

Etude des figures de Chladni sur une plaque rectangulaire à bords libres

VIGNESWARAN Subhen

2024–2025

N°SCEI 10723

Physique (*Physique ondulatoire*)

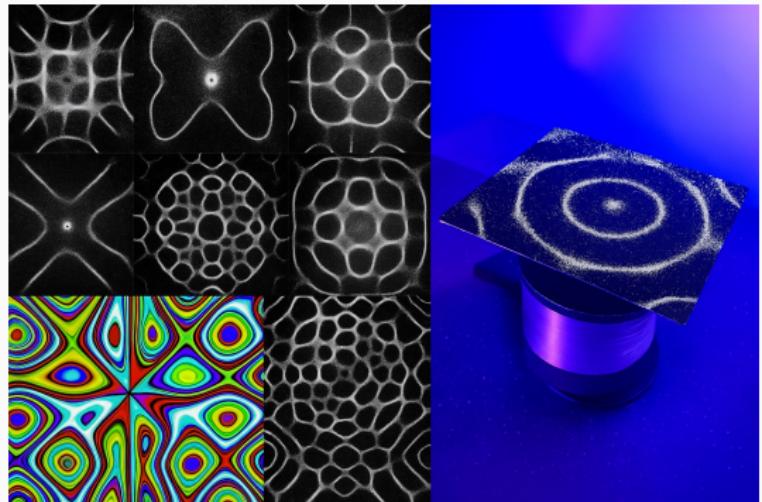
Informatique (*Informatique pratique*)

Mathématiques (*Algèbre*)

Introduction



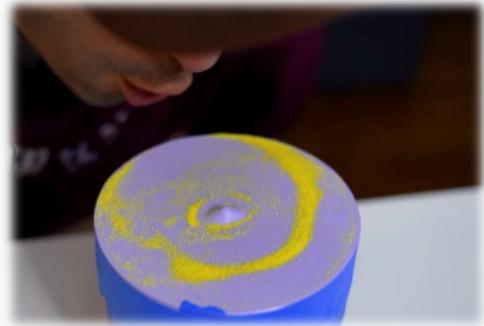
Ernst Chladni
(1756-1827)



Ernst Chladni — Figures en noir et blanc — Figure générée(IA) et colorée — Figure-fond violet

Problématique

Comment peut on prévoir les figures de Chladni qui se dessinent sur une plaque rectangulaire à bords libres ?



Plan

Explication du phénomène

Prévision de la forme des figures de Chladni

Recherche des fréquences de résonance associées

Expériences et discussions

Annexes

Explication du phénomène

Explication du phénomène

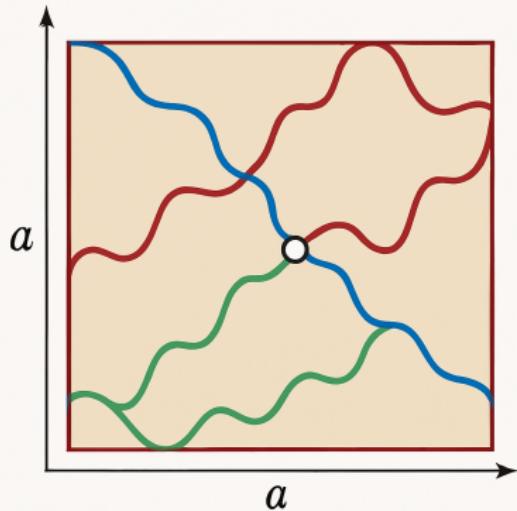


Figure 1: Ondes stationnaires

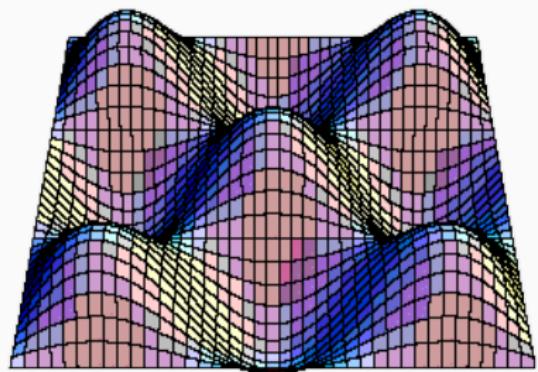


Figure 2: Ventres et noeuds

Prévision de la forme des figures de Chladni

1^{ère} modélisation: Oscillations libres

Equation de d'Alembert:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{1}{c^2} \frac{\partial^2 u}{\partial t^2}$$

où $c = \sqrt{\frac{E}{\rho}}$ avec E module de Young de la plaque et ρ sa masse volumique

Equation de d'Alembert adimensionnée:

$$\frac{\partial^2 \hat{u}}{\partial \hat{x}^2} + \frac{\partial^2 \hat{u}}{\partial \hat{y}^2} = \frac{\partial^2 \hat{u}}{\partial \hat{t}^2} \quad (1)$$

avec $(\hat{x}, \hat{y}) = (\frac{x}{a}, \frac{y}{a})$; $\hat{t} = \frac{tc}{a}$; $\hat{u} = \frac{u}{H}$ où H est l'amplitude de u

Conditions initiales:

- $\frac{\partial u}{\partial x}(x, y, t) = 0$ en $x = \pm 1$
- $\frac{\partial u}{\partial y}(x, y, t) = 0$ en $y = \pm 1$
- $u(x, y, 0) = U_0(x, y)$
- $\frac{\partial u}{\partial t}(x, y, 0) = 0$

Forme de l'onde:

$$u(x, y, t) = A \cos(k_x x) \cos(k_y y) \cos(\omega t)$$

Après résolution: $k_x = m\pi$ et $k_y = n\pi$ avec $(m, n) \in \mathbb{N}^2$;
 $\omega = \sqrt{k_x^2 + k_y^2}$ donc $\omega_{m,n} = \pi\sqrt{m^2 + n^2}$.

