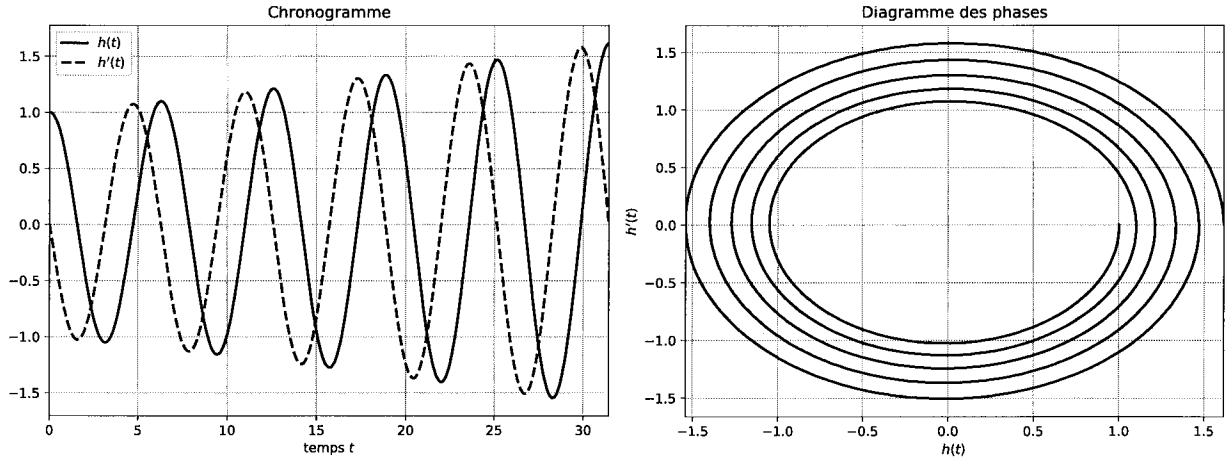


FIGURE 2 – Méthode d'Euler

b) Commentez les courbes obtenues et corrélez leurs allures aux propriétés de la méthode d'Euler et à l'équation (2).

Tournez la page S.V.P.

c) Sans modifier le principe de l'algorithme utilisé, nous pouvons néanmoins obtenir les courbes ci-après. Quelle est, selon vous, la modification apportée au code précédent ? Et quel en est l'inconvénient ?

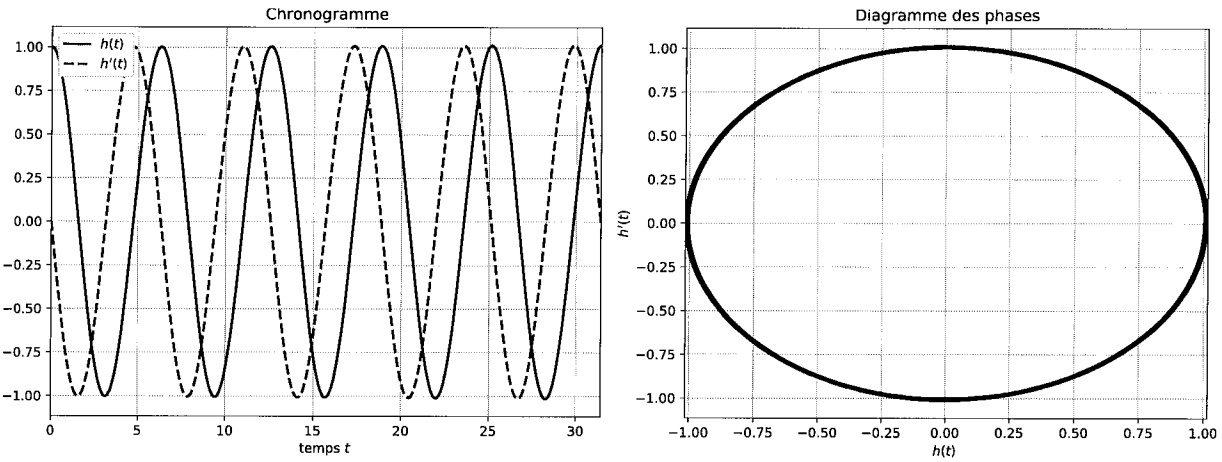


FIGURE 3 – Méthode d'Euler - version alternative

10°) Méthode de Runge-Kutta d'ordre deux (*aucune connaissance préalable n'est nécessaire*). $X(t+\tau)$ s'obtient exactement à partir de $X(t)$ avec la classique relation intégrale :

$$X(t_n+\tau) = X(t_n) + \int_{t_n}^{t_n+\tau} \frac{dX(t)}{dt}\,\mathrm{d}t$$

Nous pouvons obtenir une version approchée de cette relation en utilisant l'approximation :

$$X(t_n+\tau) \approx X(t_n) + \frac{\tau}{2}\cdot\left(\frac{dX(t_n+\tau)}{dt} + \frac{dX(t_n)}{dt}\right)$$

a) Quel est le nom de la méthode employée pour approximer l'intégrale ? Qualitativement, pourquoi est-elle préférable aux méthodes précédentes ?

b) L'analyse est faite sur la durée $T = N\tau$.
La valeur de la dérivée de X au temps $t_n+\tau$ demande la mise en œuvre d'une méthode de calcul implicite, nous l'éviterons en estimant la valeur de X au temps $t_n+\tau$ par la méthode d'Euler, c'est le principe de la méthode de Runge-Kutta. L'algorithme de calcul est alors le suivant :

$$K_1 = \frac{dX(t_n)}{dt} = A\cdot X[n] \qquad\qquad K_2 = A\cdot(X[n]+\tau K_1)$$

$$X[n+1] = X[n] + \frac{\tau}{2}\cdot(K_1+K_2)$$

Reprenez le code écrit pour la méthode d'Euler et adaptez le pour mettre en œuvre la méthode de Runge-Kutta.

c) La simulation numérique dans des conditions identiques à la méthode d'Euler nous fournit les graphes ci-après. Ils semblent correspondre aux attendus usuels.
Peut-on conclure à l'adéquation de la méthode de Runge-Kutta d'ordre deux ? Argumentez votre réponse.

