BANQUE PT 2019 : CORRIGE DE L'EPREUVE D'INFORMATIQUE ET MODELISATION DE SYSTEMES PHYSIQUES

Figures de Chladni

Deux **remarques importantes** concernant la partie informatique (parties 3 à 7) : à la fin du sujet, dans la partie annexe, des recommandations sont formulées. Même si elles ne sont pas forcément mises en valeur en étant formulées à cet endroit-là, il faut en tenir compte :

- 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**.

1. Introduction du modèle physique :

Question 1 :

- Il s'agit de l'équation de d'Alembert.
- Ces propriétés :
  - C'est une équation différentielle linéaire.
  - Pas d'effet diffusif.
  - Isotrope.
  - Pas de déformation de l'onde au cours de sa propagation.
  - C'est une équation de conservation dans l'espace-temps.
- Equation analogue :
  - Mouvement de la corde vibrante.
  - Ondes électromagnétiques dans le vide.

Question 2 :

Ce modèle n'autorise pas les pertes d'énergie car il n'y a pas de dérivée première par rapport au temps (contrairement à l'équation de la chaleur, à l'effet de peau, ou aux phénomènes diffusifs de manière générale).

Question 3 :

- Le module de Young E est homogène à une pression en Pa = $N.m^{-2}$ = kg $(m.s^{-2})$ m^{-2}
  Donc [E] : $M.L^{-1}.T^{-2}$
- La masse volumique ρ est en $kg.m^{-3}$
  Donc [ρ] : $M.L^{-3}$
- Donc $\frac{[E]}{[\rho]} = \frac{M.L^{-1}.T^{-2}}{M.L^{-3}} = L^2.T^{-2} = (L.T^{-1})^2 = [c]^2$, et donc $\boxed{c = \sqrt{\frac{E}{\rho}}}$

Question 4 :

$c = \sqrt{\frac{E}{\rho}} = \sqrt{\frac{69.10^9}{2{,}70.10^3}}$, et donc $\boxed{c = 5{,}1.10^3 \text{ m.s}^{-1}}$

Question 5 :

a) On pose $x' = \alpha x$; $y' = \alpha y$; $t' = \beta t$. Donc $g(x, y, t) = f_L(x', y', t')$.

$\frac{\partial g}{\partial x}(x, y, t) = \frac{\partial f_L}{\partial x'}(x', y', t') \times \frac{\partial x'}{\partial x} = \alpha \frac{\partial f_L}{\partial x'}(x', y', t')$ (car $\frac{\partial x'}{\partial x} = \alpha$).

En dérivant encore une fois : $\frac{\partial^2 g}{\partial x^2}(x, y, t) = \alpha^2 \frac{\partial^2 f_L}{\partial x'^2}(x', y', t')$.

De même, on a : $\frac{\partial^2 g}{\partial y^2}(x, y, t) = \alpha^2 \frac{\partial^2 f_L}{\partial y'^2}(x', y', t')$ et $\frac{\partial^2 g}{\partial t^2}(x, y, t) = \beta^2 \frac{\partial^2 f_L}{\partial t'^2}(x', y', t')$.

Par ailleurs, f_L est solution de l'équation (1), donc $\frac{\partial^2 f_L}{\partial x'^2}(x', y', t') + \frac{\partial^2 f_L}{\partial y'^2}(x', y', t') = \frac{1}{c^2} \frac{\partial^2 f_L}{\partial t'^2}(x', y', t')$.

Donc $\frac{1}{\alpha^2}\left(\frac{\partial^2 g}{\partial x^2}(x, y, t) + \frac{\partial^2 g}{\partial y^2}(x, y, t)\right) = \frac{1}{c^2} \frac{1}{\beta^2} \frac{\partial^2 g}{\partial t^2}(x, y, t)$, et donc g est solution de l'équation (1) si $\alpha^2 = \beta^2$, donc si $\boxed{\alpha = \beta}$.

Remarque : la solution mathématique $\alpha = -\beta$ n'a pas de sens physique : ici, les coordonnées de la plaque seront toujours positives (donc $\alpha > 0$), et une dilatation des distances engendrera une dilatation du temps pour arriver dans le même état, et pas une dilatation accompagnée d'une inversion du temps… (donc $\beta > 0$).

b) On a $\Delta f_L = \frac{1}{c^2} \frac{\partial^2 f_L}{\partial t^2}$. En ordre de grandeur : $\frac{f_L}{L^2} = \frac{1}{c^2} \frac{f_L}{\tau^2}$, donc $L = c\,\tau$.

De même, on a $\Delta g = \frac{1}{c^2} \frac{\partial^2 g}{\partial t^2}$. En ordre de grandeur : $\frac{g}{L'^2} = \frac{1}{c^2} \frac{g}{\tau'^2}$, donc $L' = c\,\tau'$.

Donc $\tau' = \frac{L'}{c} = \frac{\gamma L}{c} = \gamma\,\tau$. On a donc $\boxed{\tau' = \gamma\,\tau}$

Remarque : autre méthode (plus en accord avec a)) : Avec $\alpha = \beta = \gamma$, on a : $g(L', L', \tau') = f_L(\gamma L, \gamma L, \gamma\tau)$ avec $L' = \gamma L$, on obtient $\tau' = \gamma\tau$.

2. Modes de vibration :

Question 6 :

a) On a $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \times h(t)$.

Par ailleurs, $\frac{1}{c^2} \frac{\partial^2 f}{\partial t^2} = \frac{1}{c^2} u(x, y) \times h''(t)$.

Grâce à l'équation (1), on en déduit : $\frac{h''(t)}{h(t)} = c^2 \frac{1}{u(x,y)}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$.

Le terme de gauche est une fonction du temps, le terme de droite est une fonction de x et de y.
Or cette relation doit être vraie $\forall x, \forall y, \forall t$.

On en déduit : $\frac{h''(t)}{h(t)} = c^2 \frac{1}{u(x,y)}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = K$ (constante)

Donc $\boxed{h''(t) - K\,h(t) = 0}$ et $\boxed{\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{K}{c^2} u}$

b) On va faire une hypothèse sur le signe de K.

- Supposons K > 0. On a : $h''(t) - K\,h(t) = 0$, et alors les solutions sont : $h(t) = A_1 \exp\left(\sqrt{K}\,t\right) + A_2 \exp\left(-\sqrt{K}\,t\right)$.
  Or $\exp\left(\sqrt{K}\,t\right) \xrightarrow[t\to\infty]{} \infty$, ce qui n'a pas de sens physique. Donc $A_1 = 0$.
  Donc $h(t) = A_2 \exp\left(-\sqrt{K}\,t\right) \xrightarrow[t\to\infty]{} 0$, ce qui correspond à un régime transitoire… Or il n'y a pas de perte d'énergie si on considère l'équation de d'Alembert ! C'est donc impossible et donc $\boxed{K \le 0}$.
- Supposons K = 0. On a : $h''(t) = 0$, et alors les solutions sont : $h(t) = A_1 + A_2\,t$.
  Or $t \xrightarrow[t\to\infty]{} \infty$, ce qui n'a pas de sens physique. Donc $A_2 = 0$. Donc $h(t) = A_1 = c^{te} = 0$ (on s'intéresse à des ondes, donc à des déformations non constantes !). C'est donc impossible et donc $\boxed{K < 0}$.
- On pose alors $K = -\omega^2$.
  On a alors : $h''(t) + \omega^2 h(t) = 0$. On reconnait l'équation d'un oscillateur harmonique. Les seules solutions acceptables pour h(t) sont sinusoïdales. Donc $h(t) = A\cos(\omega t) + B\sin(\omega t)$.

c) Avec $h(t) = h_0 \exp(i\omega t)$, on a : $h''(t) = h_0 (i\omega)^2 \exp(i\omega t) = -\omega^2 h(t)$.

On a donc $h''(t) + \omega^2 h(t) = 0$, donc $K = -\omega^2$ (OK !).

On obtient : $\boxed{\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -\frac{\omega^2}{c^2} u}$

Cette équation ne différerait pas si nous avions privilégié la forme $h(t) = A\cos(\omega t) + B\sin(\omega t)$ car alors : $h''(t) = -A\omega^2\cos(\omega t) - B\omega^2\sin(\omega t) = -\omega^2 h(t) \Rightarrow$ idem !

3. Simulation numérique temporelle :

Question 7 : Méthode d'Euler :

a) On effectue une discrétisation temporelle (de pas $\tau = t_{n+1} - t_n$) afin d'obtenir une suite récurrente.

