

INFORMATIQUE ET MÉTHODES NUMÉRIQUES
MASTER 1 DE PHYSIQUE
UNIVERSITÉ DE TOURS
2021–2022

Compte Rendu sur l'équation d'ondes avec conditions aux limites Robin

Stam Nicolis¹

Résumé

On étudie les propriétés de l'équations d'ondes, lorsque l'on impose des conditions aux limites Robin. On trouve que pour que la solution reste bornée, il faut inclure des termes de dispersion et que leurs coefficients soient à une relation très concrète avec les coefficients des conditions Robin.

On calcule le flux d'énergie à travers les bords, en fonction du paramètre, qui permet l'interpolation entre conditions aux limites Dirichlet et conditions aux limites Neumann, les deux cas limites des conditions aux limites Robin.

1. E-Mail : stam.nicolis@idpoisson.fr ; stam.nicolis@lmpt.univ-tours.fr

Table des matières

1	Introduction générale et plan de l'exposé	2
2	Partie théorique	3
2.1	Introduction	3
2.2	Le problème spectral pour $-d^2/du^2$ avec conditions aux limites Robin	4
2.3	La solution de l'équation d'ondes avec conditions aux limites Robin : Démonstration du Théorème II <u>Solondes</u>	6
2.4	Les effets de la dispersion	8
2.5	L'énergie totale et les conditions aux limites	9
2.6	Tester le passage entre conditions aux limites Robin et conditions ax limites Dirichlet et Neumann	11
2.6.1	Conditions aux limites Dirichlet	11
2.6.2	Conditions aux limites Neumann	12
3	Exercices générales	16
4	Partie informatique	18
4.1	Tests du code	19
5	Animations de la solution de l'équation d'ondes	21
6	Les ondes progressives à 2+1 dimensions	27
6.1	Quelques remarques sur les conditions initiales le long z	30
7	Quelques remarques sur le traitement informatique	31
8	Conclusions	32
A	Description des codes informatiques	32
A.1	Les codes en C	32
A.2	Les codes en Java	33
B	Les codes sources en C	33
B.1	Le code en C pour les conditions aux limites Robin	33
B.2	Le code en C pour les conditions aux limites Dirichlet	39
B.3	Le code en C pour les conditions aux limites Neumann	46
B.4	Le code en C pour la solution de l'équation d'ondes en 2+1 dimensions	52
C	Les codes en Java	59

1 Introduction générale et plan de l'exposé

Dans ce travail on étudie la solution de l'équation d'ondes avec différentes conditions aux limites, par des méthodes spectrales. On s'intéresse, plus particulièrement aux conditions aux limites Robin, qui généralisent les conditions aux limites plus connues de Dirichlet et Neumann.

Le plan de cet exposé est le suivant :

Dans la section 2 on trouve les fonctions propres et valeurs propres de l'opérateur spatial de l'équation d'ondes (1), lorsque l'on impose les conditions aux limites Robin ; et l'on emploie l'analyse dimensionnelle pour écrire l'équation sous forme adimensionnée.

Ce travail est le sujet des sous-sections 1.1 et 2.3. C'est alors que l'on se rend compte que la solution définie ainsi est incomplète, car elle explose exponentiellement vite, comme fonction du temps, que ce soit dans le futur ou le passé.

Une manière d'éliminer cette explosion est d'introduire les effets de la dispersion, qui sont étudiés dans la sous-section 2.4. On établit, ainsi, la relation — une inégalité — entre le coefficient du terme dispersif et le coefficient des conditions de Robin pour que la solution reste bornée pour toute valeur du temps. Un cas spécial est lorsque la relation devient une égalité, ce qui a comme conséquence que la solution est purement périodique.

Dans la sous-section 2.5 on étudie la conservation de l'énergie. On montre qu'il y a un flux d'énergie à travers les bords de l'intervalle, lorsque l'on applique les conditions aux limites de Robin, ce qui indique que le système n'est pas fermé. On calcule ce flux pour notre solution et l'on discute ses propriétés. Ainsi on peut comparer les propriétés du flux, provenant de la solution numérique avec celles attendues de la solution exacte et l'on peut déduire le nombre des termes de la série que l'on doit inclure, pour que l'accord soit correct.

Dans la sous-section 2.6 on étudie l'équivalence entre conditions aux limites Robin et Dirichlet, respectivement Neumann, lorsque $\Gamma \rightarrow 0$, respectivement $\Gamma \rightarrow \infty$ et l'on trouve les valeurs *finies* du paramètre Γ pour lesquelles les conditions Robin, de point de vue numérique, seraient équivalentes à celles de Dirichlet, respectivement Neumann.

L'étude de certains aspects théoriques est le sujet de la section 3 et l'approche informatique est employée comme outil expérimental.

Dans la section 4 on présente les tests du code qui montre sa validité, étape indispensable avant la production des données "réelles".

Dans la section 5 on discute comment employer le langage Java pour réaliser des animations de la solution de l'équation d'ondes, avec une dimension spatiale.

Dans la section 6 on étend l'étude au cas, où il y aurait une dimension spatiale supplémentaire, le long laquelle la solution pouvait avoir le profil d'une onde progressive.

Dans la section 7 on discute certains aspects informatiques particuliers, qui apparaissent pour deux dimensions spatiales.

Nos conclusions et perspectives sont présentées dans la section 8 on présente nos conclusions et les perspectives pour des suites possibles de ce travail.

Dans l'annexe A on présente les éléments, permettant d'employer les codes informatiques utilisés comme des "boîtes noires".

Dans l'annexe [codessources](#) B on inclut les codes eux-mêmes.

2 Partie théorique

2.1 Introduction

On cherche à résoudre l'équation d'ondes linéaire,

$$\frac{\partial^2 \phi}{\partial t^2} = v^2 \frac{\partial^2 \phi}{\partial x^2} \quad (1) \quad \text{ondes}$$

pour $x \in [0, L]$. On doit imposer des conditions initiales et des conditions aux limites.

— *Conditions initiales* :

$$\phi(x, 0) = F(x) \quad \text{et} \quad \frac{\partial \phi}{\partial t}(x, 0) = 0 \quad (2) \quad \text{initcond}$$

— *Conditions aux limites* :

$$\phi(0, t) + \gamma \frac{\partial \phi}{\partial x}(0, t) = 0 \quad \text{et} \quad \phi(L, t) + \gamma \frac{\partial \phi}{\partial x}(L, t) = 0 \quad (3) \quad \text{Robinbc}$$

où γ est une constante, qui a, nécessairement, les dimensions d'une longueur. Les conditions (3) s'appellent les conditions aux limites (homogènes) de Robin.

Il est utile de reformuler le problème en employant des expression sans dimensions—ainsi l'on pourra résoudre tout problème de cette classe.

On pose, ainsi,

$$u \equiv \frac{x}{L} \quad , \quad \tau \equiv t \frac{v}{L} \quad , \quad \Gamma \equiv \frac{\gamma}{L} \quad (4) \quad \text{dimanaly}$$

Les quantités, u, τ et Γ sont sans dimensions.

Le problème prend, alors, la forme suivante :

Théorème 1. *La solution de l'équation*

$$\frac{\partial^2 \phi}{\partial \tau^2} = \frac{\partial^2 \phi}{\partial u^2} \quad (5) \quad \text{ondes1}$$

avec conditions initiales

$$\phi(u, 0) = F(u) \quad \text{et} \quad \frac{\partial \phi}{\partial \tau}(u, 0) = 0 \quad (6) \quad \text{initcond}$$

et conditions aux limites

$$\phi(0, \tau) + \Gamma \frac{\partial \phi}{\partial u}(0, \tau) = 0 \quad \text{et} \quad \phi(1, \tau) + \Gamma \frac{\partial \phi}{\partial u}(1, \tau) = 0 \quad (7) \quad \text{Robinbc1}$$

existe et elle est unique. Mais elle diverge exponentiellement vite, dans le temps, avec taux $|\Gamma|$.

L'idée est d'exprimer $\phi(u, \tau)$ comme combinaison linéaire des fonctions propres de l'opérateur $-d^2/du^2$, qui satisfont les mêmes conditions aux limites.

On doit, alors, trouver fonctions propres et valeurs propres de cet opérateur.

2.2 Le problème spectral pour $-d^2/du^2$ avec conditions aux limites Robin

ctralRobin

La solution générale de l'équation

$$-\frac{d^2}{du^2}f = \lambda f \Leftrightarrow \frac{d^2}{du^2}f + \lambda f = 0 \quad (8) \quad \text{Robinspe}$$

pour $u \in [0, 1]$ est donnée par l'expression

$$f(u) = Ae^{iu\sqrt{\lambda}} + Be^{-iu\sqrt{\lambda}} \quad (9) \quad \text{solspec}$$

On impose, maintenant, les conditions aux limites

$$f(0) + \Gamma f'(0) = 0 = f(1) + \Gamma f'(1) \quad (10) \quad \text{Robinbc2}$$

et l'on obtient le système d'équations suivantes pour les coefficients A et B :

$$\begin{aligned} A(1 + i\Gamma\sqrt{\lambda}) + B(1 - i\Gamma\sqrt{\lambda}) &= 0 \\ A(1 + i\Gamma\sqrt{\lambda})e^{i\sqrt{\lambda}} + B(1 - i\Gamma\sqrt{\lambda})e^{-i\sqrt{\lambda}} &= 0 \end{aligned} \quad (11) \quad \text{ABeqs}$$

C'est un système homogène, par conséquent, la condition pour qu'il possède un nombre infini de solutions est

$$\begin{vmatrix} 1 + i\Gamma\sqrt{\lambda} & 1 - i\Gamma\sqrt{\lambda} \\ (1 + i\Gamma\sqrt{\lambda})e^{i\sqrt{\lambda}} & (1 - i\Gamma\sqrt{\lambda})e^{-i\sqrt{\lambda}} \end{vmatrix} = (1 + \Gamma^2\lambda)(e^{i\sqrt{\lambda}} - e^{-i\sqrt{\lambda}}) = 0 \quad (12) \quad \text{eigenval}$$

qui est satisfaite, lorsque $\lambda = -1/\Gamma^2$ ou $\lambda_n = n^2\pi^2$, avec n entier. On ne sait pas, encore, si $n = 0$ est une valeur possible.

On note que l'ensemble de ces valeurs est discret, comme l'on pouvait s'attendre pour un opérateur défini sur un espace compact.

Les fonctions propres correspondantes peuvent être déterminées en imposant les conditions (11), pour les valeurs de λ appropriées ; bien entendu, les deux équations n'étant pas, alors, indépendantes, il suffit d'en imposer une.

Ainsi l'on trouve, pour $\lambda = -1/\Gamma^2$, que

$$A\left(1 + i\Gamma\sqrt{-\frac{1}{\Gamma^2}}\right) + B\left(1 - i\Gamma\sqrt{-\frac{1}{\Gamma^2}}\right) = 0 \quad (13) \quad \text{negevalu}$$

ce qui implique que la fonction propre correspondante est donnée par l'expression

$$f_*(u) = A_*e^{-u/|\Gamma|} \quad (14) \quad \text{negeigen}$$

où A_* est un coefficient qui sera déterminé en imposant la normalisation de la fonction propre.