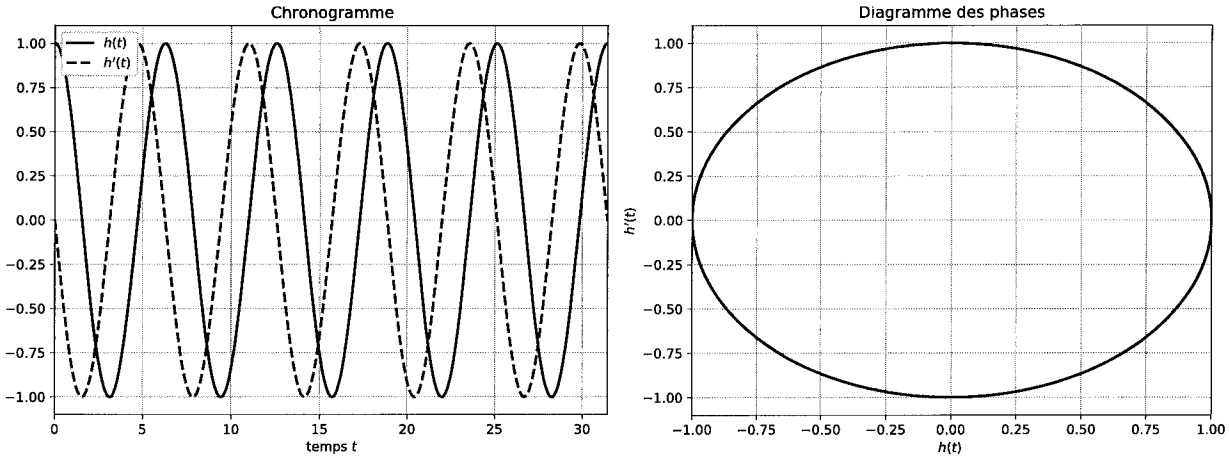


FIGURE 4 – Méthode de Runge-Kutta

11°) Méthode "stable".
Nous implémentons désormais le code ci-après :

```
# Constantes paramétriques
npoints=2**10 # index maximal des tableaux
tporte=100*2*np.pi # durée d'affichage
X0=[1.,0] # conditions initiales

# Initialisation des tableaux
t=np.linspace(0,tporte,npoints+1) # temps
tau=t[1]-t[0]
X=np.zeros((2,npoints+1),dtype=float) # h et h'

# Algorithme XXX
X[:,0]=X0 #initialisation
A=np.array([[0,1],[-1,-tau]]) # matrice de couplage
M=np.identity(2)+tau*A # matrice d'évolution
for i in range(npoints): # Calcul itératif
        X[:,i+1]=M.dot(X[:,i])

```

Il nous permet d'obtenir les courbes de la figure (5).

a) A partir du code déployé, donnez les équations itératives permettant la mise en œuvre de la simulation.

b) Quel est l'ordre de cette méthode ?
Pourquoi est elle "stable" ?

Tournez la page S.V.P.

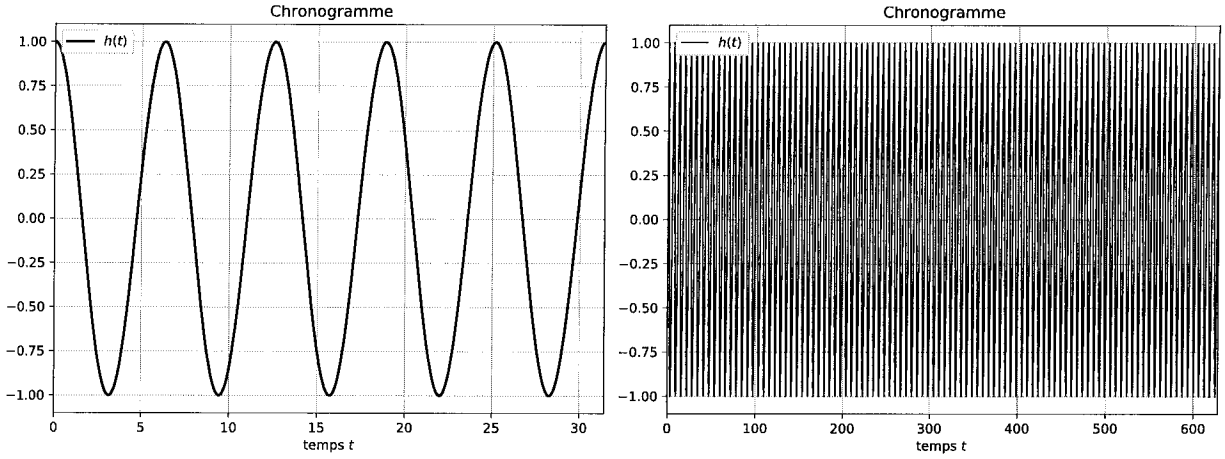


FIGURE 5 – Méthode "Stable"

I.4 Discrétisation spatiale du problème ($\sim 15\%$)

L'équation correspondant à un mode propre de pulsation ω peut s'écrire, pour $u(x, y)$ sous la forme :

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -\lambda \cdot u \tag{3}$$

où $\lambda > 0$ est la valeur propre associée au mode étudié.