Pour cela, on effectue l'approximation suivante : $\frac{dx}{dt}(t_n) \approx \frac{x(t_{n+1}) - x(t_n)}{\tau}$

Ce qui revient à écrire : $x(t_{n+1}) = x(t_n) + \int_{t_n}^{t_{n+1}} \frac{dx(t)}{dt}\,dt \approx x(t_n) + \tau . \frac{dx}{dt}(t_n)$

Si $\frac{dx}{dt}(t) = f(y,t)$, alors on effectue l'approximation : $x(t_{n+1}) \approx x(t_n) + \tau . f(x(t_n), t_n)$.

Ainsi, on obtient $x(t_{n+1})$ en fonction de $x(t_n)$.

b) On pose : $X(t)=\begin{pmatrix} h(t) \\ h'(t) \end{pmatrix}$.

D'après l'équation différentielle (2), on a donc : $X'(t)=\begin{pmatrix} h'(t) \\ h''(t) \end{pmatrix}=\begin{pmatrix} h'(t) \\ -h(t) \end{pmatrix}=\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}\begin{pmatrix} h(t) \\ h'(t) \end{pmatrix}$.

Donc $X'(t)=A\,X(t)$ avec $\boxed{A=\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}}$

c) On met en œuvre la méthode d'Euler : $\boxed{X[n+1]\approx X[n]+\tau.\frac{dX}{dt}[n]}$

Donc $X[n+1]=X[n]+\tau A\,X[n]=(I_2+\tau A)X[n]$, donc $\boxed{X[n+1]=(I_2+\tau A)X[n]}$

Donc $X[n+1]=\left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}+\tau\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}\right)X[n]$, donc $\boxed{X[n+1]=\begin{pmatrix} 1 & \tau \\ -\tau & 1 \end{pmatrix}X[n]}$

Avec de plus : $X[0]=\begin{pmatrix} h(0) \\ h'(0) \end{pmatrix}=\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ d'après les conditions initiales fournies dans le code de la question 8) b).

Question 8 : Méthode d'Euler à droite :

a) On met en œuvre la méthode d'Euler implicite : $X[n+1]\approx X[n]+\tau.\frac{dX}{dt}[n+1]$

Donc $X[n+1]=X[n]+\tau A\,X[n+1]$

Donc $X[n]=(I_2-\tau A)X[n+1]=\left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}-\tau\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}\right)X[n+1]=\begin{pmatrix} 1 & -\tau \\ \tau & 1 \end{pmatrix}X[n+1]$

Or $(I_2-\tau A)=\begin{pmatrix} 1 & -\tau \\ \tau & 1 \end{pmatrix}$ est inversible (car de déterminant $1+\tau^2\neq 0$), donc $\boxed{X[n+1]=(I_2-\tau A)^{-1}\,X[n].}$

On peut montrer que $(I_2-\tau A)^{-1}=\begin{pmatrix} 1 & -\tau \\ \tau & 1 \end{pmatrix}^{-1}=\frac{1}{1+\tau^2}\begin{pmatrix} 1 & \tau \\ -\tau & 1 \end{pmatrix}$.

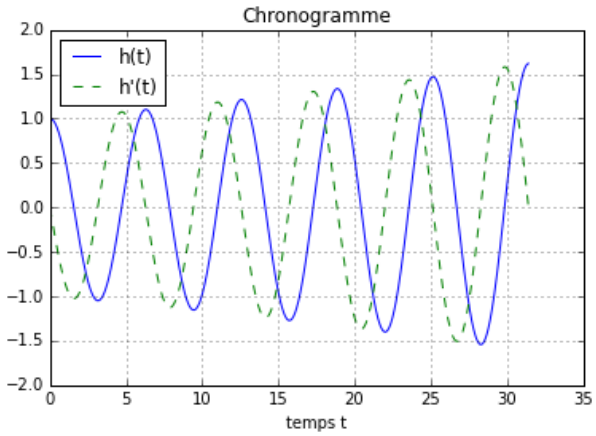
Donc $\boxed{X[n+1]=\frac{1}{1+\tau^2}\begin{pmatrix} 1 & \tau \\ -\tau & 1 \end{pmatrix}X[n]}$.

b) Il suffit de changer la ligne 13 : on la remplace par :

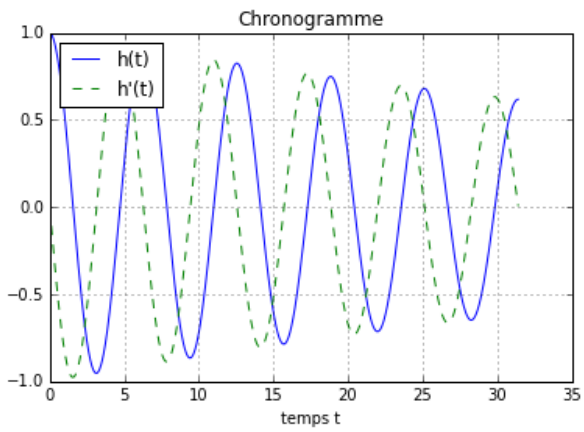
```
M = 1 / (1 + tau**2) * np.array([[1, tau], [-tau, 1]])  # matrice de l'équation
```

c) Il n'y a rien qui laisse penser a priori que la méthode d'Euler implicite (à droite) soit plus pertinente que la méthode d'Euler explicite (à gauche).
Toutefois, le schéma explicite est ici instable (cf le chronogramme de la figure 2, ou bien ci-après : l'amplitude des oscillations augmente au cours du temps, ce n'est pas stable). Le schéma implicite est quant à lui stable (cf le deuxième chronogramme ci-après : l'amplitude des oscillations diminue au cours du temps, c'est stable).
Toutefois, la méthode d'Euler implicite demande l'inversion d'une matrice, simple ici, mais ça peut s'avérer être plus coûteux.

Euler explicite :



Euler implicite :



Question 9 : Représentations graphiques :

a)

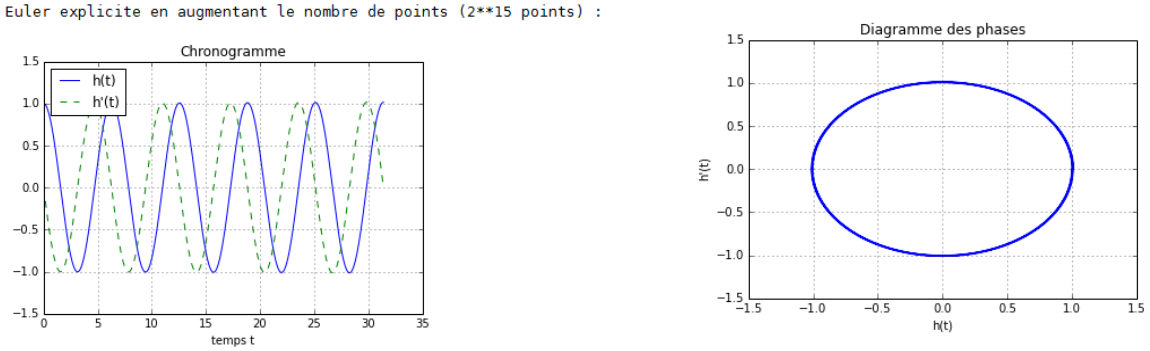
```
# Pour le chronogramme :
plt.figure()    # crée une fenêtre de tracé vide
plt.plot(t, X[0, :], '-', label = "h(t)")   # tracé de h(t)
plt.plot(t, X[1, :], '--', label = "h'(t)")     # tracé de h'(t)
plt.legend(loc='upper left')    # légendes + localisation des légendes
plt.grid(True)  # présence de la grille de fond
plt.xlabel('temps t')   # légende de l'axe des abscisses
plt.title('Chronogramme')   # titre
plt.show()  # affichage des différentes figures

# Pour le diagramme des phases :
plt.figure()    # crée une fenêtre de tracé vide
plt.plot(X[0, :], X[1, :])  # on trace h' en fonction de h
plt.grid(True)  # présence de la grille de fond
plt.xlabel("h(t)")   # légende de l'axe des abscisses
plt.ylabel("h'(t)")  # légende de l'axe des ordonnées
plt.title('Diagramme des phases')   # titre
plt.show()  # affichage de la figure
```

b) L'équation (2) est l'équation différentielle d'un oscillateur harmonique. On devrait donc avoir des oscillations sinusoïdales pour h(t) et h'(t), et une ellipse d'axes x et y pour le diagramme des phases.
Avec la méthode d'Euler explicite, on voit que l'amplitude des oscillations augmente au cours du temps (cf le chronogramme de la figure 2 : ce n'est pas stable).