Forme générale de u :

$$u(x, y, t) = \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} A_{m,n} \cos(m\pi x) \cos(n\pi y) \cos(\omega_{m,n} t)$$

Expression de $A_{m,n}$:

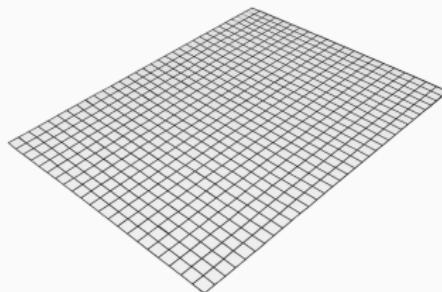
$$A_{m,n} = \int_{-1}^1 \int_{-1}^1 U_o(x, y) \cos(m\pi x) \cos(n\pi y) dx dy$$

Discrétisation 1^{er} modèle

Méthode différences finies centrées:

$$\frac{u_{i,j}^{k+1} - 2u_{i,j}^k + u_{i,j}^{k-1}}{\Delta t^2} = \frac{u_{i+1,j}^k - 2u_{i,j}^k + u_{i-1,j}^k}{h^2} + \frac{u_{i,j+1}^k - 2u_{i,j}^k + u_{i,j-1}^k}{h^2}$$
$$\Rightarrow u_{i,j}^{k+1} = 2u_{i,j}^k - u_{i,j}^{k-1} + \Delta t^2 \frac{u_{i+1,j}^k - 2u_{i,j}^k + u_{i-1,j}^k}{h^2} + \Delta t^2 \frac{u_{i,j+1}^k - 2u_{i,j}^k + u_{i,j-1}^k}{h^2}$$

où Δt représente le pas temporel et h le pas spatial

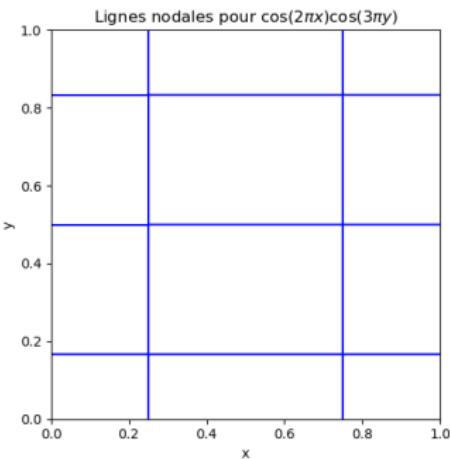


Un exemple: $w = \pi\sqrt{13}$

Modes possibles: $A_{2,3}\cos(2\pi x)\cos(3\pi y)$ (avec $(m, n) = (2, 3)$) et
 $A_{3,2}\cos(3\pi x)\cos(2\pi y)$ (avec $(m, n) = (3, 2)$)

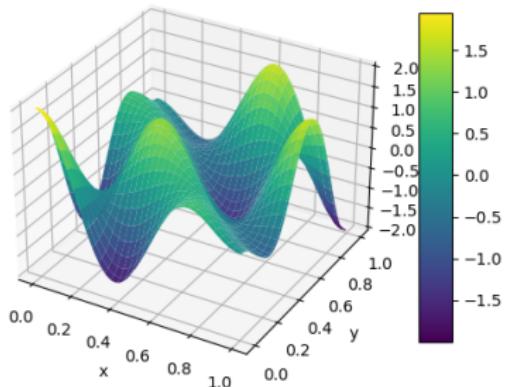
Symétrie des conditions initiales (principe de Curie):

$$A_{2,3} = A_{3,2}$$



Version analytique

Surface 3D: $u(x, y)$ pour $\omega = \pi\sqrt{13}$



Lignes nodales ($u=0$)