Pour $\lambda = n^2\pi^2$, on trouve que

$$A(1 + i\Gamma n\pi) + B(1 - i\Gamma n\pi) = 0 \Leftrightarrow A = -B \frac{1 - i\Gamma n\pi}{1 + i\Gamma n\pi} = -B \left(\frac{1 - \Gamma^2 n^2 \pi^2}{1 + \Gamma^2 n^2 \pi^2} - i \frac{2\Gamma n\pi}{1 + \Gamma^2 n^2 \pi^2} \right) = -B e^{-i\theta_n} \quad (15) \quad \boxed{\text{posevale}}$$

avec $\tan(\theta_n/2) = \Gamma n\pi$.

Par conséquent, la fonction propre correspondante est donnée par l'expression

$$f_n(u) = -B \left(e^{i(-\theta_n + n\pi u)} - e^{-in\pi u} \right) = -B e^{-i\frac{\theta_n}{2}} \left(e^{i(n\pi u - \frac{\theta_n}{2})} - e^{-i(n\pi u - \frac{\theta_n}{2})} \right) = A_n \sin \left(n\pi u - \frac{\theta_n}{2} \right) \quad (16) \quad \boxed{\text{poseigen}}$$

où A_n est le coefficient de normalisation.

L'on note que, pour $n = 0$, $f_0(u) \equiv 0$, par conséquent, il n'y a pas de valeur propre 0.

exonormneg

Exercice 1. Le coefficient A_* est déterminé par la condition

$$\int_0^1 du [f_*(u)]^2 = 1. \quad (17) \quad \boxed{\text{Astar}}$$

Trouver son expression.

Solution : L'éq. $\boxed{\text{Astar}}$ est équivalente à l'expression

$$A_*^2 \int_0^1 du e^{-2u/|\Gamma|} = 1 \Leftrightarrow A_* = \sqrt{\frac{2}{|\Gamma|(1 - e^{-2/|\Gamma|})}} \quad (18) \quad \boxed{\text{Astarsol}}$$

exonormpos

Exercice 2. Les coefficients A_n , $n = 1, 2, \dots$ sont déterminés, chacun, par la condition

$$\int_0^1 du [f_n(u)]^2 = 1 \quad (19) \quad \boxed{\text{An}}$$

Trouver l'expression pour A_n , comme fonction de n et θ_n .

Solution : L'éq. $\boxed{\text{An}}$ est équivalente à l'expression

$$\begin{aligned} A_n^2 \int_0^1 du \sin^2 \left(n\pi u - \frac{\theta_n}{2} \right) &= A_n^2 \int_0^1 du \frac{1}{2} (1 - \cos(2n\pi u - \theta_n)) \\ \frac{A_n^2}{2} - \frac{A_n^2}{2} \int_0^1 du \cos(2n\pi u - \theta_n) &= \frac{A_n^2}{2} \left(1 - \frac{1}{2\pi n} \sin(2n\pi u - \theta_n) \Big|_0^1 \right) = \\ \frac{A_n^2}{2} \left(1 - \frac{1}{2\pi n} (\sin(2\pi n - \theta_n) - \sin(-\theta_n)) \right) &= \frac{A_n^2}{2} = 1 \end{aligned} \quad (20) \quad \boxed{\text{Ansol}}$$

puisque la fonction sinus est périodique avec période 2π . Par conséquent on a que

$$\frac{A_n^2}{2} = 1 \Leftrightarrow A_n = \sqrt{2} \quad (21) \quad \boxed{\text{Ansol1}}$$

Exercice 3. Montrer que les fonctions, $f_*(u)$ et $f_n(u)$, avec les préfacteurs calculés lors des exercices précédents, sont orthogonales :

$$\begin{aligned} \int_0^1 du f_*(u) f_n(u) &= 0 \quad \forall n \in \mathbb{N} \\ \int_0^1 du f_n(u) f_m(u) &= \delta_{mn} \end{aligned} \quad (22) \quad \text{orthoeig}$$

Solution : La démonstration suit par calcul direct :

1. : L'orthogonalité entre $f_*(u)$ et $f_n(u)$: L'idée est de remplacer le sinus par la différence de deux exponentielles :

$$\begin{aligned} \int_0^1 du f_*(u) f_n(u) &= A_* A_n \int_0^1 du, e^{-u/|\Gamma|} \sin \left(n\pi u - \frac{\theta_n}{2} \right) = \\ \frac{A_* A_n}{2i} \int_0^1 du \left(e^{-i\theta_n/2} \int_0^1 du e^{u(-\frac{1}{|\Gamma|} + in\pi)} - e^{i\theta_n/2} \int_0^1 du e^{u(-\frac{1}{|\Gamma|} - in\pi)} \right) \end{aligned} \quad (23) \quad \text{fstarfno}$$

On évalue immédiatement chaque intégrale :

$$\int_0^1 du e^{u \left(-\frac{1}{|\Gamma|} + in\pi \right)} = \frac{1}{-\frac{1}{|\Gamma|} + in\pi} \left(e^{-\frac{1}{|\Gamma|} + in\pi} - 1 \right) \quad (24) \quad \text{I1fastfn}$$

La valeur de l'autre intégrale est obtenue par conjugaison complexe. Et, finalement, on trouve que

$$\int_0^1 du f_*(u) f_n(u) \propto \left(\tan \frac{\theta_n}{2} - \Gamma n\pi \right) = 0 \quad (25) \quad \text{fastfano}$$

puisque $\tan \theta_n/2 = \Gamma n\pi$.

Ayant résolu le problème spectral, on peut, maintenant, résoudre l'équation d'ondes.

2.3 La solution de l'équation d'ondes avec conditions aux limites Robin : Démonstration du Théorème ^{Solondes} I

DemoThm

La démonstration est constructive. Elle a, cependant, deux lacunes : (a) On n'a pas démontré que l'ensemble $\{f_*(u), f_n(u)\}$ est une base ; (b) on ne vas pas démontrer que la série, construite à partir des $f_n(u)$, converge.

On exprime la fonction, $\phi(u, \tau)$ comme combinaison linéaire de la fonction $f_*(u)$ et des fonctions $f_n(u)$:

$$\phi(u, \tau) = c_*(\tau) f_*(u) + \sum_{n=1}^N c_n(\tau) f_n(\tau) \quad (26) \quad \text{solserie}$$

Pour N fini l'expression est, certainement, bien définie. On va montrer que l'on peut calculer explicitement les coefficients $c_*(\tau)$ et $c_n(\tau)$, à partir des conditions initiales.

On remplace l'expression pour $\phi(u, \tau)$ dans l'éq. (5). Le membre de gauche devient :

$$\frac{\partial^2}{\partial \tau^2} \phi(u, \tau) = \frac{d^2 c_*(\tau)}{d\tau^2} f_*(u) + \sum_{n=1}^N \frac{d^2 c_n(\tau)}{d\tau^2} f_n(u) \quad (27) \quad \boxed{\text{LHS1}}$$

Le membre de droite à son tour, prend la forme :

$$\begin{aligned} \frac{\partial^2}{\partial u^2} \phi(u, \tau) &= c_*(\tau) \frac{d^2}{du^2} f_*(u) + \sum_{n=1}^N c_n(\tau) \frac{d^2}{du^2} f_n(u) = \\ &= c_* \frac{1}{\Gamma^2} f_*(u) - \sum_{n=1}^N n^2 \pi^2 c_n(\tau) f_n(u) \end{aligned} \quad (28) \quad \boxed{\text{RHS1}}$$

Les membres de gauche de ces équations étant égaux, les membres de droite devront l'être, aussi :

$$\frac{d^2 c_*(\tau)}{d\tau^2} f_*(u) + \sum_{n=1}^N \frac{d^2 c_n(\tau)}{d\tau^2} f_n(u) = c_* \frac{1}{\Gamma^2} f_*(u) - \sum_{n=1}^N n^2 \pi^2 c_n(\tau) f_n(u) \quad (29) \quad \boxed{\text{LHS1=RHS}}$$

On exploite, maintenant, les relations d'orthogonalité (22) ^{orthoeigenf} pour déduire un système d'équations différentielles ordinaires pour les coefficients, $c_*(\tau)$ et $c_n(\tau)$:

$$\begin{aligned} \frac{d^2}{d\tau^2} c_*(\tau) &= \frac{1}{\Gamma^2} c_*(\tau) \\ \frac{d^2}{d\tau^2} c_n &= -n^2 \pi^2 c_n(\tau) \end{aligned} \quad (30) \quad \boxed{\text{ODEc}}$$

Les solutions générales de ces équations sont données par les expressions :

$$\begin{aligned} c_*(\tau) &= C_+ e^{\tau/|\Gamma|} + C_- e^{-\tau/|\Gamma|} \\ c_n(\tau) &= K_n \cos(n\pi\tau) + L_n \sin(n\pi\tau) \end{aligned} \quad (31) \quad \boxed{\text{solODEc}}$$

Les coefficients, $C_{\pm}$, K_n et L_n sont déterminés par les conditions initiales :

$$\begin{aligned} \phi(u, 0) &= f(u) = (C_+ + C_-) f_*(u) + \sum_{n=1}^N K_n f_n(u) \\ \frac{\partial \phi}{\partial \tau} \phi(u, 0) &= \frac{C_+ - C_-}{\Gamma} f_*(u) + \sum_{n=1}^N n\pi L_n f_n(u) \end{aligned} \quad (32) \quad \boxed{\text{initcond}}$$

En exploitant l'orthogonalité des fonctions $f_*(u)$ et $f_n(u)$ on déduit un système d'équations pour les coefficients, qui, dans le cas actuel (6) ^{initcond1} est très simple :

$$\begin{aligned} C_+ &= C_- & L_n &= 0 \\ C_+ &= \frac{1}{2} \int_0^1 du F(u) f_*(u) & K_n &= \int_0^1 du F(u) f_n(u) \end{aligned} \quad (33) \quad \boxed{\text{iinitcond}}$$

et l'on déduit l'expression suivante pour $\phi(u, \tau)$:

$$\phi(u, \tau) = 2C_+ \cosh\left(\frac{\tau}{|\Gamma|}\right) f_*(u) + \sum_{n=1}^N K_n \cos n\pi\tau f_n(u) \quad (34) \quad \boxed{\text{sol}}$$

qui décrit une fonction qui diverge exponentiellement avec le temps, puisque C_+ ne peut s'annuler, génériquement. Ceci est le signe d'une description incomplète. Il faut bien comprendre que la divergence exponentielle avec le temps est un sujet totalement distinct de la convergence—ou non—de la série sur $K_n \cos n\pi\tau f_n(u)$.

Dans la section suivante on cherchera à décrire une manière de compléter la description.

2.4 Les effets de la dispersion

dispersion

Une manière de généraliser l'équation d'ondes, de façon à ce que celle-ci ait des solutions bornées en présence des conditions aux limites Robin est la suivante :

$$\frac{\partial^2 \phi}{\partial \tau^2} = \frac{\partial^2 \phi}{\partial u^2} + \beta \phi \quad (35) \quad \boxed{\text{dispersionPDE}}$$

On va montrer qu'il existe une valeur spéciale pour β , pour laquelle le terme en $\cosh(\tau/|\Gamma|)$ a un comportement totalement différent.