Remarques hors-programme : On dira qu'un schéma est stable si la solution discrète reste bornée quel que soit le nombre de pas de discrétisation. La stabilité de la méthode est liée au comportement des puissances d'une matrice M donnée par la relation X[n+1] = M X[n], c'est-à-dire X[n] = M^n X[0].

- Pour la méthode d'Euler explicite, les valeurs propres complexes de $M = \begin{pmatrix} 1 & \tau \\ -\tau & 1 \end{pmatrix}$ sont $(1 \pm i\tau)$, de module $|1 \pm i\tau| = \sqrt{1+\tau^2} > 1$, donc le schéma est instable.
- Pour la méthode d'Euler implicite, les valeurs propres complexes de $M = \frac{1}{1+\tau^2}\begin{pmatrix} 1 & \tau \\ -\tau & 1 \end{pmatrix}$ sont $\frac{(1 \pm i\tau)}{1+\tau^2}$, de module $\frac{|1 \pm i\tau|}{1+\tau^2} = \frac{\sqrt{1+\tau^2}}{1+\tau^2} = \frac{1}{\sqrt{1+\tau^2}} < 1$, donc le schéma est stable ($M^n \xrightarrow[n\to\infty]{} 0$ et donc $X[n] \xrightarrow[n\to\infty]{} 0$, ce qui explique le chronogramme ci-avant).

c) Il suffit que le nombre de points soit suffisamment grand : il faut donc augmenter la valeur attribuée à npoints (à partir, en gros, de 2**15, on obtient des courbes ressemblant à celles de la figure 3).
Inconvénient : Augmenter le nombre de points va entrainer davantage de calculs à effectuer. La complexité temporelle sera donc augmentée.



Question 10 : Méthode de Runge-Kutta d'ordre deux :

a) La méthode employée pour approximer l'intégrale est la méthode des trapèzes. Elle est préférable aux méthodes précédentes (rectangle à gauche et rectangle à droite) car elle est plus précise. En effet, les méthodes des rectangles à gauche ou à droite (c'est-à-dire les méthodes d'Euler explicite ou implicite) sont d'ordre 1, alors que la méthode des trapèzes (c'est-à-dire la méthode de Runge-Kutta d'ordre 2) est d'ordre 2.

Remarque hors-programme (?) : Comment déterminer l'ordre d'une méthode ?

- Grâce à la formule de Taylor-Young, on a : $\forall n \left(0 \le n \le N\right)$, pour $\tau \to 0$:

$$X(t_{n+1}) = X(t_n + \tau) = X(t_n) + \tau\, X'(t_n) + \frac{\tau^2}{2} X''(t_n) + O(\tau^3)$$

$$X(t_{n+1}) = X(t_n) + \tau\, A\, X(t_n) + \frac{\tau^2}{2} A^2\, X(t_n) + O(\tau^3) \text{ car } X' = A\,X \text{ et } X'' = A\,X' = A^2\,X$$

- Dans les différentes méthodes (Euler explicite, implicite, Runge-Kutta 2, …), on construit une suite $(X[n])$ (telle que $X(t_n) \approx X[n]$, avec $X(t_n)$ la valeur exacte et $X[n]$ une valeur approchée), de telle sorte que par construction, on a : $X[n+1] = M\,X[n]$.
- Définition : On dit que la méthode numérique est d'ordre p (p entier) si $\forall n \left(0 \le n \le N\right)$:
  $X(t_{n+1}) - M\,X(t_n) = O(\tau^{p+1})$.
  On peut alors montrer que pour $0 \le n \le N$: $\max \left|X(t_n) - X[n]\right| = O(\tau^p)$
- Application aux différentes méthodes :

- Pour la méthode d'Euler explicite :
  $X(t_{n+1}) = X(t_n) + \tau\, A\, X(t_n) + O(\tau^2)$ d'après la formule de Taylor-Young
  $X[n+1] = M\,X[n]$ avec $M = (I_2 + \tau\, A)$, donc $X[n+1] = X[n] + \tau\, A\, X[n]$
  Donc $X(t_{n+1}) - (X(t_n) + \tau\, A\, X(t_n)) = O(\tau^2)$, donc la méthode est d'ordre 1.
- Pour la méthode d'Euler implicite :
  $X(t_{n+1}) = X(t_n) + \tau\, A\, X(t_n) + O(\tau^2)$ d'après la formule de Taylor-Young
  $X[n+1] = M\,X[n]$ avec $M = (I_2 - \tau\, A)^{-1} = (I_2 + \tau\, A + O(\tau^2))$, donc $X[n+1] = X[n] + \tau\, A\, X[n] + O(\tau^2)$
  Donc $X(t_{n+1}) - (X(t_n) + \tau\, A\, X(t_n)) = O(\tau^2)$, donc la méthode est d'ordre 1.
- Pour la méthode de Runge-Kutta d'ordre 2 :

$X(t_{n+1}) = X(t_n) + \tau A X(t_n) + \frac{\tau^2}{2} A^2 X(t_n) + O(\tau^3)$ d'après la formule de Taylor-Young

D'après la question suivante, $X[n+1] = M X[n]$ avec $M = \left(I_2 + \tau A + \frac{\tau^2}{2} A^2 \right)$

Donc $X[n+1] = X[n] + \tau A X[n] + \frac{\tau^2}{2} A^2 X[n]$

Donc $X(t_{n+1}) - \left(X(t_n) + \tau A X(t_n) + \frac{\tau^2}{2} A^2 X(t_n) \right) = O(\tau^3)$, donc la méthode est d'ordre 2 (vu son nom, on aurait pu s'en douter !).

b) On met en œuvre la méthode de Runge-Kutta d'ordre deux : $X[n+1] \approx X[n] + \frac{\tau}{2}(K_1 + K_2)$

Donc $X[n+1] = X[n] + \frac{\tau}{2}(A X[n] + A(X[n] + \tau A X[n]))$

Donc $X[n+1] = X[n] + \tau A X[n] + \frac{\tau^2}{2} A^2 X[n] = \left(I_2 + \tau A + \frac{\tau^2}{2} A^2 \right) X[n]$

Donc $\boxed{X[n+1] = \left(I_2 + \tau A + \frac{\tau^2}{2} A^2 \right) X[n]}$

Donc $X[n+1] = \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \tau \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} + \frac{\tau^2}{2} \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \right) X[n]$

Or $\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$

Donc $\boxed{X[n+1] = \begin{pmatrix} 1 - \frac{\tau^2}{2} & \tau \\ -\tau & 1 - \frac{\tau^2}{2} \end{pmatrix} X[n]}$

Il suffit de changer la ligne 13 : on la remplace par :

```
M = np.array([[1-(tau**2)/2, tau], [-tau, 1-(tau**2)/2]])
```

ou bien par :

```
M = np.identity(2) + tau * A + (tau**2) / 2 * A.dot(A)
```

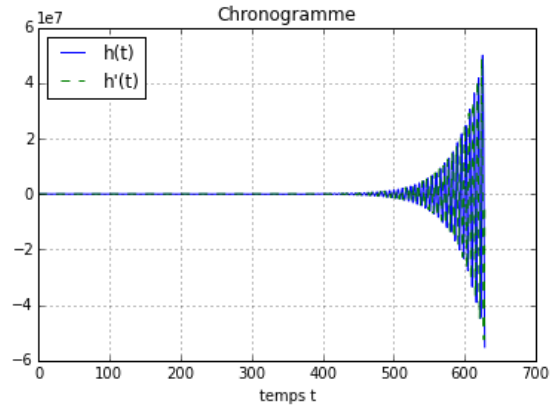
Remarque : np.*identity(2) est identique à np.eye(2), c'est la matrice identité de taille 2 × 2.*

c) Pour le chronogramme, les oscillations semblent bien sinusoïdales, et pour le diagramme de phase, il semble qu'on observe bien une ellipse.

Le nombre de points étant identique au cas de la méthode d'Euler (2**10), la méthode de Runge-Kutta d'ordre deux est donc plus performante. Toutefois, cette méthode n'est pas stable (cf graphe ci-dessous pour une durée d'affichage plus grande).