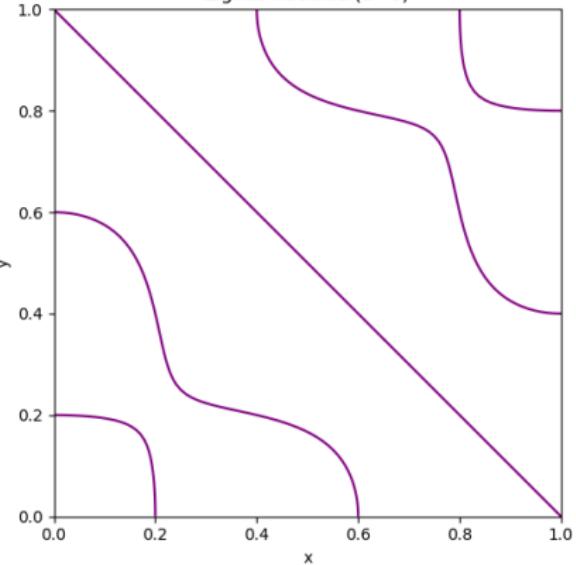


Figure 1: représentation 3D de l'onde et lignes nodales

Version équation différences finies

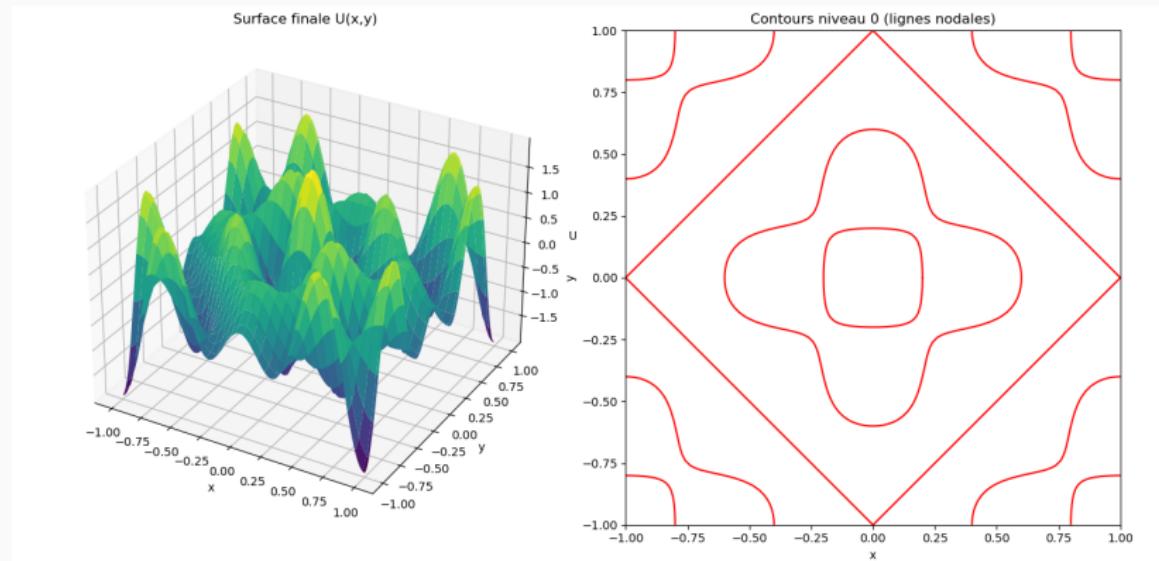


Figure 2: Représentation version numérique (différence liée à l'erreur numérique

2^{ème} modèle: Régime forcée

$$\frac{\partial^2 u}{\partial t^2}(x, y, t) = \frac{\partial^2 u}{\partial x^2}(x, y, t) + \frac{\partial^2 u}{\partial y^2}(x, y, t) + F(x, y, t)$$

où $F(x, y, t) = F_0(x, y) \cos(wt)$

Conditions initiales :

$$u(x, y, 0) = 0,$$

$$\frac{\partial u}{\partial t}(x, y, 0) = 0$$

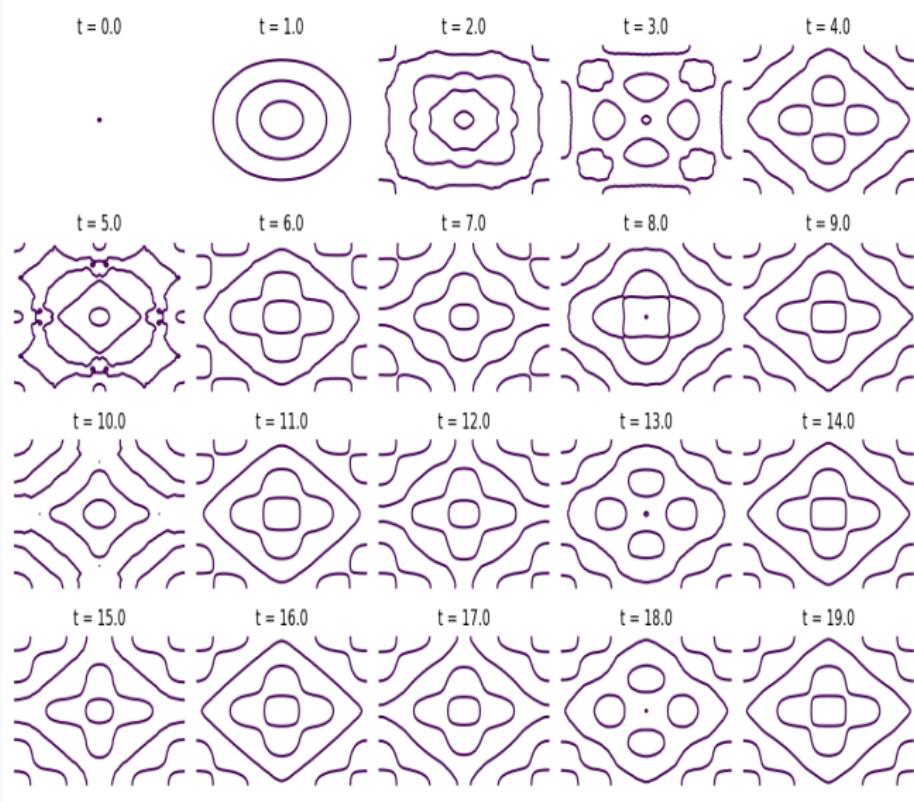
$$F_0(x, y) = \begin{cases} \alpha & \text{si } x = y = 0 \\ 0 & \text{sinon} \end{cases}$$

Discrétisation 2^{ème} modèle

Même méthode que précédemment:

$$\begin{aligned} \frac{u_{i,j}^{k+1} - 2u_{i,j}^k + u_{i,j}^{k-1}}{\Delta t^2} &= \frac{u_{i+1,j}^k - 2u_{i,j}^k + u_{i-1,j}^k}{h^2} + \frac{u_{i,j+1}^k - 2u_{i,j}^k + u_{i,j-1}^k}{h^2} + F(x_i, y_j, t_k) \\ \Rightarrow u_{i,j}^{k+1} &= 2u_{i,j}^k - u_{i,j}^{k-1} + \Delta t^2 \frac{u_{i+1,j}^k - 2u_{i,j}^k + u_{i-1,j}^k}{h^2} + \\ &\quad \Delta t^2 \frac{u_{i,j+1}^k - 2u_{i,j}^k + u_{i,j-1}^k}{h^2} + \Delta t^2 F(x_i, y_j, t_k) \end{aligned}$$