Nous nous proposons de résoudre numériquement cette équation en utilisant une discrétisation et la méthode des différences finies. Nous introduisons, pour cela, un maillage de la plaque étudiée, de pas régulier, uniforme $d = \frac{L}{N}$ où N représente le nombre de segments du maillage.

12°) Équation adimensionnée

Nous allons effectuer le changement de variable ci-après pour faciliter le processus de discrétisation. Nous posons, pour une plaque de coté L :

$$X = \frac{x}{L} \times N, \quad Y = \frac{y}{L} \times N, \quad U(X, Y) = u(x, y)$$

L'équation vérifiée par $U(X, Y)$ se mettra sous la forme.

$$\Delta U = \frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2} = -\lambda' \cdot U \tag{4}$$

Déterminer l'expression de λ' en fonction de la pulsation ω.
Que devient le pas de discrétisation initial dans ce nouveau jeu de variables ?

Nous allons chercher à résoudre numériquement cette équation différentielle linéaire en la discrétisant avec le pas régulier unité. Nous découpons les axes X et Y en sous intervalles délimités par des ensembles de points définis de la manière suivante :

$$\forall i \in [0, N] \quad X_i = i$$

$$\forall j \in [0, N] \quad Y_i = j$$

La plaque sera représentée par un ensemble de points, distribués comme ci-après si $N > 4$.

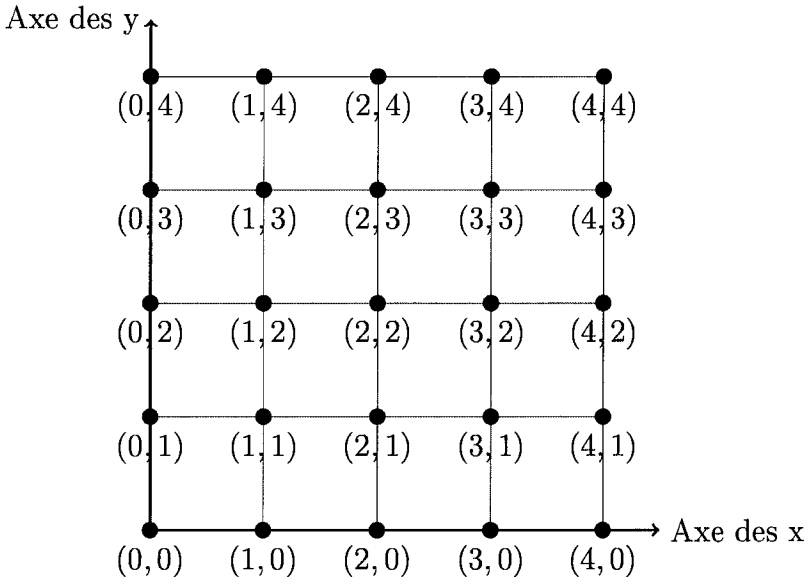


FIGURE 6 – Discrétisation de la plaque

Nous noterons, selon la convention suivante, la valeur approchée de $U(X,Y)$ correspondant au point de coordonnées (X_i, Y_j) :

$$U(X_i, Y_j) \approx U_{i,j}$$

13°) *Discrétisation autour d'un point central*
Nous étudions la situation d'un point du maillage loin des bords. Nous supposons que ce dernier de cordonnées (X_i, Y_j) possède ses huit plus proches voisins comme indiqué sur la figure (7).

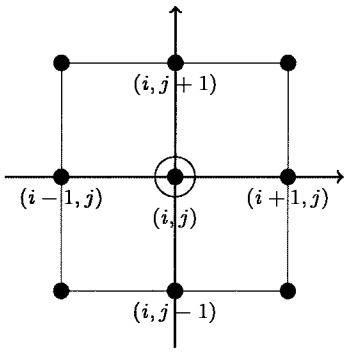


FIGURE 7 – Points voisins d'un point commun de coordonnées (i, j)

a) Exprimer la relation liant $U(X_{i-1}, Y_j)$ à $U(X_i, Y_j)$ par un développement limité à l'ordre deux.

b) Faire de même avec tous les points cardinaux du maillage et démontrer la formule approchée suivante :

$$\Delta U = \frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2} \approx U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4 \cdot U_{i,j} = -\lambda' \cdot U_{i,j}$$

Tournez la page S.V.P.

14°) *Discrétisation autour des bords*

Les bords de la plaque doivent être considérés différemment selon les conditions aux limites que nous désirons imposer. Il nous faut gérer les configurations types suivantes que nous désignerons respectivement par les termes de côté vertical, côté horizontal et de coins.

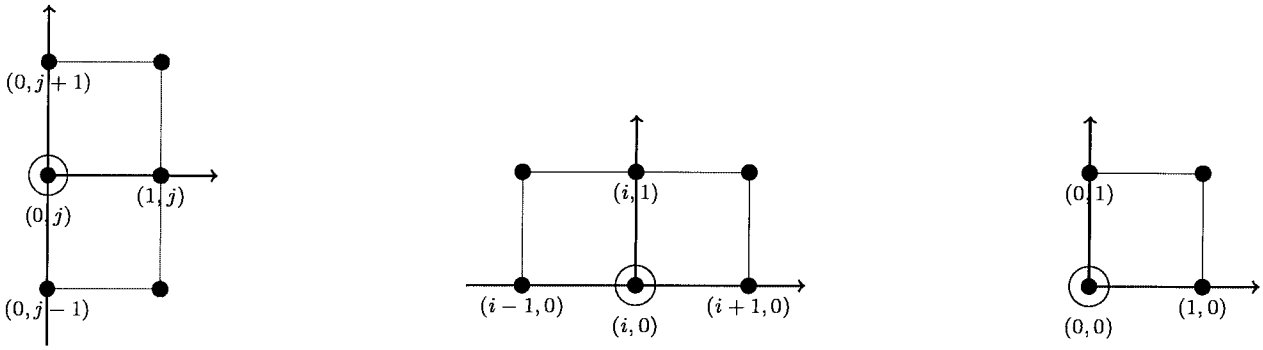


FIGURE 8 – Figuration des effets de bord

Les conditions aux limites classiques sont de deux types : soit les bords sont considérés comme fixes, soit ils sont considérés comme libres.
Dans notre cas, nous allons considérer que les points de la périphérie doivent être traités comme des points centraux, en associant aux points manquants de la distribution des valeurs nulles.