Remarque hors-programme : pour la méthode de Runge-Kutta d'ordre 2, les valeurs propres complexes de $M = \begin{pmatrix} 1 - \frac{\tau^2}{2} & \tau \\ -\tau & 1 - \frac{\tau^2}{2} \end{pmatrix}$ sont de module $\sqrt{1 + \frac{\tau^4}{4}} > 1$, donc le schéma est instable.

Runge-Kutta d'ordre 2 :



Question 11 : Méthode « stable » :

a) Matrice de couplage : $A=\begin{pmatrix}0 & 1\\ -1 & -\tau\end{pmatrix}$ (notation A maladroite mais c'est celle du script… A n'a pas le même sens que précédemment, c'est-à-dire que $X'\neq A\,X$ à présent…).

Matrice d'évolution : $M=I_2+\tau A=\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}+\tau\begin{pmatrix}0 & 1\\ -1 & -\tau\end{pmatrix}=\begin{pmatrix}1 & \tau\\ -\tau & 1-\tau^2\end{pmatrix}$

On obtient donc : $\boxed{X[n+1]=\begin{pmatrix}1 & \tau\\ -\tau & 1-\tau^2\end{pmatrix}X[n]}$

b)

- Pour la méthode proposée, la notation A pour la matrice $\begin{pmatrix}0 & 1\\ -1 & -\tau\end{pmatrix}$ est maladroite. En effet, A n'a pas le même sens que précédemment, c'est-à-dire que $X'\neq A\,X$… Notons donc dorénavant B cette matrice : $B=\begin{pmatrix}0 & 1\\ -1 & -\tau\end{pmatrix}$.

  On a $B=\begin{pmatrix}0 & 1\\ -1 & -\tau\end{pmatrix}=\begin{pmatrix}0 & 1\\ -1 & 0\end{pmatrix}+\begin{pmatrix}0 & 0\\ 0 & -\tau\end{pmatrix}=A+\begin{pmatrix}0 & 0\\ 0 & -\tau\end{pmatrix}$ (avec A la matrice telle que $X'=A\,X$).

  $M=I_2+\tau B=I_2+\tau A+\tau\begin{pmatrix}0 & 0\\ 0 & -\tau\end{pmatrix}=I_2+\tau A+O(\tau^2)$

  Donc $X[n+1]=X[n]+\tau A\,X[n]+O(\tau^2)$

  Or d'après la formule de Taylor-Young : $X(t_{n+1})=X(t_n)+\tau A\,X(t_n)+O(\tau^2)$

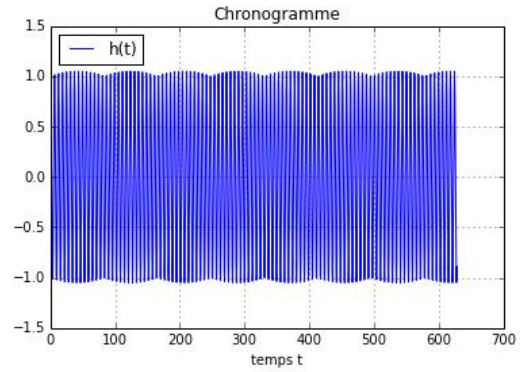
  Donc $X(t_{n+1})-(X(t_n)+\tau A\,X(t_n))=O(\tau^2)$, donc la méthode est <u>d'ordre 1</u>.

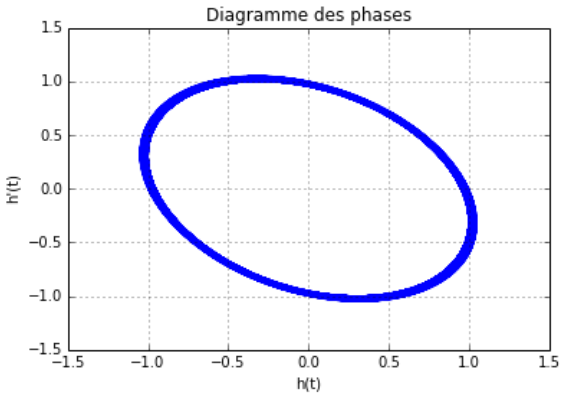
- La méthode semble <u>stable</u> car pour un temps grand, l'amplitude de la sinusoïde ne semble pas avoir augmenté.

  <u>Remarque hors-programme</u> : Pour la méthode proposée, les valeurs propres complexes de $M=\begin{pmatrix}1 & \tau\\ -\tau & 1-\tau^2\end{pmatrix}$ sont $\left(1-\frac{\tau^2}{2}\pm i\,\tau\sqrt{1-\frac{\tau^2}{4}}\right)$. Elles sont de module 1, donc la méthode est stable (la solution discrète reste bornée).

<u>Remarque</u> : avec le script proposé, on obtient (avec npoints = 2**10) :

Méthode stable :





ce qui n'est pas tout à fait conforme ni à ce qui est attendu (cf diagramme des phases, l'ellipse n'est pas d'axes x et y…), ni à la figure 5 (cf chronogramme : présence d'oscillations sur l'enveloppe…).
Par contre, avec davantage de points (npoints = 2**15 par exemple), on obtient bien le chronogramme proposé figure 5, et un diagramme des phases cohérent.

4. Discrétisation spatiale du problème :

Question 12 : Equation adimensionnée :

Dans la partie modélisation, à la question 6, on a établi : $\lambda = \frac{\omega^2}{c^2}$ avec ω la pulsation de l'onde et c sa célérité.

Par ailleurs, $\frac{\partial u}{\partial x} = \frac{\partial U}{\partial X}\frac{dX}{dx} = \frac{\partial U}{\partial X}\frac{N}{L} = \frac{1}{d}\frac{\partial U}{\partial X}$.

En dérivant une seconde fois : $\frac{\partial^2 u}{\partial x^2} = \frac{1}{d^2}\frac{\partial^2 U}{\partial X^2}$.

De même : $\frac{\partial^2 u}{\partial y^2} = \frac{1}{d^2}\frac{\partial^2 U}{\partial Y^2}$.

Donc : $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{1}{d^2}\left(\frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2}\right) = \frac{1}{d^2}\Delta U$

Or $\Delta u = -\lambda u = -\lambda U = -\frac{\omega^2}{c^2} U$

On obtient : $\Delta U = -\frac{\omega^2 d^2}{c^2} U = -\lambda' U$, d'où : $\boxed{\lambda' = \frac{\omega^2 d^2}{c^2}}$

Pour $x = d = \frac{L}{N}$, $X = \frac{x}{L} N = 1$, donc le pas de discrétisation dans ce jeu de variables vaut une unité.

Question 13 : Discrétisation autour d'un point central :

a) On a : $U(X_{i-1}, Y_j) \approx U(X_i, Y_j) + (X_{i-1} - X_i)\frac{\partial U}{\partial X}(X_i, Y_j) + \frac{(X_{i-1} - X_i)^2}{2}\frac{\partial^2 U}{\partial X^2}(X_i, Y_j)$

Or : $(X_{i-1} - X_i) = ((i-1) - i) = -1$

Donc : $\boxed{U(X_{i-1}, Y_j) \approx U(X_i, Y_j) - \frac{\partial U}{\partial X}(X_i, Y_j) + \frac{1}{2}\frac{\partial^2 U}{\partial X^2}(X_i, Y_j)}$ (a)

b) De même : $U(X_{i+1}, Y_j) \approx U(X_i, Y_j) + (X_{i+1} - X_i)\frac{\partial U}{\partial X}(X_i, Y_j) + \frac{(X_{i+1} - X_i)^2}{2}\frac{\partial^2 U}{\partial X^2}(X_i, Y_j)$

Or : $(X_{i+1} - X_i) = ((i+1) - i) = 1$

Donc : $U(X_{i+1}, Y_j) \approx U(X_i, Y_j) + \frac{\partial U}{\partial X}(X_i, Y_j) + \frac{1}{2}\frac{\partial^2 U}{\partial X^2}(X_i, Y_j)$ (b)

En faisant (a) + (b) - 2 $U_{i,j}$, on obtient : $\frac{\partial^2 U}{\partial X^2} \approx U_{i-1,j} + U_{i+1,j} - 2\,U_{i,j}$ (c)

De même : $U(X_i, Y_{j-1}) \approx U(X_i, Y_j) + (Y_{j-1} - Y_j)\frac{\partial U}{\partial Y}(X_i, Y_j) + \frac{(Y_{j-1} - Y_j)^2}{2}\frac{\partial^2 U}{\partial Y^2}(X_i, Y_j)$