La démonstration est directe : on remplace dans l'équation (35) l'expression (26). La seule nouveauté est l'apparition de termes supplémentaires dans les équations pour les coefficients :

$$\begin{aligned} \frac{d^2}{d\tau^2} c_* &= \left(\frac{1}{\Gamma^2} + \beta \right) c_* \\ \frac{d^2}{d\tau^2} c_n &= (-n^2 \pi^2 + \beta) c_n \end{aligned} \quad (36) \quad \boxed{\text{ODEcnew}}$$

On se rend compte que, si $\beta \leq -1/\Gamma^2$, alors les solutions de toutes les deux équations sont bornées—elle décrivent une dépendance périodique dans le temps et, par conséquent, cette condition est nécessaire pour que $\phi(u, \tau)$ soit bornée. (Elle n'est pas suffisante, car elle n'assure pas la convergence de la série sur les modes de valeurs propres positives.)

Si l'on pose $\Omega^2 \equiv -\beta - (1/\Gamma^2)$ et $\omega_n^2 \equiv n^2 \pi^2 - \beta$, alors les solutions générales des équations (36) sont

$$\begin{aligned} c_*(\tau) &= C_+ \cos \Omega\tau + C_- \sin \Omega\tau \\ c_n(\tau) &= K_n \cos \omega_n\tau + L_n \sin \omega_n\tau \end{aligned} \quad (37) \quad \boxed{\text{solODEcnew}}$$

On se rend compte que le traitement est tout à fait semblable au cas du spectre infini avant : On trouve $C_- = 0 = L_n$ et

$$\begin{aligned} C_+ &= \int_0^1 du F(u) f_*(u) \\ K_n &= \int_0^1 du F(u) f_n(u) \end{aligned} \quad (38) \quad \boxed{\text{CplusKn}}$$

et l'expression suivante pour la solution de l'équation d'ondes

$$\phi(u, \tau) = C_+ \cos(\Omega\tau) f_*(u) + \sum_{n=1}^N K_n \cos(\omega_n \tau) f_n(u) \quad (39) \quad \text{solnew}$$

On note que cette expression décrit, également, le cas spécial $\Omega = 0$, lorsque $\beta = -1/\Gamma^2$ et la contribution de la fonction propre $f_*(u)$ décrit un fond, indépendant du temps.

C'est, cependant, utile de comprendre que font les conditions aux limites—et celles de Robin en particulier. Pour ceci, il faut étudier les quantités conservées grâce aux symétries globales, comme la translation dans le temps—ce qui est l'énergie totale.

2.5 L'énergie totale et les conditions aux limites

Il y a plusieurs façons de montrer que l'éq. (35) est invariante sous la transformation $t \rightarrow t + \delta t \equiv t'$; la plus simple est par calcul direct. Il est moins trivial de trouver la quantité conservée, c.à.d. l'énergie totale.

La difficulté principale est comment tenir compte des conditions aux limites.

La manière “standard” est de montrer que la quantité

$$E = \int_0^1 du \left\{ \frac{1}{2} \left(\frac{\partial \phi}{\partial \tau} \right)^2 + \frac{1}{2} \left(\frac{\partial \phi}{\partial u} \right)^2 - \frac{\beta}{2} \phi^2 \right\} \quad (40) \quad \text{energy}$$

est conservée, si ϕ est solution de l'éq. (35) à des contributions des bords près.

La démonstration est le résultat du calcul de $dE/d\tau$:

$$\begin{aligned} \frac{dE}{d\tau} &= \int_0^1 du \left\{ \frac{\partial \phi}{\partial \tau} \frac{\partial^2 \phi}{\partial \tau^2} + \frac{\partial \phi}{\partial u} \frac{\partial^2 \phi}{\partial \tau \partial u} - \beta \phi \frac{\partial \phi}{\partial \tau} \right\} = \\ &= \int_0^1 du \left\{ \frac{\partial \phi}{\partial \tau} \left(\frac{\partial^2 \phi}{\partial \tau^2} - \beta \phi \right) + \frac{\partial \phi}{\partial u} \frac{\partial^2 \phi}{\partial \tau \partial u} \right\} = \int_0^1 du \left\{ \frac{\partial \phi}{\partial \tau} \frac{\partial^2 \phi}{\partial \tau^2} + \frac{\partial \phi}{\partial u} \frac{\partial^2 \phi}{\partial \tau \partial u} \right\} = \int_0^1 du \frac{\partial}{\partial u} \left\{ \frac{\partial \phi}{\partial \tau} \frac{\partial \phi}{\partial \tau} \right\} \end{aligned} \quad (41) \quad \text{dE/dt}$$

On note que le terme, auquel on a abouti, n'est pas identiquement nul; c'est l'intégrale d'une dérivée totale. En identifiant l'intégrande, qui définit E , comme une densité d'énergie, ρ , on peut identifier l'argument de la dérivée totale comme une densité de courant, J et écrire l'équation précédente sous la forme

$$\frac{\partial \rho}{\partial \tau} + \frac{\partial}{\partial u} \left(-\frac{\partial \phi}{\partial \tau} \frac{\partial \phi}{\partial u} \right) = 0 \Leftrightarrow \frac{\partial \rho}{\partial \tau} + \text{div} \cdot \mathbf{J} = 0 \quad (42) \quad \text{currentc}$$

Ici, bien sûr, $\mathbf{J}$ ne possède qu'une composante. Ce courant $\mathbf{J}$ n'est autre que le pendant du vecteur de Poynting pour le champ électromagnétique.

Le terme de bord peut être explicitement calculé, en terme des conditions aux limites :

$$\int_0^1 du \frac{\partial}{\partial u} \left\{ \frac{\partial \phi}{\partial \tau} \frac{\partial \phi}{\partial \tau} \right\} = \left[\frac{\partial \phi}{\partial \tau} \frac{\partial \phi}{\partial u} \right] (1, \tau) - \left[\frac{\partial \phi}{\partial \tau} \frac{\partial \phi}{\partial u} \right] (0, \tau) \quad (43) \quad \text{boundary}$$

et l'on remarque qu'il ne s'annule identiquement que pour les conditions aux limites Neumann. Pour les conditions aux limites Dirichlet et Robin il y a un courant à travers les bords.

Par le traitement actuel on voudrait étudier quantitativement le comportement de ce courant, comme fonction du temps.

courant

Exercice 4. *Tracer*

$$J_{\text{bord}}(\tau) = - \left[\frac{\partial \phi}{\partial \tau} \frac{\partial \phi}{\partial u} \right] (1, \tau) + \left[\frac{\partial \phi}{\partial \tau} \frac{\partial \phi}{\partial u} \right] (0, \tau)$$

comme fonction du temps, pour différentes valeurs de β et de Γ , en utilisant l'expression ⁽³⁹⁾ pour $\phi(u, \tau)$.

Indication : Employer les conditions aux limites Robin pour simplifier $J_{\text{bord}}(\tau)$ et montrer qu'il est donné par l'expression

$$J_{\text{bord}}^{\text{Robin}} = \frac{1}{\Gamma} \left(\phi(1, \tau) \frac{\partial \phi}{\partial \tau}(1, \tau) - \phi(0, \tau) \frac{\partial \phi}{\partial \tau}(0, \tau) \right) = \frac{1}{2\Gamma} \frac{\partial}{\partial \tau} (\phi(1, \tau)^2 - \phi(0, \tau)^2) \quad (44)$$

JbordRob

Dans les exercices suivants on cherche à mieux comprendre ce que cette expression implique.

courantneg

Exercice 5. Commencer avec une condition initiale, qui correspondrait à la fonction propre $f_*(u)$, c.à.d. $\phi(u, 0) = f_*(u)$, $\partial \phi(u, 0)/\partial \tau = 0$. Quelle est l'expression pour $J_{\text{bord}}(\tau)$? Comment dépend-elle des paramètres β et Γ ?

Solution : Si la condition initiale est $\phi(u, 0) = f_*(u)$ et $\frac{\partial \phi}{\partial \tau}(u, 0) = 0$, alors on a que $C_* = 1$ et $C_n = 0$, ainsi

$$\phi(u, \tau) = f_*(u) \quad (45)$$

phiinitf

Par conséquent, elle ne dépend pas du temps, ainsi $J_{\text{bord}} = 0$, ce qui veut dire qu'il n'y a pas de flux d'énergie à travers les bords.

courantpos

Exercice 6. Même question, pour le cas où $\phi(u, 0) = f_n(u)$ et $\partial \phi(u, 0)/\partial \tau = 0$.

Solution : Dans ce cas, l'on a que $C_* = 0$ et $C_m = 1$ pour $m = n$ et zéro autrement. Ainsi

$$\phi(u, \tau) = \cos(\omega_n \tau) f_n(u) \quad (46)$$

phiinitf

Par conséquent

$$\begin{aligned} \frac{\partial \phi}{\partial u} &= \cos(\omega_n \tau) f'_n(u) \\ \frac{\partial \phi}{\partial \tau} &= -\omega_n \sin(\omega_n \tau) f_n(u) \end{aligned} \quad (47)$$

Jbordunm

Et l'expression pour le flux à travers le bord devient

$$J_{\text{bord}}(\tau) = - \left[\frac{\partial \phi}{\partial u} \frac{\partial \phi}{\partial \tau} \right] (1, \tau) + \left[\frac{\partial \phi}{\partial u} \frac{\partial \phi}{\partial \tau} \right] (0, \tau) = \omega_n \cos(\omega_n \tau) \sin(\omega_n \tau) (f'_n(1) f_n(1) - f'_n(0) f_n(0)) = \frac{\omega_n}{2\Gamma} \sin(2\omega_n \tau) (f_n(0)^2 - f_n(1)^2) \quad (48)$$

Jbordunm

2fpropres

Exercice 7. *Même question, pour le cas où $C_+ \neq 0$ et un seul coefficient $K_{n_0} \neq 0$; ou $C_+ = 0$ et deux coefficients, K_{n_1} et $K_{n_2} \neq 0$.*

Solution : Pour le premier cas, la solution $\phi(u, \tau)$ est donnée par l'expression suivante

$$\phi(u, \tau) = C_+ f_*(u) + K_{n_0} \cos(\omega_{n_0} \tau) f_{n_0}(u) \quad (49)$$

solCplus

Après un calcul direct et sans subtilités on trouve l'expression pour le flux d'énergie à travers les bords,

$$J_{\text{bord}}(\tau) = \frac{1}{\Gamma} \left\{ -C_+ K_{n_0} \omega_{n_0} \sin(\omega_{n_0} \tau) [f_*(1) f_{n_0}(1) - f_*(0) f_{n_0}(0)] - \frac{K_{n_0}^2}{2} \omega_{n_0} \sin(2\omega_{n_0} \tau) [f_{n_0}(1)^2 - f_{n_0}(0)^2] \right\} \quad (50)$$

JbordRob

Si l'on compare cette expression avec celle de l'éq. (48), on constate l'apparition d'un terme supplémentaire, proportionnel à $\sin(\omega_{n_0} \tau)$, ce qui conduit à l'apparition d'harmoniques.

2.6 Tester le passage entre conditions aux limites Robin et conditions ax limites Dirichlet et Neumann

DirNeuRob

On sait que, de point de vue mathématique, la limite $\Gamma \rightarrow 0$ correspond aux conditions aux limites Dirichlet, tandis que la limite $\Gamma \rightarrow \infty$ correspond aux conditions aux limites Neumann.

Mais les expressions pour la solution et les autres quantités, telles le flux à travers le bord, ou la relation de dispersion, ne semblent pas avoir des limites "simples".