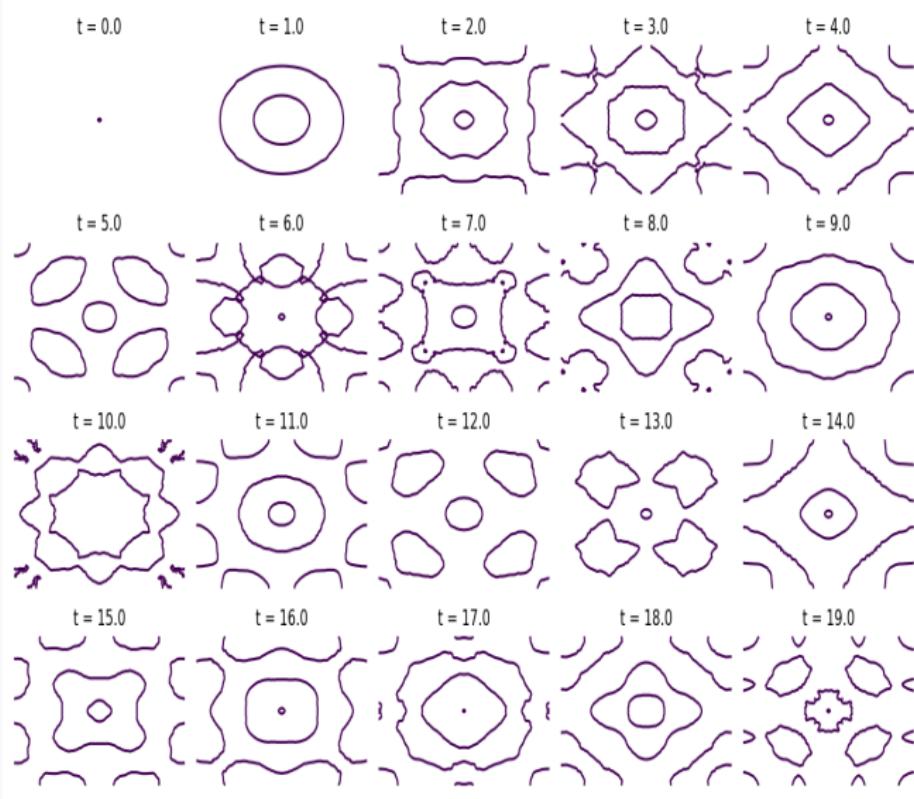
équation différences finies- $w = \pi\sqrt{13}$



Comparaison

Régime libre	Régime forcé
<ul style="list-style-type: none">- Forte dépendance aux conditions initiales : nécessite de connaître les modes propres et leurs combinaisons- Résultat immédiat dès la première itération (après perturbation initiale)- Complexité équivalente (en simulation)	<ul style="list-style-type: none">- Ne nécessite que de connaître la fréquence de résonance- Nécessite un certain temps pour atteindre un régime stable ; dépend des propriétés de la plaque- Complexité équivalente

fréquence non resonante: $w = \pi\sqrt{6.5}$



Recherche des fréquences de résonance associées

Recherche des fréquences de résonance par méthode matricielle

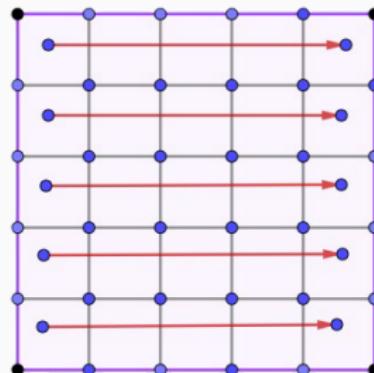
$$u(x, y, t) = f(t)\phi(x, y) \Rightarrow \left(\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2}\right)f = \frac{\partial^2 f}{\partial t^2}\phi \text{ (avec (1))}.$$

Donc

$$D(\phi)f = -\omega^2 f \phi \Rightarrow D\phi = \lambda \phi$$

où D est le laplacien en cartésien, $\lambda = -\omega^2$

$$\phi = (\phi_{00} \ \dots \ \phi_{ij} \ \phi_{i,j+1} \ \dots \ \phi_{i+1,j} \ \phi_{i+1,j+1} \ \dots \ \phi_{nn})^\top$$

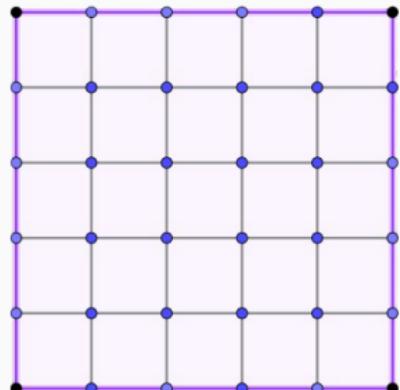


$$D\phi \hat{=} \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{h^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{h^2}$$

$$\text{Donc } \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - 4\phi_{i,j}}{h^2} = \lambda\phi_{i,j}$$

Conditions de Neumann:

- $\phi_{n+1,j} = \phi_{n-1,j}$
- $\phi_{-1,j} = \phi_{1,j}$
- $\phi_{i,n+1} = \phi_{i,n-1}$
- $\phi_{i,-1} = \phi_{i,1}$



$$D = \frac{1}{h^2} \begin{pmatrix} B & J & 0 & 0 & \cdots & 0 & 0 \\ I & B & I & 0 & \cdots & 0 & 0 \\ 0 & I & B & I & \ddots & \vdots & \vdots \\ 0 & 0 & I & B & \ddots & 0 & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & I & 0 \\ 0 & 0 & \cdots & 0 & I & B & I \\ 0 & 0 & \cdots & \cdots & 0 & J & B \end{pmatrix},$$

où chaque bloc est de taille $(n+1) \times (n+1)$

avec : $B \in \mathbb{R}^{(n+1) \times (n+1)}$

$$J = 2 \cdot I$$

$$I = I_{n+1}$$

$$B = \begin{pmatrix} -4 & 2 & 0 & 0 & \cdots & 0 \\ 1 & -4 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -4 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & -4 & 1 \\ 0 & \cdots & 0 & 0 & 2 & -4 \end{pmatrix}$$