Or : $(Y_{j-1} - Y_j) = ((j-1) - j) = -1$

Donc : $U(X_i, Y_{j-1}) \approx U(X_i, Y_j) - \frac{\partial U}{\partial Y}(X_i, Y_j) + \frac{1}{2}\frac{\partial^2 U}{\partial Y^2}(X_i, Y_j)$ (d)

De même : $U(X_i, Y_{j+1}) \approx U(X_i, Y_j) + (Y_{j+1} - Y_j)\frac{\partial U}{\partial Y}(X_i, Y_j) + \frac{(Y_{j+1} - Y_j)^2}{2}\frac{\partial^2 U}{\partial Y^2}(X_i, Y_j)$

Or : $(Y_{j+1} - Y_j) = ((j+1) - j) = 1$

Donc : $U(X_i, Y_{j+1}) \approx U(X_i, Y_j) + \frac{\partial U}{\partial Y}(X_i, Y_j) + \frac{1}{2}\frac{\partial^2 U}{\partial Y^2}(X_i, Y_j)$ (e)

En faisant (d) + (e) - 2 $U_{i,j}$, on obtient : $\frac{\partial^2 U}{\partial Y^2} \approx U_{i,j-1} + U_{i,j+1} - 2\,U_{i,j}$ (f)

En faisant (c) + (f), on obtient : $\Delta U = \frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2} \approx U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4\,U_{i,j}$

Or on a vu précédemment que $\Delta U = -\lambda' U$, donc $\Delta U(X_i, Y_j) = -\lambda' U_{i,j}$

On obtient : $\boxed{\Delta U = \frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2} \approx U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4\,U_{i,j} = -\lambda' U_{i,j}}$ (g)

Question 14 : Discrétisation autour des bords :

a) Pour i = 0, $U_{i-1,j} = 0$, donc $\boxed{U_{1,j} + U_{0,j-1} + U_{0,j+1} - 4\,U_{0,j} = -\lambda' U_{0,j}}$ (h)

b) Pour j = 0, $U_{i,j-1} = 0$, donc $\boxed{U_{i-1,0} + U_{i+1,0} + U_{i,1} - 4\,U_{i,0} = -\lambda' U_{i,0}}$ (i)

c) Pour i = j = 0, $U_{i-1,j} = 0$ et $U_{i,j-1} = 0$, donc $\boxed{U_{1,0} + U_{0,1} - 4\,U_{0,0} = -\lambda' U_{0,0}}$ (j)

d) Si les bords sont fixes, $U_{0,j} = U_{i,0} = 0 \ \forall i, \forall j$

D'après (h) : $U_{1,j} = 0 \ \forall j$

D'après (i) : $U_{i,1} = 0 \ \forall i$

Par itération, $U_{i,j} = 0 \ \forall i, \forall j$ (en utilisant la relation (g))

Alors la plaque entière serait fixe avec ce modèle !

Donc on considèrera que les bords sont libres.

Question 15 : Détermination de l'équation matricielle équivalente :

a) Pour tout i ($0 \le i \le N$), F_i est une matrice colonne à $(N + 1)$ lignes. Ainsi, F est une matrice colonne à $(N + 1) \times (N + 1) = (N + 1)^2$ lignes. La matrice M se retrouve être une matrice carrée à $(N + 1)^2$ lignes et $(N + 1)^2$ colonnes : elle possède donc au plus $(N + 1)^2$ valeurs propres réelles comptées avec multiplicité.
Remarque : On montre à la question 15-c) que M est symétrique réelle donc diagonalisable. Par conséquent, M possède exactement $(N + 1)^2$ valeurs propres réelles comptées avec multiplicité.
Il y a donc $(N + 1)^2$ modes propres possibles.

b) Pour N = 4, la matrice M est déjà énorme (25×25)… Il n'est pas raisonnable d'écrire une telle matrice… La réponse est la même qu'à la question suivante…

c) En utilisant les résultats des questions 13) et 14) :

On a :
$$\begin{cases} -U_{i-1,0} + 4U_{i,0} - U_{i,1} - U_{i+1,0} = \lambda' U_{i,0} \\ -U_{i-1,1} - U_{i,0} + 4U_{i,1} - U_{i,2} - U_{i+1,1} = \lambda' U_{i,1} \\ \vdots \\ -U_{i-1,N-1} - U_{i,N-2} + 4U_{i,N-1} - U_{i,N} - U_{i+1,N-1} = \lambda' U_{i,N-1} \\ -U_{i-1,N} - U_{i,N-1} + 4U_{i,N} - U_{i+1,N} = \lambda' U_{i,N} \end{cases}$$

Il vient :
$$\left(\begin{array}{ccccc|ccccc|ccccc} -1 & 0 & \cdots & \cdots & 0 & 4 & -1 & 0 & \cdots & 0 & -1 & 0 & \cdots & \cdots & 0 \\ 0 & -1 & \ddots & \ddots & \vdots & -1 & 4 & \ddots & \ddots & \vdots & 0 & -1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots & 0 & \ddots & \ddots & \ddots & 0 & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 & 0 & \vdots & \ddots & \ddots & 4 & -1 & \vdots & \ddots & \ddots & -1 & 0 \\ \underbrace{0}_{-I_N} & \cdots & \cdots & 0 & -1 & \underbrace{0}_{A} & \cdots & 0 & -1 & 4 & \underbrace{0}_{-I_N} & \cdots & \cdots & 0 & -1 \end{array}\right) \begin{pmatrix} F_{i-1} \\ F_i \\ F_{i+1} \end{pmatrix} = \lambda' \begin{pmatrix} F_{i-1} \\ F_i \\ F_{i+1} \end{pmatrix}$$

Donc :
$$\left(\begin{array}{ccccc} A & -I_N & 0_N & \cdots & 0_N \\ -I_N & A & \ddots & \ddots & \vdots \\ 0_N & \ddots & \ddots & \ddots & 0_N \\ \vdots & \ddots & \ddots & A & -I_N \\ \underbrace{0_N}_{M} & \cdots & 0_N & -I_N & A \end{array}\right) \begin{pmatrix} F_0 \\ F_1 \\ \vdots \\ F_{N-1} \\ F_N \end{pmatrix} = \lambda' \begin{pmatrix} F_0 \\ F_1 \\ \vdots \\ F_{N-1} \\ F_N \end{pmatrix}$$

Donc : $MF = \lambda' F$ avec $\boxed{A = \begin{pmatrix} 4 & -1 & 0 & \cdots & 0 \\ -1 & 4 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 4 & -1 \\ 0 & \cdots & 0 & -1 & 4 \end{pmatrix}}$ (matrice carrée ayant (N+1) lignes et (N+1) colonnes).

d)

```python
def genereM(A):
    """Fonction dont l'objectif est de renvoyer M à partir de A"""
    n, p = A.shape  # dimensions du tableau A   # n = N+1
    # On crée un tableau de bonne dimension, rempli de 0
    M = np.zeros(shape=(n**2, n**2))
    I = np.eye(n)   # matrice identité de bonne dimension
    for i in range(n):  # pour placer A dans M
        M[n*i:n*(i+1), n*i:n*(i+1)] = A
    for i in range(n-1):    # pour placer - I dans M
        M[n*(i+1):n*(i+2), n*i:n*(i+1)] = - I
        M[n*i:n*(i+1), n*(i+1):n*(i+2)] = - I
    return M    # renvoie la matrice M
```

5. Méthode de la puissance itérée :

Question 16 :

a)

```python
def normeM(M):
    """Fonction dont l'objectif est de renvoyer la norme de M à partir de M"""
    n, p = M.shape
    r = abs(M[0, 0])   # initialisation de la valeur glissante
    for i in range(n):
        for j in range(p):
            if abs(M[i, j]) > r:
                r = abs(M[i, j])
    return r    # norme de M
```

b)

```python
def normevecteur(F):
    Fn = F / normeM(F)
    return Fn   # vecteur normé
```