On ne peut pas poser $\Gamma = 0$ ni introduire une valeur $\Gamma = \infty$. Par conséquent, on voudrait comprendre, pour quelles valeurs *finies* de Γ , de point de vue numérique, la solution ne peut pas se distinguer de celle obtenue, en résolvant directement les problèmes aux conditions aux limites correspondantes.

2.6.1 Conditions aux limites Dirichlet

DirBC

Le cas des conditions aux limites Dirichlet correspond à $\Gamma = 0$, c.à.d. $\phi(0, \tau) = 0 = \phi(1, \tau)$. Les fonctions propres de l'opérateur spatial, $-d^2/du^2$, sont

$$[f_n(u)]_{\text{Dir}} = \sqrt{2} \sin(n\pi u) \quad (51)$$

fnpropres

et les valeurs propres correspondantes sont

$$\lambda_n = n^2 \pi^2 \quad (52)$$

evalprop

avec $n = 1, 2, \dots$. On note que le spectre est défini-positif.

Alors la solution est donnée par l'expression

$$[\phi(u, \tau)]_{\text{Dir}} = \sum_{n=1}^{\infty} c_n \cos(\omega_n \tau) [f_n(u)]_{\text{Dir}} \quad (53)$$

solDir

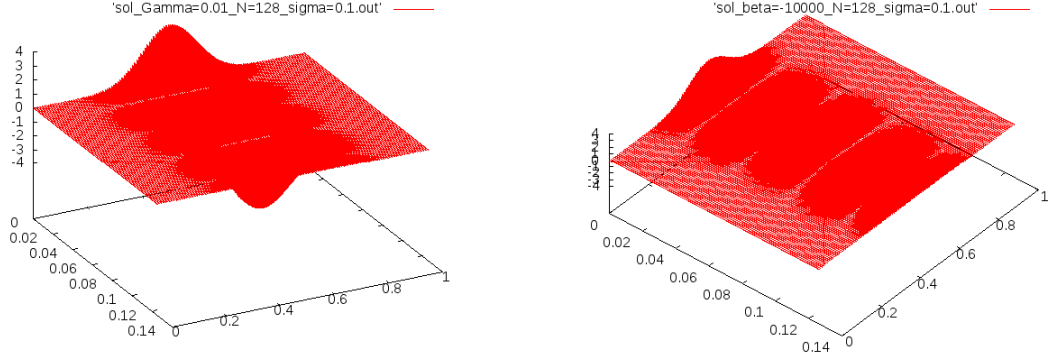


FIGURE 1 – La solution de l’équation d’ondes, pour conditions aux limites Dirichlet, et $\beta = -10000$ et conditions aux limites Robin, pour $\Gamma = 0.01$. Les paramètres satisfont, ainsi, la relation $\beta = -1/\Gamma^2$.

où,

$$\omega_n^2 = n^2\pi^2 - \beta \quad (54)$$

relation

On note que cette relation implique que, si $\beta > 0$, alors tous les modes ne peuvent pas se propager, seuls les modes, pour lesquels $n^2\pi^2 > \beta$ peuvent le faire.

Par contre, pour $\beta < 0$, tous les modes peuvent se propager. Seulement, maintenant, comme $\Gamma = 0$, il n’y a plus de relation particulière entre β et Γ ; l’on s’attend, cependant, que, si l’on prend $\beta \ll 0$, alors on devra pouvoir trouver une solution “similaire” à la solution de Robin– dans le volume. La question est, à quelle “distance du bord” les conditions aux limites se manifestent. On peut s’attendre à ce que cette distance sera proportionnelle à l’inverse de la masse de la particule relativiste, définie par la relation de dispersion, c.à.d. proportionnelle à $|\Gamma| = \sqrt{|\beta|}$.

On peut contrôler cette hypothèse en affichant la solution, pour conditions aux limites Dirichlet, pour $\beta \ll 0$ et la comparant avec la solution, obtenue pour conditions aux limites Robin, pour $0 < \Gamma \ll 1$. On affiche des exmples dans la fig. 2.6.1. On sait que $[\phi(0, \tau)]_{\text{Dir}} = 0 = [\phi(1, \tau)]_{\text{Dir}}$, et l’on cherche à contrôler que notre code peut confirmer ceci. Les résultats sont dans la fig. 2.

2.6.2 Conditions aux limites Neumann

Les conditions aux limites Neumann, $\partial\phi(0, \tau)/\partial u = 0 = \partial\phi(1, \tau)/\partial u$, correspdnent au cas limite $\Gamma \rightarrow \infty$ des conditions aux limites Robin.

Les fonctions propres de l’opérateur $-d^2/du^2$, qui satisfont les conditions aux limites Neu-

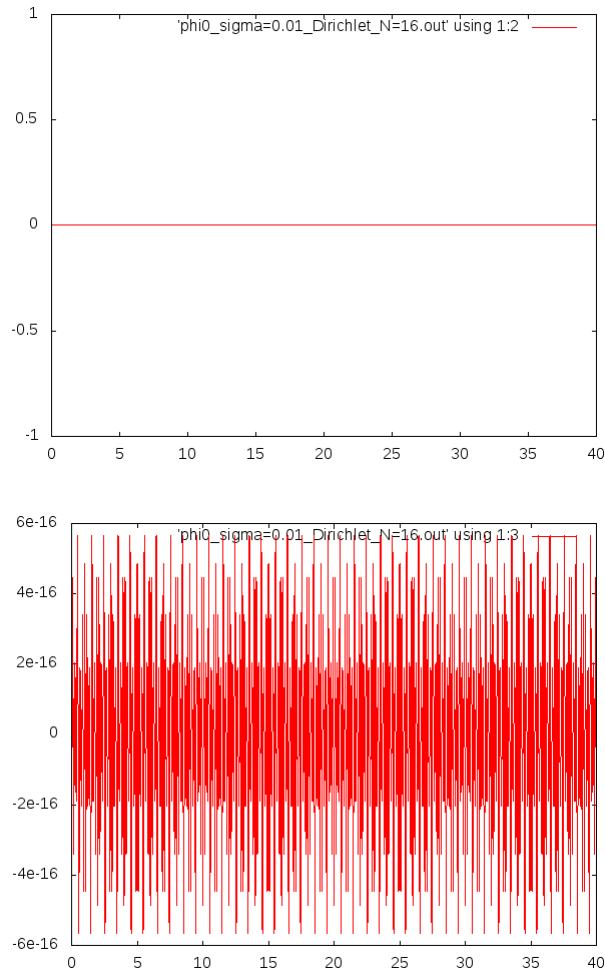


FIGURE 2 – La solution aux bords, pour les conditions aux limites Dirichlet. On note que $\sin(n\pi) \neq 0$ identiquement, mais oscille entre -6×10^{-16} et 6×10^{-16} .

bordDir

mann, sont solutions du problème

$$\begin{aligned} -\frac{d^2}{du^2}f(u) &= \lambda f(u) \\ f'(0) &= 0 = f'(1) \end{aligned} \quad (55) \quad \text{ProbNeu}$$

L'expression des $f(u)$ est donnée par

$$f(u) = Ae^{iu\sqrt{\lambda}} + Be^{-iu\sqrt{\lambda}} \Rightarrow f'(u) = i\sqrt{\lambda} (Ae^{iu\sqrt{\lambda}} - Be^{-iu\sqrt{\lambda}}) \quad (56) \quad \text{solProbNeu}$$

Si l'on impose les conditions aux limites Neumann, on trouve un système linéaire, homogène pour les coefficients A et B :

$$\begin{aligned} Ai\sqrt{\lambda} - Bi\sqrt{\lambda} &= 0 \\ Ai\sqrt{\lambda}e^{iu\sqrt{\lambda}} - Bi\sqrt{\lambda}e^{-iu\sqrt{\lambda}} &= 0 \end{aligned} \quad (57) \quad \text{Neusys}$$

et l'on cherche la condition pour que ce système possède un nombre infini de solutions.

Cette condition est que le déterminant des coefficients soit égal à zéro :

$$\lambda \sin(\sqrt{\lambda}) = 0 \Leftrightarrow \begin{cases} \lambda_* = 0 \\ \text{ou} \\ \lambda_n = n^2\pi^2, n = 1, 2, \dots \end{cases} \quad (58) \quad \text{detNeusy}$$

On vient, ainsi, de déterminer les valeurs propres de l'opérateur $-d^2/du^2$ avec conditions aux limites Neumann. Il faut déterminer les fonctions propres correspondantes.

1. *La fonction propre, $f_*(u)$* : La fonction propre, $f_*(u)$, qui correspond à la valeur propre $\lambda_* = 0$, est

$$f_*(u) = A_*u + B_* \quad (59) \quad \text{fastNeu}$$

Si l'on impose $f'_*(0) = 0 = f'_*(1)$, on trouve que

$$f_*(u) = B_* \quad (60) \quad \text{fastNeu1}$$

qui est une fonction propre admissible, puisqu'elle n'est pas identiquement égale à zéro. En plus, elle est normalisable, car

$$\|f_*(u)\|^2 = \int_0^1 du [f_*(u)]^2 = B_*^2 = 1 \Leftrightarrow B_* = \pm 1 \quad (61) \quad \text{fastNeu}$$

Exercice 8. Montrer que ce signe n'a pas d'importance physique et que l'on peut choisir, par conséquent, $B_* = 1$.

Solution :

signeBast

2. *Les fonctions propres, $f_n(u)$* : La fonction propre, $f_n(u)$, qui correspond à la valeur propre $\lambda_n = n^2\pi^2$, est donnée par l'expression

$$f_n(u) = A_n e^{in\pi u} + B_n e^{-in\pi u} \Rightarrow f'_n(u) = in\pi (A_n e^{in\pi u} - B_n e^{-in\pi u}) \quad (62) \quad \text{fnNeu}$$

Si l'on impose les conditions aux limites Neumann, puisque l'on emploie déjà une valeur propre, il suffit d'étudier une des deux équations, par exemple, celle pour $u = 0$:

$$f'_n(0) = in\pi(A_n - B_n) = 0 \Leftrightarrow A_n = B_n \Leftrightarrow f_n(u) = 2A_n \cos(n\pi u) \quad (63) \quad \text{fnNeu1}$$

puisque le cas $n = 0$ correspond à la fonction propre $f_*(u)$.

On peut, à nouveau, trouver que la condition, pour que les $f_n(u)$ soient normalisées,

$$\|f_n(u)\|^2 = \int_0^1 du [f_n(u)]^2 = 1 \Leftrightarrow A_n = 1/\sqrt{2} \Leftrightarrow f_n(u) = \sqrt{2} \cos(n\pi u) \quad (64) \quad \text{fnNeuon}$$

orthofnNeu

Exercice 9. Montrer que les $f_n(u)$ sont orthogonales.