Expériences et discussions

Expériences



Montage expérimental



Montage vu de haut

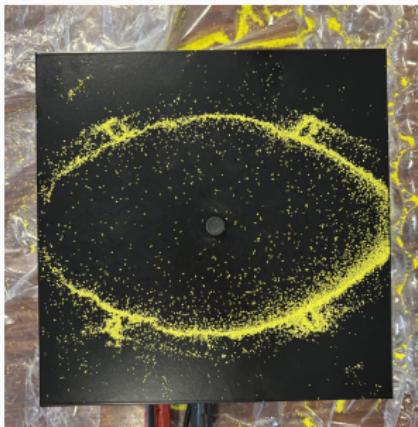


Figure 1: 149 Hz

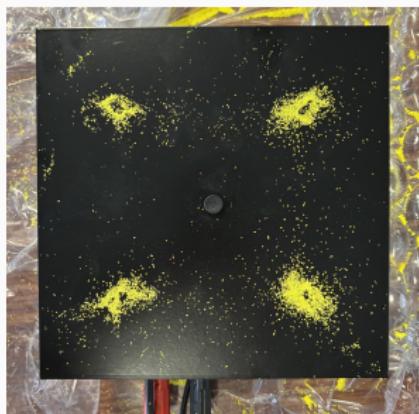


Figure 2: 154 Hz

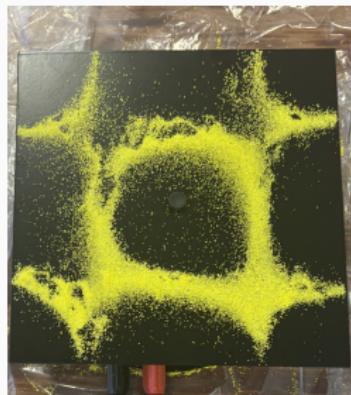


Figure 3: 440 Hz

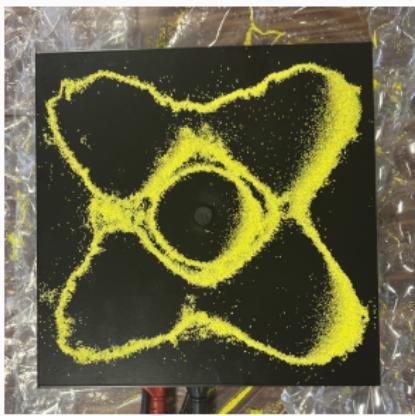


Figure 4: 751.769 Hz



Figure 5: 1047 Hz

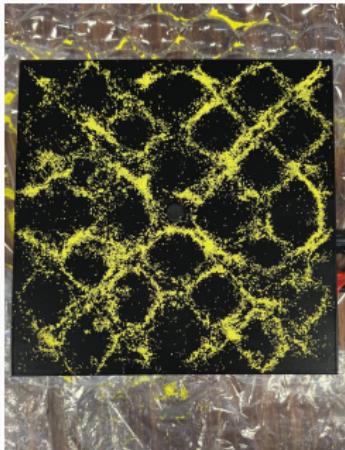
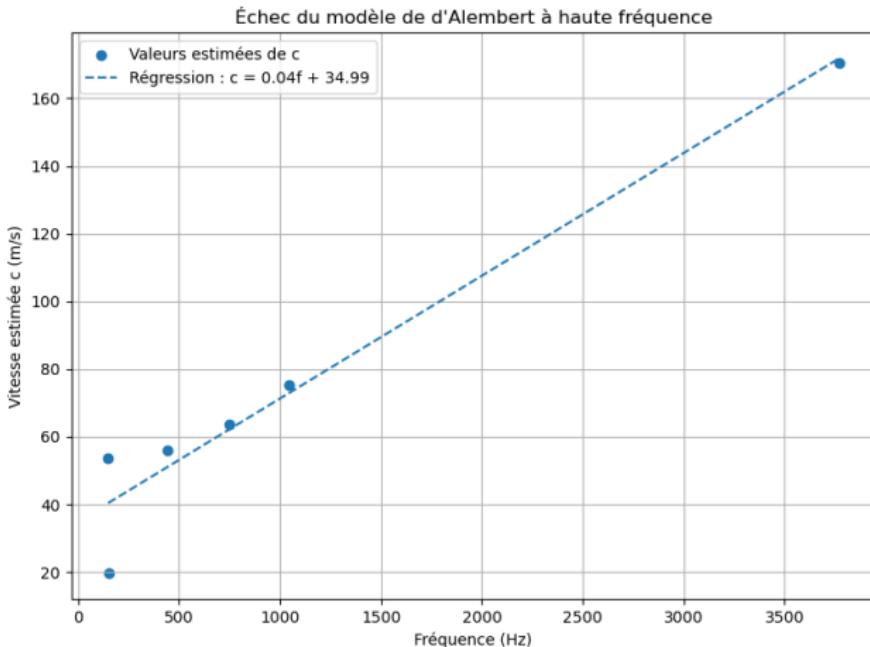


Figure 6: 3768 Hz

Discussions: 1^{ère} limite du modèle - c non constant

Avec $\hat{t} = \frac{ct}{a}$, $\omega = \frac{c\hat{\omega}}{a}$. Donc $\omega = \frac{c\pi\sqrt{m^2+n^2}}{a}$

$\Rightarrow f = \frac{c\sqrt{m^2+n^2}}{2a} \Rightarrow c = \frac{2fa}{\sqrt{m^2+n^2}}$ où a vaut 18 cm.