Question 17 :

a) Le vocabulaire probabiliste utilisé dans l'énoncé est inapproprié avec « choisit aléatoirement un vecteur F_0 » de R^n. Selon quelle loi ? Uniforme sous-entendu mais il faut préciser des bornes.
Dans le code qui suit, F0 est un vecteur de dimension adéquate (tableau 2D à n lignes, 1 colonne), dont les coefficients sont choisis aléatoirement entre 0 et 1, alors que l'énoncé demande simplement que les coefficients soient réels, sans condition supplémentaire. Il est plus sensé de fixer des valeurs minimale et maximale. Le fait de choisir des réels de [0,1[est naturel.

```python
def puissanceiter(M, p):
    n, m = M.shape
    F0 = rd.rand(n, 1)  # tableau 2D de dimension adéquate choisi aléatoirement
    Fp = F0     # initialisation
    for k in range(p):
        Fp = normevecteur(M.dot(Fp))
    return Fp   # pième vecteur
```

b) Si on veut que la fonction « iterstabilise » choisisse aléatoirement <u>un</u> vecteur F_0 (et pas un nouveau vecteur F_0 à chaque itération, alors on ne peut pas utiliser la fonction « puissanceiter » étudiée à la question précédente…
De plus, utiliser la fonction « puissanceiter » dans la fonction « iterstabilise » serait beaucoup moins intéressant du point de vue de la complexité, car il faudrait recalculer à chaque itération les termes F_0, F_1, F_2, …, F_{p-1}.
Du coup, on ne voit pas l'intérêt d'avoir créé la fonction « puissanceiter »…

```
def iterstabilise(M, er):
    n, m = M.shape
    F0 = rd.rand(n, 1)  # tableau 2D de dimension adéquate choisi aléatoirement
    F = normevecteur(M.dot(F0))
    while normeM(F - F0) >= er:
        F0 = F
        F = normevecteur(M.dot(F0))
    return F    # vecteur stabilisé
```

c) Un invariant de boucle permet de démontrer qu'une boucle produit l'effet attendu. On utilise un invariant de boucle, c'est-à-dire une propriété qui :
- est vérifiée avant d'entrer dans la boucle,
- si elle est vérifiée avant une itération, est vérifiée après celle-ci,
- lorsqu'elle est vérifiée en sortie de boucle permet d'en déduire que le programme est correct.

Ici : « (F0, F) contient (F_p, F_{p+1}) avant la $p^{ème}$ itération de la boucle conditionnelle » est un invariant de boucle.

Un variant de boucle permet de démontrer qu'une boucle se termine effectivement. On identifie un variant, autrement dit une expression (c'est souvent le simple contenu d'une variable) :
- qui est un entier positif tout au long de la boucle,
- et qui diminue strictement après chaque itération.

On peut alors en conclure que la boucle se termine.

Ici : Je n'en vois pas et je ne pense pas qu'il y en ait ! Quelle mauvaise situation pour tester cette notion ! Le fait que la boucle se termine est dû au fait que la suite converge (admis par l'énoncé). On peut d'ailleurs s'interroger sur la terminaison réelle dans le cas où e_r est très petit du fait des approximations liées à la représentation des flottants…

Question 18 : Vecteur et valeur propre principaux

a) Méthode 1 :

On sait que la suite $(F_p)_{p\geq 0}$ converge vers F (vecteur propre principal associé à la valeur propre λ'). Donc si « er » est faible, le résultat de l'appel de la fonction « iterstabilise(M, er) » nous retourne un vecteur F_{approx} proche de F.

On aura ainsi : $M.F_{approx} \approx \lambda' F_{approx}$.

En comparant la première composante, ou la deuxième, ..., ou la dernière composante de $M.F_{approx}$ à celle de F_{approx}, on dispose d'une valeur approchée de λ'. Afin de limiter les erreurs, on choisit la composante de F_{approx} de valeur absolue maximale.

Code Python :

```
# On suppose que A a été généré par d'autres personnes.
M = genereM(A)      # on commence par générer M
F = iterstabilise(M, er=0.00000001)  # on identifie F et Fapprox
# On recherche l'indice i de la valeur absolue maximale de Fapprox
i = 0
m = abs(F[0])
for j in range(len(F)):
    if abs(F[j]) > m:
        i = j
        m = abs(F[j])
# On retourne une valeur approchée de la valeur propre
G = M.dot(F)
print("lambdaprime = ", float(G[i]/F[i]))
```

Méthode 2 :
On sait que la suite $(F_p)_{p\geq 0}$ converge vers F (vecteur propre principal associé à la valeur propre λ'). Donc si « er » est faible, le résultat de l'appel de la fonction « iterstabilise(M, er) » nous retourne un vecteur proche de F.

D'après la relation de récurrence $F_{p+1} = \frac{M.F_p}{\|M.F_p\|}$, on obtient $M.F = \|M.F\| F$.

Par ailleurs, λ' est la valeur propre associée à F, donc $M.F = \lambda' F$.
On obtient $\lambda' = \|M.F\| = \text{normeM}(M.\text{dot}(F))$.

Remarque : On remarque au passage que la valeur propre de M de module maximal est positive ce que l'énoncé ne précise pas (les valeurs propres sont juste supposées non nulles). En réalité, on pourrait montrer que $Sp(M) \subset [0,8]$.

Code Python :

```
# On suppose que A a été généré par d'autres personnes.
M = genereM(A)      # on commence par générer M
F = iterstabilise(M, er=0.00000001)  # renvoie le vecteur propre principal de M
lambdaprime = normeM(M.dot(F))      # on calcule lambdaprime
print("lambdaprime = ", lambdaprime)    # on affiche
```

b) A la question 12, on a établi : $\lambda' = \frac{\omega^2 d^2}{c^2}$ avec $\omega = 2 \pi \nu$ (ω étant la pulsation, et ν la fréquence).

On obtient : $\boxed{\nu = \frac{c\sqrt{\lambda'}}{2\pi d}}$

6. Représentation graphique :

Question 19 :

F est un vecteur-colonne contenant N+1 vecteurs F_i.
F_i est un vecteur-colonne contenant N+1 coefficients U_{ij}.
F est donc un vecteur-colonne contenant $(N+1)^2$ coefficients U_{ij}.
En posant nb = N+1 :

```
def transforme(F):
    Z = F.copy()
    nb = np.sqrt(F.size)
    Z = Z.flatten()     # on transforme en un tableau 1D (on met "tout à plat")
    Z = Z.reshape(nb, nb)     # reforme Z en tableau 2D nb*nb
    return Z    # matrice des valeurs U
```

Question 20 :

- Les bords semblent libres, mais les altitudes y semblent faibles (en accord avec les conditions aux limites).
- Les lignes de niveau z = 0 correspondent aux figures de Chladni (l'altitude des points situés sur ces courbes ne changera jamais au cours du temps étant donnée la forme z = u(x,y) h(t) supposée des solutions).
  Les deux premiers modes ne semblent pas correspondre à une des figures données sur la figure 1 du sujet (cette liste n'est pas exhaustive donc ce n'est pas si inquiétant). Par contre, le mode

correspondant à la dernière figure proposée semble nous donner la figure 72a, ce qui accrédite la pertinence du modèle retenu.

7. Distribution des valeurs propres :

Question 21 :

a) Si le tableau n'était pas ordonné, il faudrait (dans le pire des cas, c'est-à-dire si la valeur recherchée est absente, ou bien si cette valeur se situe en dernière position) comparer la valeur recherchée à toutes les valeurs propres contenues dans le tableau. Il faudrait donc parcourir tout le tableau. La complexité asymptotique serait donc linéaire (O(n)).

b) Je propose deux solutions :

```
def recherchebrute(valpropre, v, er):
    r = False
    n = valpropre.size - 1      # valpropre contient n+1 éléments
    i = n       # i est le variant de boucle
    while i >= 0 and abs(valpropre[i] - v) > er:
        i = i - 1
    if i >= 0:
        r = True
    return r    # booléen


def recherchebrute_bis(valpropre, v, er):
    r = False
    n = valpropre.size - 1      # valpropre contient n+1 éléments
    for i in range(n+1):
        if abs(valpropre[i] - v) < er:
            r = True
    return r    # booléen
```

Pour la 1ère solution (boucle while), dès que l'on a trouvé un élément dans le tableau « valpropre » suffisamment proche de la valeur « v », on arrête la recherche.
Pour la 2ème solution (boucle for), on parcourt tout le tableau « valpropre », même si on a déjà trouvé un élément suffisamment proche de la valeur v…
La 1ère solution est préférable en termes de complexité dans le meilleur des cas.

c) La complexité asymptotique d'une recherche dichotomique est logarithmique (O(log n)).