Solution : La démonstration est par calcul direct :

$$\begin{aligned} \int_0^1 du f_n(u) f_m(u) &= 2 \int_0^1 du \cos(n\pi u) \cos(m\pi u) = \int_0^1 du \{ \cos[(n+m)\pi u] + \cos[(n-m)\pi u] \} = \\ &= \frac{1}{(n+m)\pi} (-\sin[(n+m)\pi u])|_0^1 + \frac{1}{(n-m)\pi} (-\sin[(n-m)\pi u])|_0^1 = \begin{cases} 1, & \text{si } n = m \\ 0, & \text{sinon} \end{cases} \end{aligned} \quad (65) \quad \text{solortho}$$

Par conséquent, la solution, $\phi(u, \tau)$, de l'équation d'ondes, qui satisfait les conditions aux limites Neumann, peut être écrite sous la forme

$$\phi(u, \tau) = C_*(\tau) f_*(u) + \sum_{n=1}^{\infty} C_n(\tau) f_n(u) \quad (66) \quad \text{solNeu}$$

Les fonctions $C_*(\tau)$ et $C_n(\tau)$ sont déterminées en remplaçant cette expression dans l'équation d'ondes :

$$\begin{aligned} \frac{\partial^2 \phi}{\partial \tau^2} &= \frac{\partial^2 \phi}{\partial u^2} + \beta \phi \Leftrightarrow \\ \frac{\partial^2 \phi}{\partial \tau^2} &= \frac{d^2 C_*}{d\tau^2} + \sum_{n=1}^{\infty} \frac{d^2 C_n}{d\tau^2} f_n(u) = \\ \frac{\partial^2 \phi}{\partial u^2} + \beta \phi &= \beta C_*(\tau) + \sum_{n=1}^{\infty} \overbrace{(-n^2 \pi^2 + \beta)}^{-(n^2 \pi^2 - \beta) \equiv -\omega_n^2} C_n(\tau) f_n(u) \Leftrightarrow \\ \frac{d^2 C_*}{d\tau^2} &= \beta C_* \\ \frac{d^2 C_n}{d\tau^2} &= -\omega_n^2 C_n \end{aligned} \quad (67) \quad \text{CtauNeu}$$

Ici il est crucial que $\beta < 0$, ce qui implique que la solution pour $C_*(\tau)$ n'explose pas dans le temps et que $\omega_n^2 > 0$.

Puisque $\beta < 0$, on peut écrire $\beta \equiv -1/\Gamma^2$, avec Γ réel. Ce Γ n'a rien à avoir avec le Γ des conditions aux limites de Robin—on est dans le cadre des conditions aux limites Neumann !

On trouve, ainsi, que

$$\begin{aligned} C_*(\tau) &= C_*^{(1)} \cos\left(\frac{\tau}{|\Gamma|}\right) + C_*^{(2)} \sin\left(\frac{\tau}{|\Gamma|}\right) \\ C_n(\tau) &= C_n^{(1)} \cos(\omega_n \tau) + C_n^{(2)} \sin(\omega_n \tau) \end{aligned} \quad (68)$$

et les coefficients, $C_*^{(1)}, C_*^{(2)}$ et $C_n^{(1)}, C_n^{(2)}$ sont déterminés par les conditions initiales, $\phi(0, \tau) \equiv F(u)$ et $\partial\phi(0, \tau)/\partial\tau \equiv G(u)$.

Exercice 10. *Ecrire les expressions pour les coefficients, dans le cas général.*

Solution :

Pour le cas particulier, où $\phi(u, 0) = F(u)$, $\partial\phi(u, 0)/\partial\tau = 0$, on trouve les expressions suivantes :

$$\begin{aligned} C_*^{(1)} &= \int_0^1 du F(u) \\ C_*^{(2)} &= 0 \\ C_n^{(1)} &= \int_0^1 du F(u) f_n(u) \\ C_n^{(2)} &= 0 \\ \phi(u, \tau) &= C_*^{(1)} \cos \frac{\tau}{|\Gamma|} + \sum_{n=1}^{\infty} C_n^{(1)} (\cos \omega_n \tau) f_n(u) \end{aligned} \quad (69)$$

Les expressions pour l'amplitude de l'onde aux bords sont

$$\begin{aligned} \phi(0, \tau) &= C_*^{(1)} \cos \frac{\tau}{|\Gamma|} + \sqrt{2} \sum_{n=1}^{\infty} C_n^{(1)} \cos \omega_n \tau \\ \phi(1, \tau) &= C_*^{(1)} \cos \frac{\tau}{|\Gamma|} + \sqrt{2} \sum_{n=1}^{\infty} C_n^{(1)} (-)^n \cos \omega_n \tau \end{aligned} \quad (70)$$

où l'on a utilisé la notation $(-)^n \equiv (-1)^n$. On affiche, dans la fig. [3](#), la solution pour l'équation d'ondes, lorsque l'on applique les conditions aux limites Neumann et l'on compare ses valeurs à la solution, lorsque l'on applique les conditions aux limites Robin, pour le cas $\beta = -1/\Gamma^2 = 0.01$. On note une accord impressionnant=sauf dans certains endroits.

3 Exercices générales

exogen
dimrest

Exercice 11. *Exprimer la solution de l'équation d'ondes, pour les trois cas de conditions aux limites, en termes des quantités “physiques” (L , v et γ) et discuter à quoi correspond le paramètre β .*

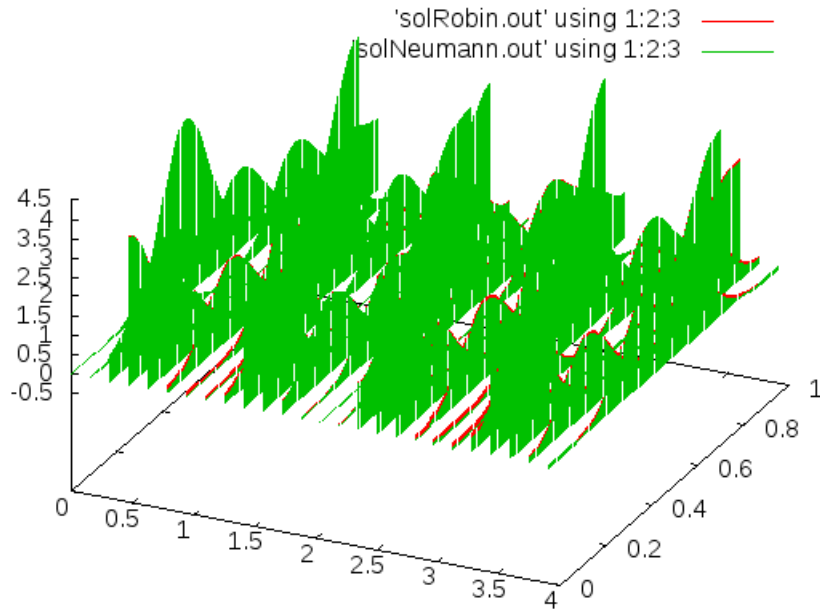


FIGURE 3 – La solution de l'équation d'ondes, pour les conditions aux limites de type Robin (points rouges), pour $\Gamma = 10.0$, $\beta = -0.01$, ainsi que pour les conditions aux limites de type Neumann, pour la même valeur de β . On remarque un accord impressionnant, qui, en principe, est parfait, dans la limite $\Gamma \rightarrow \infty$.

RobinNeu

Solution : Le dictionnaire entre u, τ et γ d'une part et x, t et Γ , d'autre part, est établi par les relations (4).

Pour la solution avec conditions aux limites Dirichlet ce dictionnaire implique que

$$[\phi(x, t)]_{\text{Dir}} = [\phi\left(\frac{x}{L}, t\frac{v}{L}\right)]_{\text{Dir}} = \sum_{n=1}^{\infty} c_n \cos\left(t\frac{v}{L}\sqrt{n^2\pi^2 - \beta}\right) \sqrt{2} \sin\left(n\pi\frac{x}{L}\right) \quad (71)$$

Exercice 12. Identifier l'expression pour le vecteur d'onde, k et écrire la relation de dispersion, $\omega = \omega(k)$ pour les trois cas de conditions aux limites. Si l'on s'intéresse à la propagation d'ondes de longueur d'onde de l'ordre du millimètre, quelles sont les contraintes sur les paramètres ?

Solution : On peut identifier le vecteur d'onde, k_n , du mode n en regardant la fonction propre $f_n(u)$, pour les trois classes de conditions aux limites :

$$k_n = n\pi \quad (72)$$

(dans le cas, où l'on restitue les "unités physiques", $k_n = n\pi/L$)

La pulsation, ω_n , correspondante, est donnée par l'expression

$$\omega_n(k_n) = \begin{cases} \sqrt{k_n^2 - \beta} & \text{Dir/Neu} \\ \sqrt{k_n^2 - \left(\beta + \frac{1}{\Gamma^2}\right)} & \text{Robin} \end{cases} \quad (73)$$

Exercice 13. La relation de dispersion peut être interprétée comme la relation entre énergie et impulsion de l'excitation à une particule, dont l'onde serait la superposition cohérente à un très grand nombre. Quelle est la masse de cette particule, pour le cas des trois conditions aux limites ? Est-elle bien définie, ou dépend-elle du mode de propagation ?

Solution :

Exercice 14. Facultatif : Quelles des trois conditions aux limites sont compatibles avec l'invariance sous transformations de Lorentz,

$$\begin{aligned} \tau' &= \tau \cosh \varphi + u \sinh \varphi \\ u' &= \tau \sinh \varphi + u \cosh \varphi \end{aligned} \quad (74)$$

qui sont symétries de l'équation d'ondes ? Dans ces notations $v/c = \tanh \varphi$.

Solution :

4 Partie informatique

Le traitement informatique a été réalisé par le code en C, `prog_tp0.c`, donné en annexe. Ici on présentera, tout d'abord, les tests, que le code a passé avec succès et, par la suite, on fournira les éléments pour l'entrée et la sortie plus en détail, ainsi que l'on décrira la structure des fonctions.

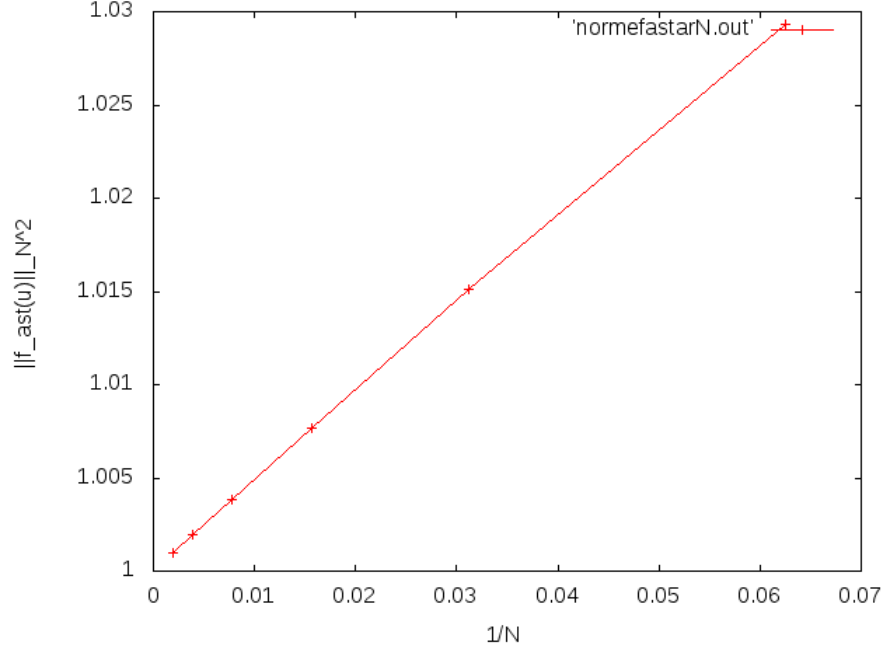


FIGURE 4 – La norme (au carré) de la fonction $f_*(u)$, calculée par la méthode des trapèzes, pour $N = 16, 32, 64, 128, 256$ et 512 points intermédiaires, affichée comme fonction de $1/N$.

normefst

4.1 Tests du code

Pour s'assurer que le code fonctionne correctement, on a réalisé les tests suivants :

1. *Normalisation de la fonction propre $f_*(u)$.* L'évaluation de l'expression

$$||f_*(u)||^2 = \int_0^1 du [f_*(u)]^2 \quad (75)$$

normefst

est réalisée par une méthode d'intégration numérique, c.à.d. on pose $u_j \equiv j/(N+1)$, $j = 0, 1, 2, 3, \dots, N+1$, où l'on évalue la fonction à intégrer, $[f_*(u)]^2$ et l'on utilise des coefficients, $\{s_j\}$, qui dépendent de la méthode concrète employée, pour trouver une valeur approchée pour l'intégrale (75) :

$$(||f_*(u)||^2)_N \equiv \sum_{j=0}^{N+1} s_j [f_*(u_j)]^2 \quad (76)$$

normefst

Le résultat de ce test est affiché dans la fig. 4.