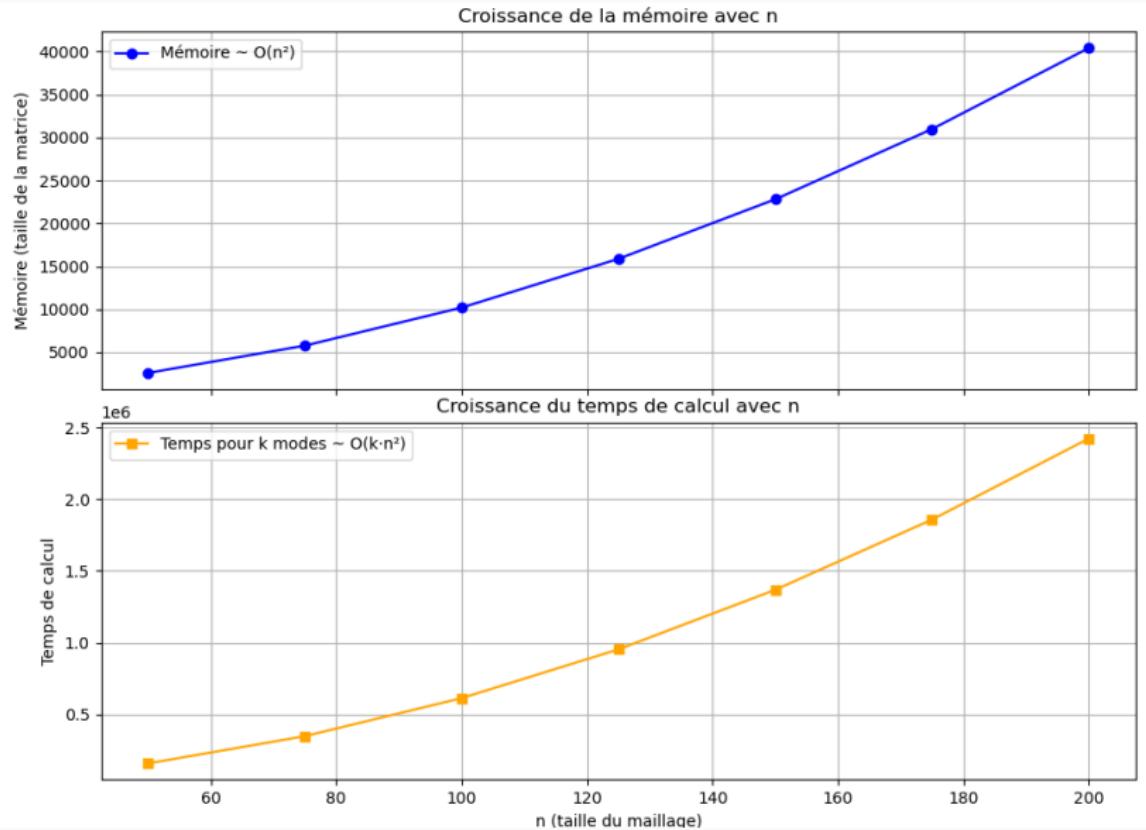


2^{ème} limite: valeurs propres (pour $c = 54.5 \text{m.s}^{-1}$ et $n = 100$)

f_{cible} (Hz)	λ_{cible}	$\lambda_{\text{trouvée}}$	f_{phys} (Hz)
0.0	-0.000	-0.000000	0.00
111.1	-5.317	-5.316327	698.12
222.2	-21.266	-8.000261	856.40
333.3	-47.849	-7.999972	856.38
444.4	-85.064	-8.000078	856.39
555.6	-132.913	-8.000206	856.40
666.7	-191.394	-8.000081	856.39
777.8	-260.509	-7.999898	856.38
888.9	-340.257	-8.000199	856.40
1000.0	-430.637	-8.000316	856.40

Formule: $\lambda = -\left(\frac{a}{c}\right)^2 (2\pi f_{\text{phys}})^2$

3^{ème} limite: complexité ($k=60$)



Modèle de Kirchoff-Love (plus approprié)

$$\rho h \frac{\partial^2 u}{\partial t^2} + D \nabla^4 u = 0$$

où :

- ρ : masse volumique,
- h : épaisseur de la plaque,
- $D = \frac{Eh^3}{12(1 - \nu^2)}$: rigidité en flexion,
- ∇^4 : laplacien d'ordre 4 (bi-laplacien), défini comme :

$$\nabla^4 = \frac{\partial^4}{\partial x^4} + 2 \frac{\partial^4}{\partial x^2 \partial y^2} + \frac{\partial^4}{\partial y^4}$$

$$f_{mn} = \alpha_{mn} \cdot \frac{h}{a^2} \cdot \sqrt{\frac{E}{12\rho(1 - \nu^2)}}$$

Merci de votre attention !

Annexes

Relation de dispersion (démonstration)

- Avec $u(x, y, t) = A \cos(k_x x) \cos(k_y y) \cos(\omega t)$, $\frac{\partial u}{\partial x}(x, y, t) = 0$ en $x = \pm 1$

$\Rightarrow \pm A k_x \sin(k_x) \cos(k_y y) \cos(\omega t) = 0$. Donc $\sin(k_x) = 0$ donc
 $k_x = m\pi$ avec $m \in \mathbb{N}$.

- De même, $k_y = n\pi$ avec $n \in \mathbb{N}$.

- En injectant la forme de u dans l'équation de d'Alembert adimensionnée (1), on a:

$$-\omega^2 u = -(k_x^2 + k_y^2)u \Rightarrow \omega = \sqrt{k_x^2 + k_y^2} = \pi \sqrt{m^2 + n^2}$$