a) En déduire la formule approchée de (4) pour le côté vertical (X_0, Y_j)

b) En déduire la formule approchée de (4) pour le côté horizontal (X_i, Y_0)

c) En déduire la formule approchée de (4) pour le coin (X_0, Y_0)

d) Quelles sont les conditions aux limites retenues dans ce cas ?
Justifiez votre réponse.

15°) *Détermination de l'équation matricielle équivalente*

Nous introduisons le vecteur F_i tel que : $\forall\, i \in [0 \cdots N] \quad F_i = \begin{pmatrix} U_{i,0} \\ \vdots \\ U_{i,N} \end{pmatrix}$

Nous utilisons l'ensemble des vecteurs F_i pour générer le vecteur d'état F de l'ensemble de la plaque :

$$F = \begin{pmatrix} F_0 \\ \vdots \\ F_N \end{pmatrix}$$

Le système d'équations précédentes peut alors se mettre sous la forme d'une équation matricielle :

$$MF = \lambda' F \tag{5}$$

Les vecteurs F solutions de ce système sont les vecteurs propres de la matrice M, les λ' correspondants sont les valeurs propres associées.

a) Combien y-a-t-il de modes propres possibles ?

b) Nous nous plaçons dans le cas où $N = 4$ (cf figure 6), déterminer la matrice M.

c) *Généralisation*
Dans le cas ou $N > 4$, la matrice M peut se mettre sous la forme suivante où I_N est la matrice identité de trace $N+1$, et 0_N la matrice associée nulle de même dimension.

$$M = \begin{pmatrix} A & -I_N & 0_N & \cdots & 0_N \\ -I_N & A & \ddots & \ddots & \vdots \\ 0_N & \ddots & A & \ddots & 0_N \\ \vdots & \ddots & \ddots & A & -I_N \\ 0_N & \cdots & 0_N & -I_N & A \end{pmatrix}$$

Déterminer l'expression de A.

d) La matrice M est symétrique, à coefficients réels, elle est, d'après le *Théorème spectral*, diagonalisable. C'est une matrice tridiagonale par blocs. Nous admettrons de plus qu'aucune de ces valeurs propres n'est nulle.
Nous supposons que le code qui génère la matrice A sous la forme d'un tableau 2D de flottants existe, et a été produit par d'autres.

Construire la fonction *genereM* recevant A en entrée et renvoyant M.

```
def genereM(A):
    """Fonction dont l'objectif est de renvoyer M
    à partir de A"""
    ...
    return M # renvoie la matrice M
```

I.5 Méthode de la puissance itérée ($\sim 10\%$)

Nous allons chercher l'une des solutions de l'équation aux valeurs propres (5) en utilisant la méthode dite de la *puissance itérée*. Soit $\mathcal{M}_{n,p}(\mathbb{R})$ l'ensemble des matrices de taille $n \times p$, et $\mathcal{M}_n(\mathbb{R}) = \mathcal{M}_{n,n}(\mathbb{R})$ sa restriction à l'ensemble des matrices carrées.

16°) Á chaque matrice $M = (m_{ij}) \in \mathcal{M}_{n,p}(\mathbb{R})$, nous associons un nombre réel positif appelé norme de M et défini par :

$$\|M\| = \max |m_{ij}| \quad \forall i, j \in [0 \cdots n-1, 0 \cdots p-1]$$

a) Écrire en python la fonction *normeM* qui reçoit en entrée une matrice M de taille quelconque et renvoie la *norme* de M.
Le recours à une fonction max native de python est interdit pour cette question.

```
def normeM(M):
    """Fonction dont l'objectif est de renvoyer
    la norme de M à partir de M"""
    ...
    return r # norme de M
```

Tournez la page S.V.P.

b) Écrire en python la fonction *normevecteur* qui reçoit en entrée un vecteur (matrice colonne) F non nul, de taille quelconque, et renvoie un vecteur de même forme F^* défini par :

$$F^* = \frac{F}{\|F\|}$$

Le recours à une fonction max native de python est toujours interdit pour cette question.

```
def normevecteur(F):
        ...
        return Fn # vecteur normé
```

17°) Soit une matrice carrée $M \in \mathcal{M}_n(\mathbb{R})$, et soit F_0 un vecteur aléatoire de $\mathbb{R}^n$. Nous formons la suite $(F_p)_{p\geq 0}$ d'éléments de $\mathbb{R}^n$ définie par la relation de récurrence :

$$F_{p+1} = \frac{M \cdot F_p}{\|M \cdot F_p\|}$$

a) Écrire une fonction *puissanceiter* qui à partir d'une matrice carrée M et un entier p, choisit aléatoirement un vecteur F_0 de dimension adéquate, puis calcule et renvoie le $p^{\text{ième}}$ terme de la suite F_p.

```
def puissanceiter(M,p):
        ...
        return Fp # pième vecteur
```

b) Nous admettrons que la suite F_p converge dans certaines conditions vers le vecteur propre principal associé à la valeur propre maximale en valeur absolue.
Nous allons donc itérer le processus de calcul précédent jusqu'à ce que la suite F_p se stabilise. Soit $e_r > 0$ la valeur de seuil choisie pour la stabilité, il nous faut calculer les termes de la suite F_p jusqu'à ce que $\|F_p - F_{p-1}\| < e_r$.