2. *Normalisation des fonctions propres, $f_n(u)$.* On affiche, de la même façon, l'approximation à la norme, $||f_n(u)||_N^2$ des fonctions propres, du spectre positif. Pour ne pas surcharger le dessin, on ne montre que l'évolution, en $1/N$ pour $||f_1||_N^2$, $||f_{N/2}||_N^2$ et $||f_N||_N^2$ dans la fig. 5.

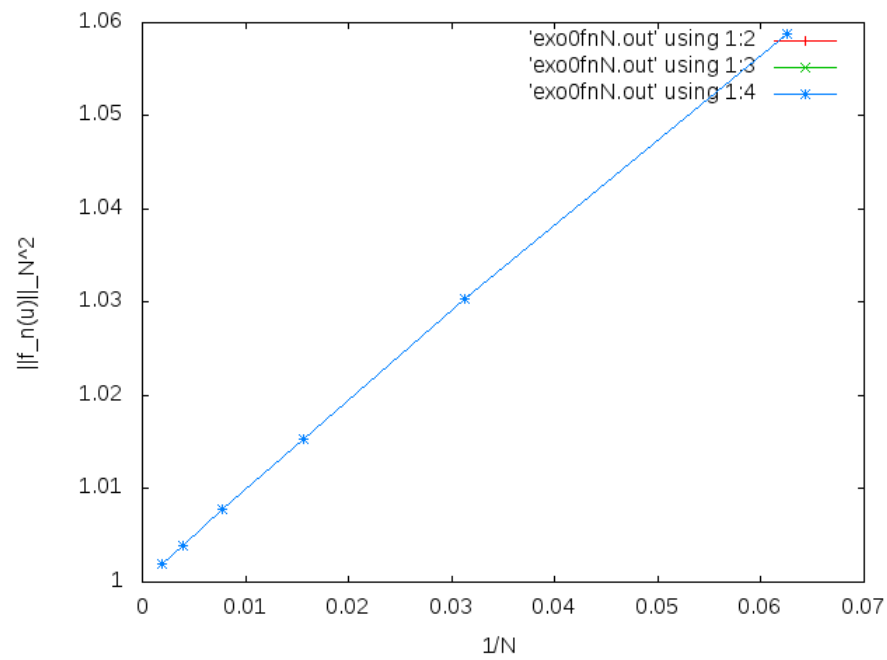


FIGURE 5 – La norme (au carré) des fonctions $f_1(u)$, $f_{N/2}(u)$ et $f_N(u)$, calculée par la méthode des trapèzes, pour $N = 16, 32, 64, 128, 256$ et 512 points intermédiaires, affichée comme fonction de $1/N$.

normefnN

3. *Réconstruction de la condition initiale* : Après avoir testé que les fonctions propres sont correctement représentées par le code, on doit tester que la condition initiale l'est, également. En fait, son calcul teste que l'orthogonalité des fonctions propres est, aussi, correctement reproduite. Ainsi, dans la fig. 6, on affiche la condition initiale, évaluée aux points $u_j = j/(N + 1), j = 0, 1, 2, 3, \dots, N + 1$ (courbe rouge) ainsi que le résultat de l'évaluation par les sommes finies (courbe verte),

$$\begin{aligned}
 [\phi(u_j, 0)]_N &\equiv [c_*]_N + \sum_{n=1}^N [c_n]_N f_n(u_j) \\
 [c_*]_N &\equiv \sum_{j=0}^{N+1} s_j \phi(u_j, 0) f_*(u_j) \\
 [c_n]_N &\equiv \sum_{j=0}^{N+1} s_j \phi(u_j, 0) f_n(u_j)
 \end{aligned} \tag{77}$$

4. *Test de la conservation locale de l'énergie par le calcul du flux à travers les bords* : On voudrait tester que le code peut reproduire l'expression (44) pour le flux à travers les bords.

Le test le plus simple est celui, où $\phi(u, 0) = f_*(u)$ et $\partial\phi(u, 0)/\partial\tau = 0$ (pour $\Omega^2 = 0$) puisque, dans ce cas, $\phi(u, \tau) = f_*(u)$ et, par conséquent, $J_{\text{bord}}(\tau) \equiv 0$.

On affiche $[J_{\text{bord}}(\tau)]_N$, calculé par le code, pour $N = 16 - 1024$ dans la fig. 7. On note que, à partir de $N = 512$, l'amplitude croît, au lieu de diminuer, tout en affichant des oscillations-symétriques-autour de la "vraie" valeur, zéro.

5. *Afficher $\phi(0, \tau)$ et $\phi(1, \tau)$ qui représentent les valeurs de la solution de l'équation d'onde aux bords*. Les conditions aux limites Robin déterminent de manière indirecte la valeur de la solution aux bords. On l'affiche, pour différentes valeurs des paramètres, dans les figs. 8 et 9. En fait il suffit d'afficher la solution à un des bords.
6. *Comprendre les propriétés de la solution $[\phi(u, \tau)]_N$, à partir de son graphe et, en particulier les cas limites, $\Gamma \rightarrow 0$ (Dirichlet) et $\Gamma \rightarrow \infty$ (Neumann) et, pour une valeur de N donnée, à partir de quelles valeurs de Γ commencent les domaines de chaque classe de conditions aux limites*. On affiche la solution dans la fig. 10.

5 Animations de la solution de l'équation d'ondes

Une onde représente un mouvement périodique aussi bien dans l'espace que dans le temps, ce qui mène, naturellement, à imaginer la construction d'animations pour afficher la solution.

A cette fin on va employer le langage Java pour réaliser des animations des solutions de l'équation d'ondes à une dimension spatiale.

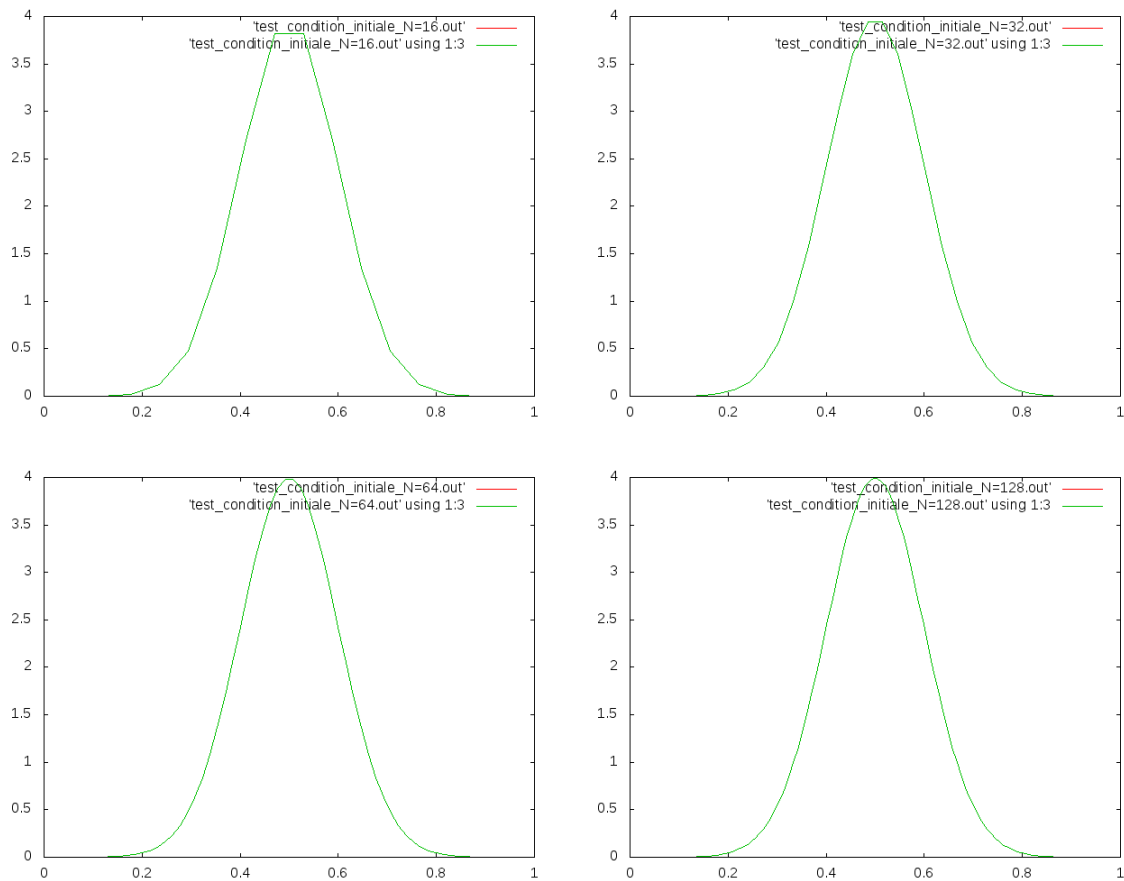


FIGURE 6 – Figures, qui montrent la relation entre la condition initiale, comme évaluée par la bibliothèque mathématique du compilateur C (courbe rouge) et par le formalisme de l'éq. (77), pour $N = 16, 32, 64, 128$. On remarque que la courbe verte et la courbe rouge se confondent et c'est juste le nombre fini de points qui fait que les courbes ne sont pas lisses, pour $N = 16, 32$; elle deviennent lisse pour $N > 32$.