Contours version analytique

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Grille spatiale
5 n = 200
6 x = np.linspace(0, 1, n+1)
7 y = np.linspace(0, 1, n+1)
8 X, Y = np.meshgrid(x, y)
9 m1, n1 = 2, 3
10 m2, n2 = 3, 2
11
12 # Modes propres
13 U_1 = np.cos(m1 * np.pi * X) * np.cos(m2 * np.pi * Y)
14 U_2 = np.cos(m2 * np.pi * X) * np.cos(n2 * np.pi * Y)
15
16 # Tracé des lignes nodales
17 fig, axs = plt.subplots(1, 2, figsize=(12, 5))
18
19 # Mode (2,3)
20 axs[0].contour(X, Y, U_1, levels=[0], colors='blue')
21 axs[0].set_title(r"**Lignes nodales pour  $\cos(2\pi x) \cos(3\pi y)$ **")
22 axs[0].set_xlabel("x")
23 axs[0].set_ylabel("y")
24 axs[0].set_aspect('equal')
25
26 # Mode (3,2)
27 axs[1].contour(X, Y, U_2, levels=[0], colors='red')
28 axs[1].set_title(r"**Lignes nodales pour  $\cos(3\pi x) \cos(2\pi y)$ **")
29 axs[1].set_xlabel("x")
30 axs[1].set_ylabel("y")
31 axs[1].set_aspect('equal')
32
33 plt.tight_layout()
34 plt.show()
35
36 # Superposition
37
38 # Modes propres
39 m1, n1 = 2, 3
40 m2, n2 = 3, 2
41
42 # Combinaison linéaire avec A = B = 1
43 U = np.cos(m1 * np.pi * X) * np.cos(n1 * np.pi * Y) + \
44     np.cos(m2 * np.pi * X) * np.cos(n2 * np.pi * Y)
45
46
47 # --- Tracé 3D ---
48 fig1 = plt.figure(figsize=(10,6))
49 ax = fig1.add_subplot(121, projection='3d')
50 surf = ax.plot_surface(X, Y, U, cmap='viridis')
51 ax.set_title(r"**Surface 3D:  $u(x,y)$  pour  $\omega = |\pi| \sqrt{13}$ **")
52 ax.set_xlabel("x")
53 ax.set_ylabel("y")
54 ax.set_zlabel("Amplitude")
55 fig1.colorbar(surf, ax=ax, shrink=0.5, aspect=10)
56
57 # --- Tracé des lignes nodales ---
58 ax2 = fig1.add_subplot(122)
59 contours = ax2.contour(X, Y, U, levels=[0], colors='purple')
60 ax2.set_title("**Lignes nodales ( $u=0$ )**")
61 ax2.set_xlabel("x")
62 ax2.set_ylabel("y")
63 ax2.set_aspect('equal')
64
65 plt.tight_layout()
66 plt.show()
67
```

Contours version différences finies

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 p = 100 # nombre d'intervalles
5 x = np.linspace(-1, 1, p + 1)
6 y = np.linspace(-1, 1, p + 1)
7 h = 2 / p
8 m = 3
9 n = 2
10 dt = h / 2
11 T = 10 # temps total de simulation
12
13 # Initialisation des conditions initiales
14 X, Y = np.meshgrid(x, y, indexing='ij')
15 U0 = np.cos(m * np.pi * X) * np.cos(n * np.pi * Y) + \
16     np.cos(n * np.pi * X) * np.cos(m * np.pi * Y)
17 Uoo = U0.copy()
18
19 U = np.zeros_like(U0)
20 nb_past = int(T / dt)
21
22 for t in range(nb_past):
23     for i in range(p + 1):
24         for j in range(p + 1):
25             i1 = i - 1 if i > 0 else 1
26             i1 = i + 1 if i < p else p - 1
27             j1 = j - 1 if j > 0 else 1
28             j1 = j + 1 if j < p else p - 1
29
30             U[i, j] = ((U0[i_1, j] - 4 * U0[i, j] + U0[i1, j] + \
31                         U0[i, j_1] + U0[i, j1]) \
32                         / h**2 + dt**2 - Uoo[i, j] + 2 * U0[i, j])
33
34 Uoo = U0.copy()
35 U = U.copy()
36
37 # Affichage de la surface 3D finale
38 fig = plt.figure(figsize=(18, 6))
39
40 ax1 = fig.add_subplot(121, projection='3d')
41 X_plot, Y_plot = np.meshgrid(x, y)
42 ax1.plot_surface(X_plot, Y_plot, U, cmap='viridis')
43 ax1.set_title("Surface finale U(x,y)")
44 ax1.set_xlabel("x")
45 ax1.set_ylabel("y")
46 ax1.set_zlabel("U")
47
48 # Affichage des lignes nodales (contour niveau 0)
49 ax2 = fig.add_subplot(122)
50 ax2.contour(x, y, U.T, levels=[0], colors='red')
51 ax2.set_title("Contours niveau 0 (lignes nodales)")
52 ax2.set_xlabel("x")
53 ax2.set_ylabel("y")
54 ax2.set_aspect('equal')
55
56 plt.tight_layout()
57 plt.show()
```

Snapshots différences forcées

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 n = 100
5 x = np.linspace(-1, 1, n + 1)
6 y = np.linspace(-1, 1, n + 1)
7 alpha = 5
8 h = 2 / n
9 dt = h / 2
10 T = 20
11 w = np.pi * np.sqrt(3**2+2**2)
12
13 def force(alpha, w, t):
14     return alpha * np.cos(w * t)
15
16 Uo = np.zeros((n + 1, n + 1))
17 Uoo = np.zeros((n + 1, n + 1))
18 U = np.zeros((n + 1, n + 1))
19
20 nb_pas = int(T / dt)
21 nb_capture = 20
22 intervalle_temps = nb_pas // nb_capture
23
24 images = []
25 temps = []
26
27 t = 0
28 while t < nb_pas:
29     for i in range(n + 1):
30         for j in range(n + 1):
31             if not (i == (n + 1) // 2 and j == (n + 1) // 2):
32                 i_1 = i - 1 if i > 0 else 1
33                 i_1 = i + 1 if i < n else n - 1
34                 j_1 = j - 1 if j > 0 else 1
35                 j_1 = j + 1 if j < n else n - 1
```