Écrire le code de la fonction *iterstabilise* qui à partir d'une matrice carrée M et un réel $e_r > 0$, choisit aléatoirement un vecteur F_0 de dimension adéquate, puis calcule et renvoie le premier terme vérifiant la condition de stabilité de la suite F_p.

```
def iterstabilise(M,er):
        ...
        return F # vecteur stabilisé
```

c) Pour produire le code de la question précédente, il vous faudra utiliser une boucle conditionnelle.
Ce code va donc posséder un *variant* et un *invariant* de boucle. Après avoir rappelé les définitions de ces deux grandeurs, identifiez les dans le code que vous avez produit.

18°) Vecteur et valeur propre principaux

a) Le vecteur F renvoyé par la fonction est alors le vecteur propre principal de M. Écrire le code python pour obtenir la valeur propre associée λ'.

b) En déduire l'expression de la fréquence propre en Hertz du mode correspondant en fonction de λ' .

I.6 Représentation graphique ($\sim 5\%$)

Nous supposons que, par des méthodes numériques dont nous ne donnerons pas ici le détail, nous obtenons la suite des vecteurs propres et valeurs propres correspondants aux modes propres. Ces valeurs sont stockées dans deux tableaux de taille notée $n+1$.

— Le tableau des valeurs propres est un tableau 1D où les valeurs propres sont ordonnées par ordre croissant, il est nommé *valprope*.

— Le tableau des vecteur propres est un tableau 2D où les vecteurs propres sont stockés selon l'indexation des valeurs propres, il est nommé *vecteurprope*. Le $i^{\text{ième}}$ vecteur propre s'obtient par $vecteurpropre[i,:]$.

```
#librairies complémentaires
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

def graphe(Z):
        n=Z.shape[0]
        X=Y = np.arange(n)
        X, Y = np.meshgrid(X, Y)

        fig = plt.figure(0,figsize=(10,8))
        ax = fig.gca(projection='3d')
        surf = ax.plot_surface(X, Y, Z,rstride=1,cstride=1,
                cmap=cm.spectral_r, linewidth=0, antialiased=False)
        ax.set_zlim(-1, 1)
        fig.colorbar(surf, shrink=0.5, aspect=10)
        plt.tight_layout()

        fig1=plt.figure(1,figsize=(6,6))
        cont=plt.contour(Z,np.arange(21)/10-1,cmap=cm.spectral_r)
        plt.clabel(cont, cont.levels, inline=True,
                fmt='%1.1f', fontsize=10)
        plt.xlabel('x')
        plt.ylabel('y')
        plt.title('Courbes de niveau')
        plt.tight_layout()
        plt.show()
```

Tournez la page S.V.P.

19°) Le code ci-dessus permet la représentation des courbes de la figure (9). Pour fonctionner, il attend la matrice Z des valeurs U_{ij}.

$$Z = \begin{pmatrix} U_{00} & \cdots & U_{0j} & \cdots & U_{0n} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ U_{i0} & \cdots & U_{ij} & \cdots & U_{in} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ U_{n0} & \cdots & U_{nj} & \cdots & U_{nn} \end{pmatrix}$$

Il faut la fabriquer à partir du vecteur propre de type F (définition en Q15).

Écrire la fonction *transformeF* qui à partir d'un vecteur propre F renvoie la matrice U.

```
def transforme(F):
    ...
    return Z # matrice des valeurs U
```

20°) Vous trouverez en page suivante (figure 9), quelques uns des modes de résonance de la plaque. Commenter ces courbes et corréler leurs tracés aux choix de modélisation effectués.

I.7 Distribution des valeurs propres ($\sim$ 15%)

Nous disposons du tableau des valeurs propres. C'est un tableau 1D *Numpy* où les valeurs propres sont ordonnées par ordre croissant, il est nommé *valpropre*.

21°) Nous cherchons à savoir si une valeur se trouve dans les valeurs propres de la plaque.
Le recours à une fonction ***in*** *native de python est interdit pour cette question.*

a) Si le tableau n'était pas ordonné, quelle serait la complexité asymptotique d'une telle recherche ?

b) Écrire le code permettant une telle recherche pour la valeur v, tolérant un écart e_r de résolution et renvoyant un booléen comme résultat.

```
def recherchebrute(valpropre,v,er):
    ...
    return r # booléen
```

c) Le tableau étant ordonné, une recherche par dichotomie est préférable.
Quelle est la complexité asymptotique d'une recherche dichotomique ?

d) Rappelez le détail d'un algorithme de dichotomie.

e) Écrire l'algorithme correspondant pour la valeur v.
Pour cette implémentation, vous pouvez ignorer l'erreur de précision et la considérer comme nulle.

```
def recherchedicho(valpropre,v):
    ...
    return r # booléen
```