d) D'après Wikipédia (!) : La recherche dichotomique est un algorithme de recherche pour trouver la position d'un élément dans un tableau trié. Le principe est le suivant : comparer l'élément avec la valeur de la case au milieu du tableau ; si les valeurs sont égales, la tâche est accomplie, sinon on recommence dans la moitié du tableau pertinente.

e)

```
def recherchedicho(valpropre, v):
    r = False   # booléen
    a = 0   # a et b sont les bornes de recherche, ici on initialise
    b = valpropre.size - 1      # valpropre contient b+1 éléments
    m = (a+b)//2    # milieu du tableau
    while a <= b and r is False:
        if valpropre[m] == v:   # c'est gagné !
            r = True
        elif valpropre[m] < v:
            a = m + 1   # on cherche dans la moitié à droite
        else:
            b = m - 1   # on cherche dans la moitié à gauche
        m = (a+b)//2
    return r    # booléen
```

Question 22 :

a) Que faut-il comprendre quand l'énoncé dit : « utilisant la dichotomie de la question précédente » ? Faut-il utiliser la fonction « recherchedicho » de la question précédente, fonction qui retourne un booléen ? Ou bien faut-il utiliser la méthode générale de la dichotomie pour rechercher les indices minimum et maximum à conserver (fonction « recherche_ind » dans l'algorithme ci-dessous) ? Pas clair…
Bref, ci-dessous une solution, mais qui comporte 2 « return » (avec cependant un « return » se trouvant dans la fonction imbriquée « recherche_ind », et un seul return dans la fonction principale…) :

```
def rechercheplagedicho(valpropre, valmin, valmax):
    def recherche_ind(v):
        r = False
        a = 0
        b = valpropre.size - 1
        m = (a+b)//2
        while a <= b and r is False:
            if valpropre[m] == v:
                r = True    # dans ce cas, m est l'indice recherché
            elif valpropre[m] < v:
                a = m + 1
            else:
                b = m - 1
            m = (a+b)//2
        return m    # dans les autres cas, b = m et a = m + 1
        # on retourne m (indice inférieur : valpropre[m] < v < valpropre[m+1])
    ind_min = recherche_ind(valmin) + 1
    ind_max = recherche_ind(valmax)
    if valpropre[ind_max] == valmax:    # valeur à exclure (intervalle ouvert)
        valselect = valpropre[ind_min: ind_max]
    else:
        valselect = valpropre[ind_min: ind_max + 1]
    return valselect    # tableau 1D
```

b) L'algorithme présenté est à base de « technique de masquage ».
Pour les valeurs propres valpropre[i] de la plaque comprises dans le bon intervalle :
- (valpropre[i] > valmin) = True
- (valpropre[i] < valmax) = True
- Donc mask[i] = (valpropre[i] > valmin) * (valpropre[i] < valmax) = True*True = True

Pour les valeurs propres valpropre[i] de la plaque inférieures à valmin :
- (valpropre[i] > valmin) = False
- (valpropre[i] < valmax) = True
- Donc mask[i] = (valpropre[i] > valmin) * (valpropre[i] < valmax) = False*True = False

Pour les valeurs propres valpropre[i] de la plaque supérieures à valmax :
- (valpropre[i] > valmin) = True
- (valpropre[i] < valmax) = False
- Donc mask[i] = (valpropre[i] > valmin) * (valpropre[i] < valmax) = True*False = False

On obtient alors des tableaux 1D qui pourraient ressembler aux tableaux suivants :

```
valpropre > valmin =  [False False False  True  True  True  True  True]
valpropre < valmax =  [ True  True  True  True  True  True False False]
mask =  [False False False  True  True  True False False]
```

L'instruction valpropre[mask] permet alors de ne conserver que les valeurs où le masque vaut True.

c) C'est un algorithme en temps linéaire garanti dans le cas le pire comme dans le cas le meilleur : valpropre > valmin retourne un tableau de booléens t tel que t[i] est vrai si et seulement si valpropre[i] > valmin. Pour le construire, il faut évidemment parcourir tout le tableau valpropre. Cet algorithme parcourt d'abord le tableau deux fois (calcul de valpropre > valmin et valpropre < valmax), une troisième fois pour calculer la conjonction terme à terme des résultats (c'est le sens de l'opération « * ») et enfin une quatrième pour calculer valpropre[mask].
La complexité asymptotique de ce code est donc linéaire (O(n)).

d) Je ne suis pas convaincu par l'énoncé de cette question… L'auteur affirme que l'utilisation des masques est une centaine de fois plus rapide que la dichotomie… A partir de quoi ? Je ne sais pas…
Par la méthode dichotomique, on trouve les indices minimum et maximum en complexité logarithmique, ce qui est plus performant qu'une complexité linéaire. Il reste ensuite à créer le tableau valselect (complexité linéaire dans le pire des cas).
Or on a vu dans la question précédente qu'avec l'algorithme utilisant des masques, on parcourt 4 fois le tableau en complexité linéaire.
Donc :

- Soit l'auteur a interverti ses constatations…
- Soit il a mal programmé sa dichotomie…
- Soit il y a une autre explication qui m'échappe…

Peut-être que l'auteur a essayé de programmer une dichotomie et s'est raté. Il l'a peut-être fait en partageant le tableau en deux moitiés par slicing (« technique » dont il est « coutumier ») mais :

- Soit il n'a pas bien compris que le slicing sur les tableaux Python se faisait en temps linéaire ce qui rendait la complexité de son algorithme de dichotomie linéaire et non logarithmique ;
- Soit il sait que le slicing se fait en temps constant sur les tableaux numpy mais a par erreur appliqué son algorithme à des tableaux Python...

8. Confrontation du modèle aux expériences :

Question 23 : Modèle initial :

a) On a établi à la question 6) c) : $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -\frac{\omega^2}{c^2} u$.

Avec $u = C \sin\left(m \pi \frac{x}{L}\right) \sin\left(n \pi \frac{y}{L}\right)$, il vient : $-m^2 \frac{\pi^2}{L^2} u - n^2 \frac{\pi^2}{L^2} u = -\frac{\omega^2}{c^2} u$, d'où $\frac{\omega^2}{c^2} = \left(m^2 + n^2\right)\frac{\pi^2}{L^2}$

Or $\omega = 2 \pi \nu$, donc $\boxed{\nu = \frac{c}{2L}\sqrt{m^2 + n^2}}$

b) Pour m = 1 et n = 1 : $\nu_1 = \frac{c}{2L}\sqrt{1^2 + 1^2}$ donc $\boxed{\nu_1 = \frac{c}{L\sqrt{2}}}$

Or c = 5,1.10^3 m.s^{-1} et L = 25,0 cm, donc $\nu_1 = \frac{5{,}1.10^3}{25{,}0.10^{-2}\sqrt{2}}$. On obtient $\boxed{\nu_1 = 1{,}4.10^4 \text{ Hz}}$.

Or l'expérience dit que ν_1 est de l'ordre de la dizaine de Hz ! Ce n'est donc pas compatible !

c) On a :

- pour m = 1, n = 2 ou bien m = 2, n = 1 : $\nu = \frac{c}{2L}\sqrt{1^2+2^2} = \frac{c}{2L}\sqrt{5} = \frac{c}{L\sqrt{2}}\sqrt{\frac{5}{2}} = \nu_1\sqrt{\frac{5}{2}}$

  On a donc : $\frac{\nu}{\nu_1} = \sqrt{2,5} \approx 1,6 \neq 2,5$!!! (on lit 2,5 dans le tableau pour les fréquences d'index 2 et 3).

- pour m = 2, n = 2 : $\nu = \frac{c}{2L}\sqrt{2^2+2^2} = \frac{c}{2L}\sqrt{8} = \frac{c}{L\sqrt{2}}2 = \nu_1 2$

  On a donc : $\frac{\nu}{\nu_1} = 2 \neq 4$!!! (on lit 4 dans le tableau pour la fréquence d'index 4).

Donc la distribution des premières fréquences de résonance n'est pas conforme à celle prévue par l'équation de dispersion.

d) Au vu des résultats de b) et c), on en conclut que le modèle utilisé n'est pas pertinent.

Question 24 : Un improbable modèle :

a) L'équation d'onde proposée correspond à une équation de diffusion (dérivée seconde par rapport aux coordonnées d'espace, dérivée première par rapport au temps). Ce modèle est inadéquat car on sait que les phénomènes diffusifs sont « lents » et efficaces uniquement à courte échelle spatiale, ce qui ne semble pas être le cas ici.

b) On a $[\Delta f] = [f].L^{-2} = L.L^{-2} = L^{-1}$ et $\left[\alpha_1 \frac{\partial f}{\partial t}\right] = [\alpha_1].[f].T^{-1} = [\alpha_1].L.T^{-1}$.