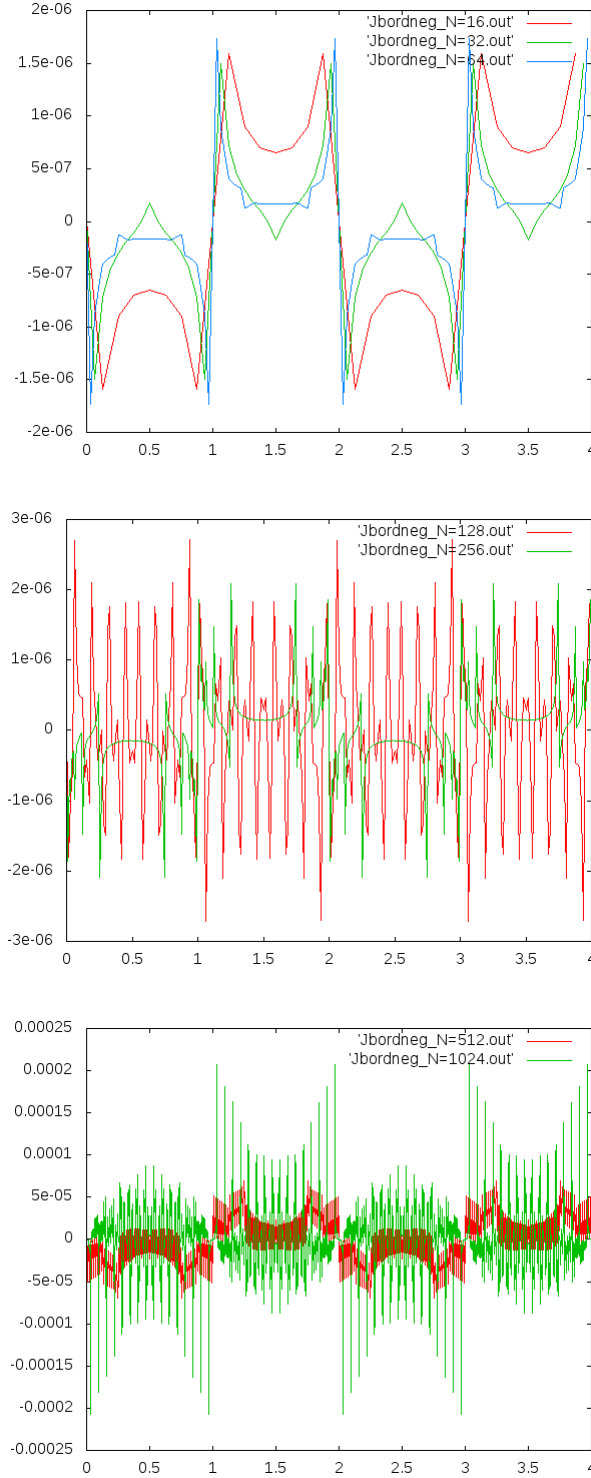


FIGURE 7 – Le flux à travers les bords, $[J_{\text{bord}}]_N$, pour le cas de la condition initiale $\phi(u, 0) = f_*(u)$, $\partial\phi(u, 0)/\partial u = 0$, pour $N = 16 - 1024$. On remarque que les courbes affichent des oscillations, symétriques autour de la “vraie” valeur, zéro, mais l’amplitude de ces oscillations ne décroît pas de façon monotone, lorsque N croît.

ords_sigma

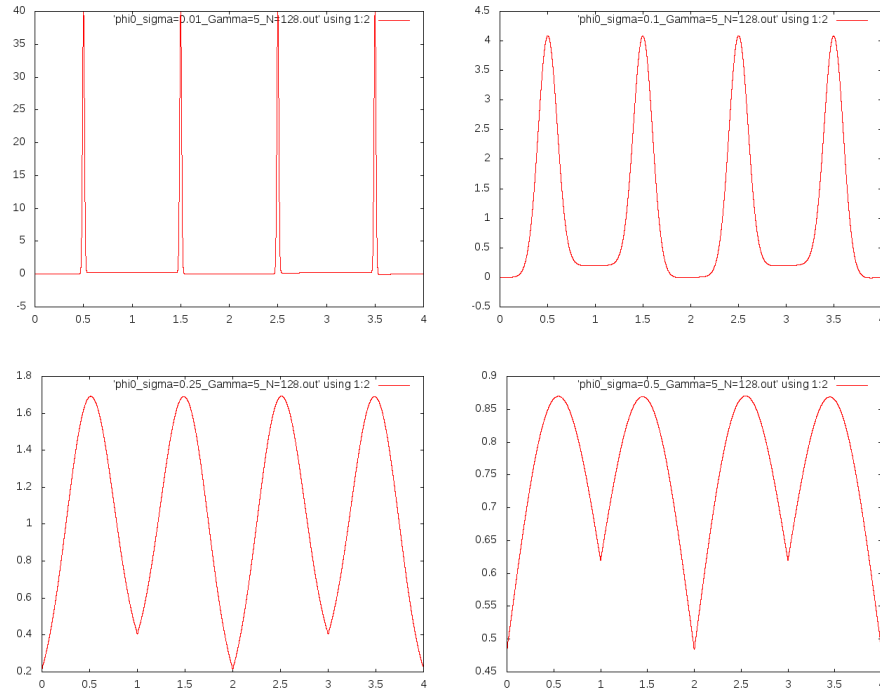


FIGURE 8 – La solution $\phi(0, \tau)$, pour $\sigma = 0.01, 0.1, 0.25, 0.5$ pour $\Gamma = 5$.

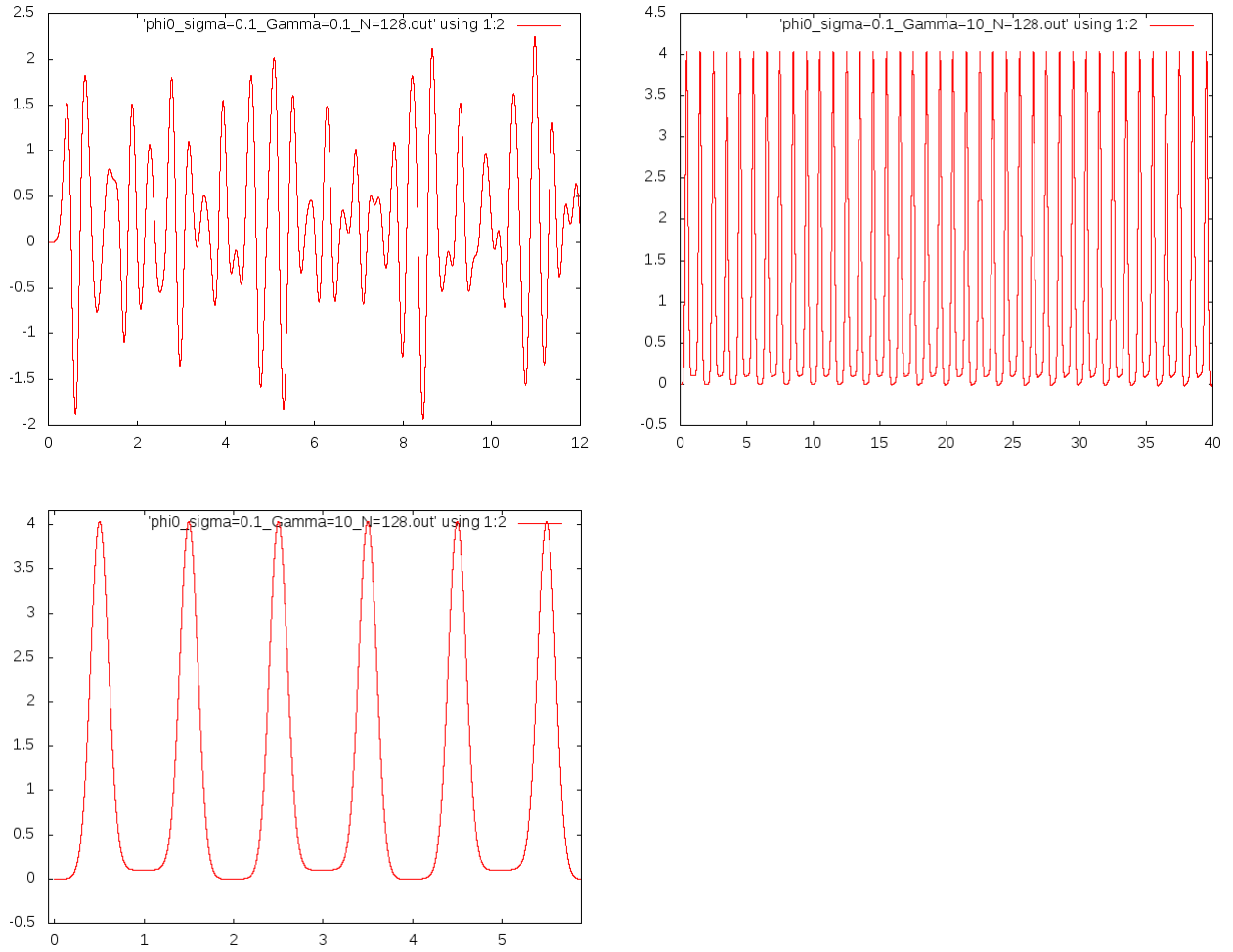


FIGURE 9 – La solution $\phi(0, \tau)$, pour $\Gamma = 0.1, 10$ pour $\sigma = 0.1$. Pour $\Gamma = 10$ on affiche l'agrandissement, qui montre des fortes similarités avec le cas $\Gamma = 500$ de la figure précédente.

bords_Ga

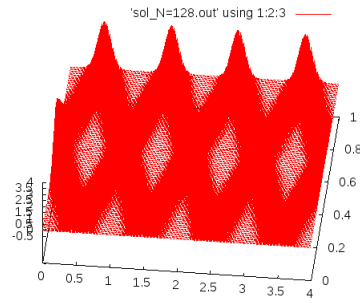
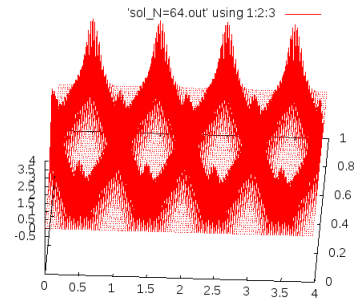
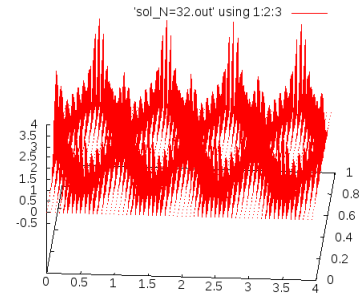
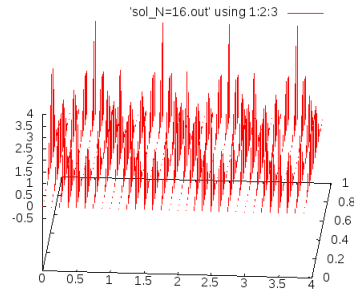


FIGURE 10 – La solution, $[\phi(u, \tau)]_N$, pour $N = 16, 32, 64, 128$ et $\Gamma = 500$.

solN

Un exemple est réalisé par le code `Example5Applet.java`, qui est fourni en annexe [codeJava](#). Le code, en fait, réalise une solution particulière de l'équation d'ondes, sans dispersion, c.à.d. de l'équation

$$\frac{\partial^2 \phi}{\partial \tau^2} = \frac{\partial^2 \phi}{\partial u^2} \quad (78) \quad \text{eqondes1}$$

à savoir celle de la superposition de deux paquets d'ondes gaussiennes, qui voyagent dans des directions opposées :

$$\phi(u, \tau) = A \exp\left(-\frac{(u - \tau)^2}{2\sigma} + \phi_A\right) + B \exp\left(-\frac{(u + \tau)^2}{2\sigma} + \phi_B\right) \quad (79) \quad \text{solondes}$$

où l'on a imposé des conditions aux limites périodiques, par les commandes, qui déclarent les variables `phase1` et `phase2` dans cette partie du code :

```
int phase1 = (x-v*frame)%d.width + d.width/10;
int phase2 = (x+v*frame)%d.width - d.width/10;
double sigma1 = 70.0;
double ph1 = sqr(phase1)/(double)(sigma1);
double ph2 = sqr(phase2)/(double)(sigma1);
int y1 = (int)(10*d.height*Math.exp(-ph1)/Math.sqrt((double)sigma1));
int y2 = (int)(10*d.height*Math.exp(-ph2)/Math.sqrt((double)sigma1));

int y = (y1 + y2)/2;
```

Exercice 15. Montrer que les déclarations des variables `phase1` et `phase2`, effectivement réalisent les conditions aux limites périodiques, sachant que la commande `%d.width` calcule le reste de la division par l'entier `d.width`.