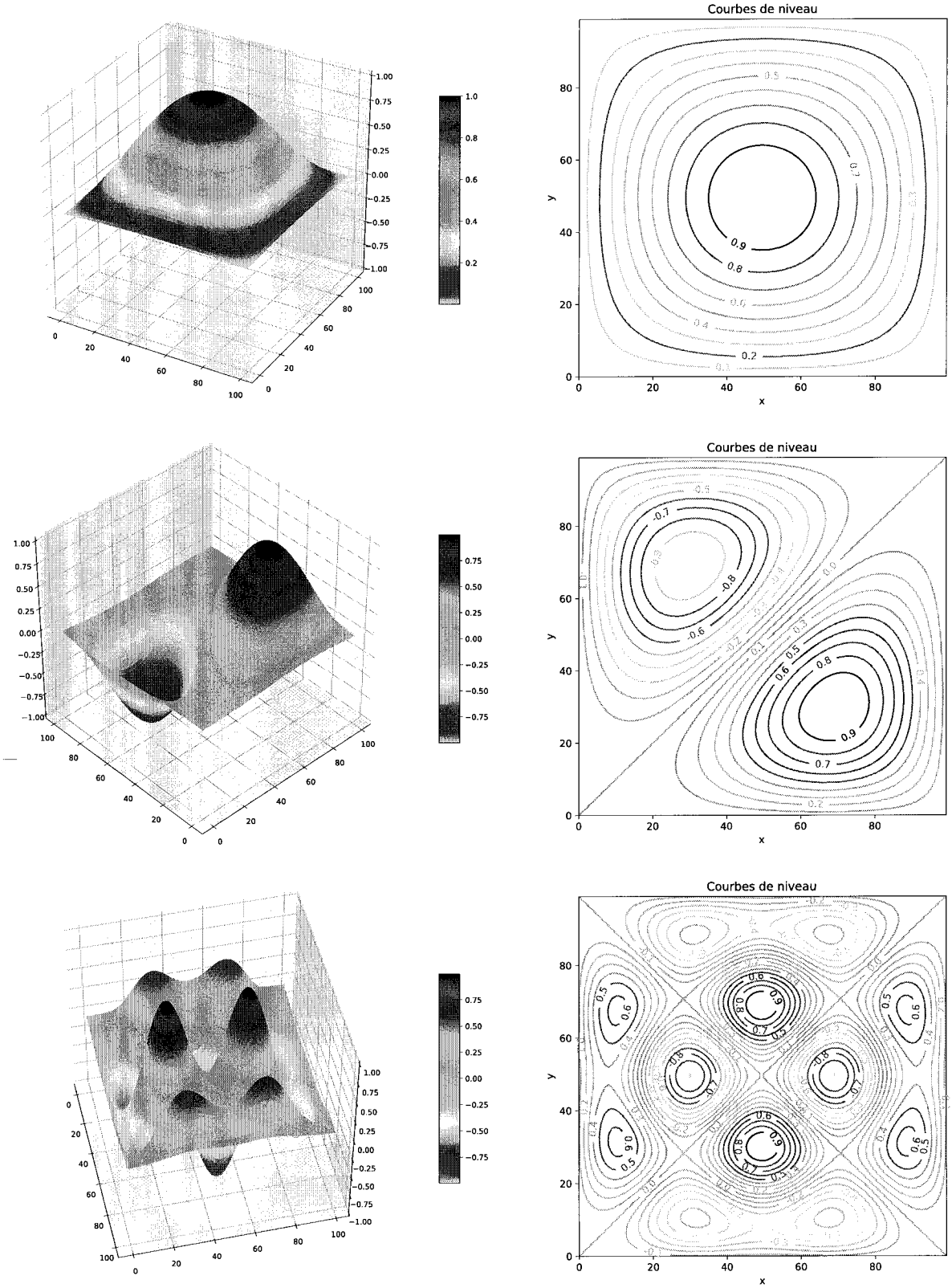


FIGURE 9 – Quelques modes propres

Tournez la page S.V.P.

22°) En pratique, ces valeurs propres correspondent aux fréquences de résonance de la plaque. Dans l'étude d'ouvrages, il est bon de savoir si dans une plage de fréquence donnée, la structure possède des valeurs propres.
Nous pouvons alors prévoir les risques de mise en résonance de l'ouvrage. Toute résonance engendrera des oscillations de fortes amplitudes qui risquent d'amener les matériaux à sortir de leur plage de linéarité, endommageant ainsi durablement l'ouvrage.

Par exemple, le **pont de la Basse-Chaîne** en 1850 à Angers, et le **pont de Broughton** en 1831 près de Manchester, s'effondrent sous le passage d'une troupe militaire avançant au pas cadençé. Plus connu, l'effondrement du **pont de Tacoma** en 1940 est associé, dans la culture populaire, à l'existence d'une résonance du pont sous l'action des vents. L'exemple contemporain le plus proche est celui du **pont piéton du Millénium Bridge**. Inauguré en 2000 à Londres, il dut fermer deux jours plus tard en raison d'un phénomène de résonance provoqué par la marche des piétons.

a) Nous désirons disposer d'une fonction nous renvoyant sous forme de tableau 1D *Numpy* toutes les valeurs propres de la plaque dans un intervalle délimité ouvert par *valmin* et *valmax*. Écrire le code permettant une telle recherche et utilisant la dichotomie de la question précédente.

```
def rechercheplagedicho(valpropre,valmin,valmax):
    ...
    return valselect # tableau1D
```

b) Un habitué de la librairie *Numpy* écrit le code python suivant qui renvoie le tableau attendu. Quel est son principe de fonctionnement ? Commentez chacun des ordres présentés.

```
1 def valsearch(valpropre,valmin,valmax):
2     mask=(valpropre>valmin) * (valpropre<valmax)
3     return valpropre[mask]
```

c) Quelle est la complexité asymptotique de ce code ?

d) En pratique, ce code s'avère près d'une centaine de fois plus rapide que celui de dichotomie. Comment pouvez-vous expliquer ce constat ?

I.8 Confrontation du modèle aux expériences ($\sim 20\%$)

Nous réalisons un dispositif expérimental permettant de confronter nos simulations à nos mesures. Nous considérons une plaque métallique uniforme et carrée sur laquelle nous disposons de fins grains de sable. La plaque est excitée par une onde acoustique émise par un haut parleur proche, pour certaines fréquences, la plaque entre en résonance avec l'émetteur sonore.