On en déduit : $\boxed{[\alpha_1] = T.L^{-2}}$ (α_1 est en $s.m^{-2}$).

Question 25 : Un autre modèle :

a) On a $[\Delta(\Delta f)] = [f].L^{-4} = L.L^{-4} = L^{-3}$ et $\left[\alpha_2 \frac{\partial^2 f}{\partial t^2}\right] = [\alpha_2].[f].T^{-2} = [\alpha_2].L.T^{-2}$.

On en déduit : $\boxed{[\alpha_2] = T^2.L^{-4}}$ (α_2 est en $s^2.m^{-4}$).

b) On a vu dans la question 3) : $[E] = M.L^{-1}.T^{-2}$ et $[\rho] = M.L^{-3}$

Donc $\left[\frac{\rho}{E\,e^2}\right] = \frac{M.L^{-3}}{M.L^{-1}.T^{-2}.L^2} = T^2.L^{-4} = [\alpha_2]$, or $\alpha_2 < 0$ (voir question suivante : $\omega^2 > 0$)

On propose donc : $\boxed{\alpha_2 = -\frac{\rho}{E\,e^2}}$

Remarque : $\alpha_2 = -\frac{\rho}{E\,L^2}$ ou $\alpha_2 = -\frac{\rho}{E\,e\,L}$ sont aussi possibles du point de vue dimensionnelle…

c) On a $f = u(x,y)\times h(t) = C\,h_0 \sin\left(m\pi\frac{x}{L}\right)\sin\left(n\pi\frac{y}{L}\right)\exp(i\omega t)$

Donc $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = -m^2\frac{\pi^2}{L^2}f - n^2\frac{\pi^2}{L^2}f = -(m^2+n^2)\frac{\pi^2}{L^2}f$

Donc $\Delta(\Delta f) = (m^2 + n^2)^2 \frac{\pi^4}{L^4} f$

Par ailleurs, $\alpha_2 \frac{\partial^2 f}{\partial t^2} = -\alpha_2 \omega^2 f$

On en déduit : $\omega^2 = \frac{\pi^4 (m^2 + n^2)^2}{-\alpha_2 L^4}$ (d'où $\alpha_2 < 0$ car $\omega^2 > 0$), or $\omega = 2\pi\nu$, donc

$$\boxed{\nu = \frac{\pi (m^2 + n^2)}{2\sqrt{-\alpha_2}\, L^2} = \frac{\pi \sqrt{E}\, e (m^2 + n^2)}{2\sqrt{\rho}\, L^2}}$$

Remarque : pour m = 1 et n = 1 :

$\nu_1 = \frac{\pi \sqrt{E}\, e (1^2 + 1^2)}{2\sqrt{\rho}\, L^2} = \frac{\pi \sqrt{E}\, e}{\sqrt{\rho}\, L^2} = 2{,}5.10^2$ Hz / dizaine de Hz : bof !!

d) On a :

- pour m = 1, n = 2 ou bien m = 2, n = 1 : $\nu = \frac{\pi \sqrt{E}\, e (1^2 + 2^2)}{2\sqrt{\rho}\, L^2} = \frac{5}{2} \frac{\pi \sqrt{E}\, e}{\sqrt{\rho}\, L^2} = \frac{5}{2} \nu_1$

  On a donc : $\frac{\nu}{\nu_1} = 2{,}5$ (OK, on lit 2,5 dans le tableau pour les fréquences d'index 2 et 3).

- pour m = 2 et n = 2 : $\nu = \frac{\pi \sqrt{E}\, e (2^2 + 2^2)}{2\sqrt{\rho}\, L^2} = \frac{\nu_1}{2} (2^2 + 2^2) = 4\nu_1$

  On a donc : $\frac{\nu}{\nu_1} = 4$ (OK, on lit 4 dans le tableau pour la fréquence d'index 4).

- pour m = 1, n = 3 ou bien m = 3, n = 1 : $\frac{\nu}{\nu_1} = \frac{(1^2 + 3^2)}{2} = 5$ (OK index 5 et 6).

- pour m = 2, n = 3 ou bien m = 3, n = 2 : $\frac{\nu}{\nu_1} = \frac{(2^2 + 3^2)}{2} = \frac{13}{2} = 6{,}5$ (OK index 7 et 8).

- pour m = 1, n = 4 ou bien m = 4, n = 1 : $\frac{\nu}{\nu_1} = \frac{(1^2 + 4^2)}{2} = \frac{17}{2} = 8{,}5$ (OK index 9 et 10).

- pour m = 3 et n = 3 : $\frac{\nu}{\nu_1} = \frac{(3^2 + 3^2)}{2} = 9$ (OK index 11).

- pour m = 2, n = 4 ou bien m = 4, n = 2 : $\frac{\nu}{\nu_1} = \frac{(2^2 + 4^2)}{2} = 10$ (OK index 12 et 13).

On voit que la distribution des fréquences obtenue expérimentalement est conforme à celle prévue par cette nouvelle équation de dispersion.

e) On a $f(x, y, t) = u(x, y) \times h(t)$

Si l'équation (3) reste adéquate : $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -\lambda u$

Alors $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \times h(t) = -\lambda u(x, y) \times h(t)$

Alors $\Delta(\Delta f) = -\lambda \Delta u \times h(t) = -\lambda (-\lambda u) \times h(t) = \lambda^2 u(x, y) \times h(t)$

Or $\Delta(\Delta f) = \alpha_2 \frac{\partial^2 f}{\partial t^2} = \alpha_2 u(x, y) \times h''(t)$

Donc $\lambda^2\, u(x,y)\times h(t)=\alpha_2\, u(x,y)\times h''(t) \quad \forall x, \forall y$

Donc $\lambda^2\, h(t)=\alpha_2\, h''(t)$, donc $h''(t)-\frac{\lambda^2}{\alpha_2}h(t)=0$ qui devient ($\alpha_2 < 0$) : $h''(t)+\omega^2\, h(t)=0$, ce qui est cohérent avec $\lambda=\omega\sqrt{-\alpha_2}$ (solution harmonique).

Donc même dans le cadre de ce modèle, la recherche des solutions numériques de l'équation (3) reste adéquate.

Question 26 : Dégénérescence des modes :

a) Voir la question 25) c) et d) : les modes mn et nm ont même fréquence : $\nu=\frac{\nu_1}{2}\left(m^2+n^2\right)$.

b) On veut $z(x,y,t)=0 \ \forall t$, donc $\sin\left(\pi\frac{x}{L}\right)\sin\left(2\pi\frac{y}{L}\right)+\sin\left(2\pi\frac{x}{L}\right)\sin\left(\pi\frac{y}{L}\right)=0$

Or $\sin(2\alpha)=2\sin(\alpha)\cos(\alpha)$, on obtient donc : $\sin\left(\pi\frac{x}{L}\right)\sin\left(\pi\frac{y}{L}\right)\left(\cos\left(\pi\frac{y}{L}\right)+\cos\left(\pi\frac{x}{L}\right)\right)=0$

Les lignes nodales ont donc pour équation :

- $\boxed{x = 0}$ (c'est un bord de la plaque : OK)
- $\boxed{x = L}$ (c'est un bord de la plaque : OK)
- $\boxed{y = 0}$ (c'est un bord de la plaque : OK)
- $\boxed{y = L}$ (c'est un bord de la plaque : OK)
- $\left(\cos\left(\pi\frac{y}{L}\right)+\cos\left(\pi\frac{x}{L}\right)\right)=0$, donc $\cos\left(\pi\frac{y}{L}\right)=-\cos\left(\pi\frac{x}{L}\right)=\cos\left(\pi-\pi\frac{x}{L}\right)$

  Or $0<\pi\frac{y}{L}<\pi$ (car $0<y<L$) et $0<\pi\frac{x}{L}<\pi$ (car $0<x<L$), donc $0<\pi-\pi\frac{x}{L}<\pi$

  On en déduit $\pi\frac{y}{L}=\pi-\pi\frac{x}{L}$, donc $\frac{y}{L}+\frac{x}{L}=1$, et donc $\boxed{y=L-x}$

  La plaque carrée de côté L est représentée ci-dessous, ainsi que la ligne nodale (l'axe x est horizontal, l'axe y vertical) :