```
36
37     U[i, j] = ((Uo[i_1, j] + 4 * Uo[i, j] + Uo[i_1, j]) + \
38                  Uo[i, j_1] + Uo[i, j_1])) \
39                  / h**2 * dt**2 - Uoo[i, j] + 2 * Uo[i, j])
40
41 pc = (n + 1) // 2
42 U[pc, pc] = (((Uo[pc - 1, pc] + 4 * Uo[pc, pc] + Uo[pc + 1, pc] + \
43                  Uo[pc, pc - 1] + Uo[pc, pc + 1])) \
44                  / h**2 + force(alpha, w, dt * t)) * dt**2 - \
45                  Uoo[pc, pc] + 2 * Uo[pc, pc])
46
47 Uoo = Uo.copy()
48 Uo = U.copy()
49
50 if t % intervalle_temps == 0:
51     images.append(U.copy())
52     temps.append(t * dt)
53
54 t += 1
55
56 # Affichage final 4x5 des contours
57 fig, axs = plt.subplots(4, 5, figsize=(12, 6))
58 for k, ax in enumerate(axs.flat):
59     cs = ax.contour(images[k], levels=[0])
60     ax.set_title(f't = {temps[k]:.1f}')
61     ax.axis('off')
62
63 plt.tight_layout()
64 plt.show()
```

Régression : limite du modèle de d'Alembert

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Données expérimentales : fréquences et vitesses estimées
5 fréquences = np.array([149, 154, 440, 751.769, 1047, 3768]) # en Hz
6 c_estimée = np.array([53.64, 19.601, 56.003, 63.79, 75.384, 170.496])
7 # en m/s
8
9 # Régression linéaire avec numpy.polyfit
10 coeffs = np.polyfit(fréquences, c_estimée, deg=1) # Ajustement linéaire
11 fit_line = np.polyval(coeffs, fréquences)
12 # Valeurs ajustées pour la droite
13
14
15 # Tracé du graphique
16 plt.figure(figsize=(8, 6))
17 plt.scatter(fréquences, c_estimée, label="Valeurs estimées de c", \
18             marker='o')
19 plt.plot(fréquences, fit_line, linestyle='--', label=f"Régression : | \
20           c = {coeffs[0]:.2f}f + {coeffs[1]:.2f}")
21 plt.xlabel("Fréquence (Hz)")
22 plt.ylabel("Vitesse estimée c (m/s)")
23 plt.title("Échec du modèle de d'Alembert à haute fréquence")
24 plt.grid(True)
25 plt.legend()
26 plt.tight_layout()
27 plt.show()
```

Valeurs propres de D

```
1 import numpy as np
2 import scipy.sparse as sp
3 import scipy.sparse.linalg as spla
4
5 def build_custom_B(n):
6     B = np.zeros((n+1, n+1))
7     for i in range(n+1):
8         B[i, i] = -4
9         if i > 0:
10            B[i, i-1] = 1
11        if i < n:
12            B[i, i+1] = 1
13    B[0, 1] = 2
14    B[n, n-1] = 2
15    return sp.csr_matrix(B)
16
17 def build_custom_D(n, h=1.0):
18     B = build_custom_B(n)
19     I_block = sp.eye(n+1, format='csr')
20     J = 2 * I_block
21
22     blocks = []
23     for row in range(n+1):
24         row_blocks = []
25         for col in range(n+1):
26             if row == col:
27                 row_blocks.append(B)
28             elif abs(row - col) == 1:
29                 if row == 0 or row == n:
30                     row_blocks.append(J)
31                 else:
32                     row_blocks.append(I_block)
33             else:
34                 row_blocks.append(sp.csr_matrix((n+1, n+1)))
35     blocks.append(row_blocks)
36
37     D = sp.bmat(blocks, format='csr') / (h**2)
38     return D
39
40 # Constantes physiques
41 a = 0.18 # m
42 c = 54.5 # m/s
43
44 # Construction de D pour n = 100
45 n = 100
46 D = build_custom_D(n)
47
48 # Fréquences cibles uniformément espacées
49 f_target = np.linspace(0, 1000, 10)
50 omega_target = 2 * np.pi * f_target
51 lambda_target = - (a / c)**2 * omega_target**2
52
53 # Recherche des valeurs propres proches
54 true_lambdas = []
55 true_frequencies = []
56
57 for target in lambda_target:
58     eigval, _ = spla.eigs(D, k=1, sigma=target, which='LM')
59     true_lambdas.append(eigval[0])
60     true_frequencies.append((c / a) * np.sqrt(-eigval[0]))
61
62 # Résultat
63 for f, lam, lfound, wphys in zip(f_target, lambda_target, true_lambdas, \
64                                     true_frequencies):
65     print(f'{f:.1f} Hz | {lam:.3f} | {lfound:.6f} | {wphys:.2f} Hz")
66     lambda_trouvee = {lfound:.6f} | {wphys:.2f} Hz"
```

Courbes complexité

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Valeurs de n à tester
5 n = np.array([50, 75, 100, 125, 150, 175, 200])
6
7 # Taille totale du système : N = (n+1)^2
8 N = (n + 1) ** 2
9
10 # Coût mémoire ~ O(N)
11 cout_memoire = N
12
13 # Coût en temps pour k valeurs propres ~ O(k·N)
14 k = 60
15 cout_temps = k * N
16
17 # Tracer les deux courbes sur des sous-graphes séparés
18 fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 10), sharex=True)
19
20 # Graphique 1 : mémoire
21 ax1.plot(n, cout_memoire, label='Mémoire ~  $O(n^2)$ ', marker='o', color='blue')
22 ax1.set_ylabel("Mémoire (taille de la matrice)")
23 ax1.set_title("Croissance de la mémoire avec n")
24 ax1.grid(True)
25 ax1.legend()
26
27 # Graphique 2 : temps
28 ax2.plot(n, cout_temps, label='Temps pour k modes ~  $O(k \cdot n^2)$ ', marker='s', \
29           color='orange')
30 ax2.set_xlabel("n (taille du maillage)")
31 ax2.set_ylabel("Temps de calcul")
32 ax2.set_title("Croissance du temps de calcul avec n")
33 ax2.grid(True)
34 ax2.legend()
35
36 plt.tight_layout()
37 plt.show()
```