Nous pouvons alors distinguer, sur la plaque, la présence stable de ventres où l'amplitude de vibration est maximale, et la présence de nœuds où elle est nulle. Les grains sont expulsés des ventres et se concentrent à proximité des nœuds. Ces nœuds forment des courbes dites *lignes nodales*. Ce sont celles qu'a relevé Chladni et dont vous trouvez quelques exemples sur la figure 1. Nous avons relevé quelques unes des fréquences liées à ces résonances et nous les avons associées aux valeurs propres de l'équation (4) dans le tableau ci-après.

Nombre	1	2	3	4	5	6	7	8	9	10	11	12	13
valeur propre en ratio $\left(\frac{\lambda}{c_0}\right)$	2	5	5	8	10	10	13	13	17	17	18	20	20
fréquence en ratio $\left(\frac{\nu}{\nu_1}\right)$	1	2,5	2,5	4	5	5	6,5	6,5	8,5	8,5	9	10	10

Pour éviter toute difficulté de calcul, les valeurs propres λ ont été divisées par une constante dimensionnée c_0 dont la valeur n'a pas à être connue. Il en est de même pour les fréquences relevées qui sont rapportées à la première fréquence de résonance détectée qui est de l'ordre de la dizaine de *Hertz*.

L'analyse, par transformée de *Fourier*, des courbes obtenues nous montre que les solutions spatiales caractérisées par la simulation numérique sont des familles de fonctions $u(x, y)$ du type ci-après :

$$u(x, y) = C \sin\left(m\pi \frac{x}{L}\right) \sin\left(n\pi \frac{y}{L}\right)$$

où $m, n \in \mathbb{N}^*$ sont des entiers caractérisant le mode mn.

23°) Modèle initial. Il repose sur l'équation d'onde (1).

a) En considérant les formes d'ondes introduites, établir l'équation de dispersion liant la fréquence temporelle de l'onde ν aux entiers m, n.

b) Application numérique.
Déterminer l'ordre de grandeur de la fréquence de résonance la plus basse ν_1.
Est-il compatible avec celui donné par l'expérience ?

c) La distribution des premières fréquences de résonance est elle conforme à celle prévue par l'équation de dispersion ? (Une réponse argumentée est attendue)

d) Que pouvez-vous en conclure sur la pertinence du modèle utilisé ?

24°) Un improbable modèle.
Un étudiant suggère d'utiliser le modèle régit par l'équation d'onde ci-après :

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \alpha_1 \cdot \frac{\partial f}{\partial t}$$

a) Expliquez en quelques lignes pourquoi ce modèle est inadéquat.

b) Quelle est la dimension de la constante α_1.

Tournez la page S.V.P.

Figures de Chladni

25°) Un autre modèle.

Un étudiant suggère d'utiliser le modèle régit par l'équation d'onde ci-après :

$$\Delta\left(\Delta f\right) = \nabla^2\left(\nabla^2 f\right) = \nabla^4 f = \alpha_2 \cdot \frac{\partial^2 f}{\partial t^2}$$

a) Quelle est la dimension de la constante α_2.

b) Par analyse dimensionnelle, proposez une expression de la constante α_2.

c) Quelle est l'équation de dispersion correspondant à cette équation d'onde pour les formes d'ondes introduites dans cette section ?

d) La distribution des fréquences obtenue expérimentalement est elle conforme à celle prévue par cette nouvelle équation de dispersion ?

e) Supposons que ce modèle soit plus pertinent que celui décrit par l'équation (1), montrer que, même dans ce cas, la recherche des solutions numériques de l'équation (3) reste adéquate.

26°) Dégénérescence des modes.

a) Dans le tableau des relevés expérimentaux, nous constatons la présence de fréquences identiques, comment pouvez vous justifier ces dernières ?

b) La prise en compte des fréquences d'index 2 et 3 nous permet de proposer une solution $z(x, y, t)$ sous la forme :

$$z(x,y,t) = C\left(\sin\left(\pi\frac{x}{L}\right)\sin\left(2\pi\frac{y}{L}\right) + \sin\left(2\pi\frac{x}{L}\right)\sin\left(\pi\frac{y}{L}\right)\right)\cos(\omega t)$$

Déterminer l'équation de la ligne nodale correspondante.

II Annexe : Rappel Python

L'objectif de cette épreuve est de modéliser des systèmes, et d'assurer, par le biais d'une approche numérique, la résolution des problématiques posées. L'étude des figures de Chladni nous amène à simuler numériquement la réponse d'une plaque métallique. Notre approche se situe dans le cadre de l'ingénierie numérique du programme et conformément aux recommandations évoquées dans ce dernier nous privilégierons l'usage de python avec les bibliothèques *Numpy*.

Rappel du programme officiel - extrait Ingénierie numérique et simulation :

...

L'objectif est de familiariser les étudiants avec un environnement de simulation numérique. Cet environnement doit permettre d'utiliser des bibliothèques de calcul numérique et leur documentation pour développer et exécuter des programmes numériques. On veillera à faire aussi programmer par les étudiants les algorithmes étudiés. Aucune connaissance des fonctions des bibliothèques n'est exigible des étudiants. Au moment de l'élaboration de ces programmes d'enseignement, l'atelier logiciel Scilab ou le langage de programmation Python, avec les bibliothèques Numpy/Scipy, sont les environnements choisis.

...

Scilab n'a pas été retenu pour cette épreuve et nous nous limitons au langage *Python*.

En accord avec les exigences officielles, les fonctions et procédures spécifiques de la bibliothèque *Numpy*, dont nous avons l'usage, sont rappelées dans la présente annexe.

Recommandations :
Les fonctions attendues dans le sujet n'utiliseront pas la récursivité.
Elles ne doivent disposer dans leur code d'implémentation que d'un seul **return**.
Les éventuelles procédures n'en auront, bien sur, aucun.

II.1 Librairies utilisées

Nous partons du principe que tous les programmes demandés sont précédés des appels suivants :

```
# -*-coding:Utf-8 -*

#Librairies utilisées et alias
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rd
```

Les codes produits s'inspireront de ces quelques lignes pour l'exploitation des alias facilitant l'usage des fonctions et procédures des librairies appelées.

Tournez la page S.V.P.

II.2 Usage courant de *Numpy*

La librairie *Numpy* privilégie l'usage du type tableau. Ces derniers seront ici essentiellement des tableaux à une dimension ou deux dimensions. A l'occasion, les premiers seront parfois désignés comme des vecteurs, et les seconds comme des matrices.

Nous n'utiliserons toutefois pas le type spécifique python *Matrix*.

```
## commandes de base

# définir un tableau nul de taille nxp
A=np.zeros(shape=(n,p))

# définir un tableau identité de taille nxn
B=np.eye(n)

# définir un tableau rempli de 1 uniformément nxp
C=np.ones(shape=(n,p))

# obtenir les dimensions d'un tableau A : nxp
n,p=A.shape # usage de l'attribut shape

# obtenir le nombre d'élements d'un tableau A
nb_elements=A.size # usage de l'attribut size
```

Les tableaux se prêtent à de nombreuses opérations arithmétiques. Par défaut, chaque opération est effectuée sur l'ensemble des cellules contenues dans le tableau.

```
## Opérations élémentaires sur les tableaux

# Addition d'un scalaire j à chaque terme du tableau
j=1
D=A+j

# multiplication par un scalaire k
k=2
D=k*D

# addition, soustration, produit terme à terme
AddBC=B+C
SubBC=B-C
ProdBC=B*C
```

Attention le produit de base est un produit terme à terme de chaque élément du tableau. Il ne correspond pas au produit matriciel dont certains ont coutume.

Pour effectuer un produit matriciel, il faut utiliser la méthode *dot*.

```
## Opérations "matricielles" courantes

# produit matriciel B x C
PmatBC=B.dot(C)

# Tranposée du tableau C
C.T

# Lecture de la cellule [i,j] du tableau C à 2 dimensions
i,j=1,2
r=C[i,j]
```

Nous rappelons à cette occasion que l'indexation d'un tableau 2D à $N \times N$ éléments couvre en python la distribution $(i, j) \in [0, 1, \cdots, N-1]$.

II.3 Manipulation et copies de tableaux

Pour assurer la copie complète d'un tableau, il faut utiliser la méthode copy. Le simple égal est, à l'instar des listes de base, l'occasion de définir des alias

```
# Alias
F = C # F et C référencent le même tableau

## Copie d'un tableau
F=C.copy() # F référence une copie du tableau C
```

la Librairie propose le redimensionnement d'un tableau avec la fonction *reshape*. Il faut bien sur que la forme visée ait le même nombre d'éléments que la forme initiale.

```
## Reformage

# un tableau peut être réorganisé par la méthode reshape
E=np.ones(9) # vecteur 1D de neuf 1
print(E)
[ 1.  1.  1.  1.  1.  1.  1.  1.  1.]

F=E.reshape(3,3) # reforme E en tableau 2D 3x3
print(F)
[[ 1.  1.  1.]
[ 1.  1.  1.]
[ 1.  1.  1.]]

G=F.flatten() # renvoie le tableau sous forme 1D
print(G)
[ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

Tournez la page S.V.P.

Numpy permet d'assembler les vecteurs et les matrices, de les concaténer en utilisant la fonction correspondante *concatenate*. Par défaut l'assemblage se fait selon la 1ère dimension (les lignes, donc assemblage vertical). L'option axis=1 assemble "horizontalement".

```
a=np.arange(4).reshape(2,2)
        array([[0, 1],
               [2, 3]])

b=4+np.arange(4).reshape(2,2)
        array([[4, 5],
               [6, 7]])

np.concatenate((a,b))
        array([[0, 1],
               [2, 3],
               [4, 5],
               [6, 7]])

np.concatenate((a,b),axis=0)
        array([[0, 1],
               [2, 3],
               [4, 5],
               [6, 7]])

np.concatenate((a,b),axis=1)
        array([[0, 1, 4, 5],
               [2, 3, 6, 7]])
```

Les plus coutumiers d'entre vous pourront utiliser les techniques de slicing et de masquage qui ne seront pas rappelées ici. Si la librairie favorise le traitement en masse, nous acceptons néanmoins tous les codes utilisant de simples boucles.

II.4 Utilisation du générateur aléatoire - *numpy.random*

Python dispose de plusieurs librairies permettant la génération aléatoire de nombres. Pour des raisons pratiques évidentes(vectorisation), nous exploitons ici la librairie random intégrée au sein de *Numpy*. Les utilisations correctes d'autres librairies, correctement introduites et exploitées seront bien sur acceptées.

La fonction *rand()* crée un tableau d'un format donné de réels aléatoires dans $[0, 1[$. La dimension du tableau est donnée par le paramètre passé dans le champ argument de la fonction, qui n'est autre que la forme *shape* du tableau renvoyé.

Exemple :

```
import numpy.random as rd # Appel de la librairie

# tableau 1D comportant 10 éléments
print(rd.rand(10))
>>> [0.16653536 0.37216705 0.56565704 0.62948842 0.65555812 0.38543034
 0.1486412  0.52610282 0.4686787  0.81558018]


# tableau 2D comportant 3X2 éléments
print(rd.rand(3,2))
>>> [[0.98992966 0.58907376]
 [0.6404654  0.53852321]
 [0.12416014 0.28279454]]
```

FIN DE L'ÉPREUVE