Solution :

Exercice 16. La commande `int y = (y1+y2)/2;` impose une relation de phase fixe entre les deux solutions y_1 et y_2 . Discuter si cette relation est satisfaite par les conditions aux limites Robin.

Solution :

6 Les ondes progressives à 2+1 dimensions

Jusqu'à maintenant on a étudié la solution de l'équation d'ondes dans un domaine borné—maintenant on va ajouter une dimension spatiale non-compacte, pour pouvoir étudier des ondes progressives.

Ainsi, l'équation prendra la forme suivante :

$$\frac{\partial^2 \phi}{\partial \tau^2} = \frac{\partial^2 \phi}{\partial u^2} + \beta \phi + \frac{\partial^2 \phi}{\partial z^2} \quad (80) \quad \text{ondes2p1}$$

et l'on cherche des solutions de la forme suivante :

$$\phi(u, z, \tau) = f(u)e^{i(kz - \omega\tau)} \quad (81) \quad \text{sol2plus1}$$

où la fonction $f(u)$ est supposée satisfaire les conditions aux limites de type Robin, à savoir

$$\begin{aligned} f(0) + \Gamma f'(0) &= 0 \\ f(1) + \Gamma f'(1) &= 0 \end{aligned} \quad (82) \quad \text{fRobin}$$

On cherche, ainsi, à comprendre, si l'on peut trouver des solutions, de la forme ^{sol2plus1}(98), où les paramètres ω et k prennent des valeurs réelles. La raison est que c'est lorsque ceci est possible que l'on peut dire que la solution décrit la propagation d'ondes progressives le long z . On imposera des conditions aux limites dans la direction z à la fin.

En remplaçant l'expression ^{sol2plus1}(98) dans l'éq. ^{ondes2plus1}(80) on trouve l'équation suivante pour $f(u)$:

$$-\omega^2 f = -\frac{d^2}{du^2} f + \beta f - k^2 f \Leftrightarrow \frac{d^2}{du^2} f + \underbrace{(\beta + \omega^2 - k^2)}_{\lambda} f = 0 \Leftrightarrow f(u) = Ae^{iu\sqrt{\lambda}} + Be^{-iu\sqrt{\lambda}} \quad (83) \quad \text{eqf2plus1}$$

C'est exactement le même problème que l'on a eu à traiter dans la section ^{spectralRobin}2.2. On a besoin de l'expression pour $f'(u)$:

$$f'(u) = i\sqrt{\lambda}Ae^{iu\sqrt{\lambda}} - i\sqrt{\lambda}Be^{-iu\sqrt{\lambda}} \quad (84) \quad \text{fprime2plus1}$$

Les conditions de Robin ^{fRobin}(82) donnent :

$$\begin{aligned} A + B + \Gamma i\sqrt{\lambda}A - \Gamma i\sqrt{\lambda}B &= 0 \Leftrightarrow (1 + i\Gamma\sqrt{\lambda})A + (1 - \Gamma i\sqrt{\lambda})B = 0 \\ Ae^{iu\sqrt{\lambda}} + Be^{-iu\sqrt{\lambda}} + \Gamma i\sqrt{\lambda} \left(Ae^{iu\sqrt{\lambda}} - Be^{-iu\sqrt{\lambda}} \right) &= 0 \Leftrightarrow Ae^{iu\sqrt{\lambda}}(1 + \Gamma i\sqrt{\lambda}) + Be^{-iu\sqrt{\lambda}}(1 - \Gamma i\sqrt{\lambda}) = 0 \end{aligned} \quad (85) \quad \text{fRobin1}$$

d'où l'on déduit la condition sur les valeurs propres, λ :

$$(1 + \Gamma^2\lambda) \sin \sqrt{\lambda} = 0 \Leftrightarrow \begin{cases} \lambda \equiv \lambda_* = -1/\Gamma^2 \\ \lambda \equiv \lambda_n = n^2\pi^2, \quad n = 1, 2, \dots \end{cases} \quad (86) \quad \text{lambdaev}$$

tout comme avant, ainsi que les fonctions propres,

$$\begin{aligned} f_* &= A_* \sin \left(n\pi u - \frac{\theta_n}{2} \right) \\ f_n &= A_n \sin(n\pi u) \end{aligned} \quad (87) \quad \text{fpropres}$$

(avec $\tan \theta_n = \Gamma n\pi$.)

Les effets nouveaux de la présence de la dimension supplémentaire le long z apparaissent, lorsque l'on se rend compte que $\lambda = k^2 - \omega^2 - \beta$. Les expressions pour les valeurs propres ont les conséquences suivantes pour l'expression de la solution :

1. De l'expression pour λ_* on déduit que

$$\lambda_* = -\frac{1}{\Gamma^2} = \beta + \omega_*^2 - k_*^2 \Leftrightarrow \omega_*^2 = k_*^2 - \beta - \frac{1}{\Gamma^2} \quad (88) \quad \text{omegastar}$$

L'expression pour $\phi_*(u, z, \tau)$ prend, ainsi, la forme

$$\phi_*(u, z, \tau) = A_* \sin\left(n\pi u - \frac{\theta_n}{2}\right) e^{i(k_* z - \omega_* \tau)} \quad (89) \quad \text{phistar}$$

En particulier, si $\beta = -1/\Gamma^2$, alors,

$$\omega_* = |k_*| \quad (90) \quad \text{omegastar}$$

et l'on a une onde plane, qui se propage, sans dispersion, le long la direction z , tandis qu'une onde stationnaire remplit la direction u .

Si $\beta \neq -1/\Gamma^2$, on se rend compte que l'on peut écrire l'éq. (88) peut être écrite comme

$$k_*^2 = \omega_*^2 + \beta + \frac{1}{\Gamma^2} \quad (91) \quad \text{kstarnew}$$

et que, si $\beta > -1/\Gamma^2$, alors k_* est réel et l'on a une propagation avec phénomènes de dispersion, puisque la relation n'est pas linéaire ; si $\beta < -1/\Gamma^2$, alors il y a une *fréquence de coupure*,

$$\omega_{\text{coup}} = \sqrt{-\beta - \frac{1}{\Gamma^2}} \quad (92) \quad \text{omegacou}$$

telle que, si $\omega < \omega_{\text{coup}}$, alors $k_* = i|k_*|$, et, par conséquent,

$$\phi_*(u, z, \tau) = f_*(u) e^{-|k_*|z - i\omega\tau} \quad (93) \quad \text{phistar}$$

ce qui décrit une onde exponentiellement amortie le long z avec une longueur d'amortissement, ℓ_* , donnée par l'expression

$$\ell_* = \frac{2\pi}{|k_*|} \quad (94) \quad \text{ellstar}$$

2. De l'expression pour $\lambda_n = n^2\pi^2$, l'on déduit que

$$\lambda_n^2 = n^2\pi^2 = \beta + \omega_n^2 - k_n^2 \Leftrightarrow \omega_n^2 = k_n^2 + n^2\pi^2 - \beta \quad (95) \quad \text{omegankn}$$

On remarque que la présence de la dimension supplémentaire le long z a ajouté un terme non-négatif à la relation pour ω_n^2 , que l'on a trouvée en son absence ; et que, pourvu que $\beta < 0$, les modes, étiquetés par l'entier n sont toujours des modes qui se propagent ; pourvu que, cette fois-ci,

$$\omega_n > n^2\pi^2 - \beta \equiv \omega_{\text{coup},n}^2 \quad (96) \quad \text{omegacou}$$

La fonction propre, $\phi_n(u, z, \tau)$ est, ainsi, donnée par l'expression

$$\phi_n(u, z, \tau) = A_n \sin(n\pi u) e^{i(k_n z - \omega_n \tau)} \quad (97) \quad \text{phinfull}$$

Par conséquent, la solution générale de l'éq. (80) est donnée par l'expression

$$\phi(u, z, \tau) = C_* \phi_*(u, z, \tau) + \sum_{n=1}^{\infty} C_n \phi_n(u, z, \tau) \quad (98) \quad \text{sol2plus}$$

où les coefficients C_* et C_n sont déterminés par la condition initiale, $\phi(u, z, \tau)$:

$$\begin{aligned} C_* &= \int_0^1 du \int_{-\infty}^{\infty} dz \phi(u, z, 0) \phi_*(u, z, 0) \\ C_n &= \int_0^1 du \int_{-\infty}^{\infty} dz \phi(u, z, 0) \phi_n(u, z, 0) \end{aligned} \quad (99) \quad \text{CstarCnn}$$

De point de vue pratique, bien entendu, on ne peut calculer une intégrale entre $-\infty$ et $+\infty$ à l'ordinateur.

D'autre part, en pratique, on ne mesure pas infiniment loin ; donc, on doit imposer des conditions supplémentaires, qui expriment le fait que notre condition initiale est connue sur un domaine fini. Ainsi l'on peut imposer que

$$\phi(u, z, 0) = \phi(u, 0) \delta(z) \quad (100) \quad \text{phiinitf}$$

où $\delta(z)$ est la fonction δ de Dirac, auquel cas l'intégrale sur z devient triviale ; ou que la dépendance sur z de la condition initiale ne s'étend que sur un domaine fini en pratique.

Si la dépendance de la condition initiale le long z est une localisation à $z = 0$, alors les coefficients sont donnés par les expressions

$$\begin{aligned} C_* &= \int_0^1 du \phi(u, 0, 0) \phi_*(u, 0, 0) \\ C_n &= \int_0^1 du \phi(u, 0, 0) \phi_n(u, 0, 0) \end{aligned} \quad (101) \quad \text{CstarCnn}$$

6.1 Quelques remarques sur les conditions initiales le long z

En fait, on a été un peu rapides sur le traitement des conditions initiales le long z . L'expression générale pour les fonctions propres peut être écrite en termes de fonctions réelles comme

$$\begin{aligned} \phi_*(u, z, \tau) &= f_*(u) [A_*^{(1)} \cos(k_* z - \omega \tau) + A_*^{(2)} \sin(k_* z - \omega \tau)] \\ \phi_n(u, z, \tau) &= f_n(u) [A_n^{(1)} \cos(k_n z - \omega \tau) + A_n^{(2)} \sin(k_n z - \omega \tau)] \end{aligned} \quad (102) \quad \text{phistarp}$$

Si l'on impose la condition initiale $\partial \phi(u, z, 0) / \partial \tau = 0$, alors on trouve les conditions $A_*^{(2)} = 0 = A_n^{(2)}$ et la solution prend la forme

$$\phi(u, z, \tau) = C_* f_*(u) \cos(k_* z - \omega_* \tau) + \sum_{n=1}^{\infty} C_n f_n(u) \cos(k_n z - \omega_n \tau) \quad (103) \quad \text{soldphid}$$

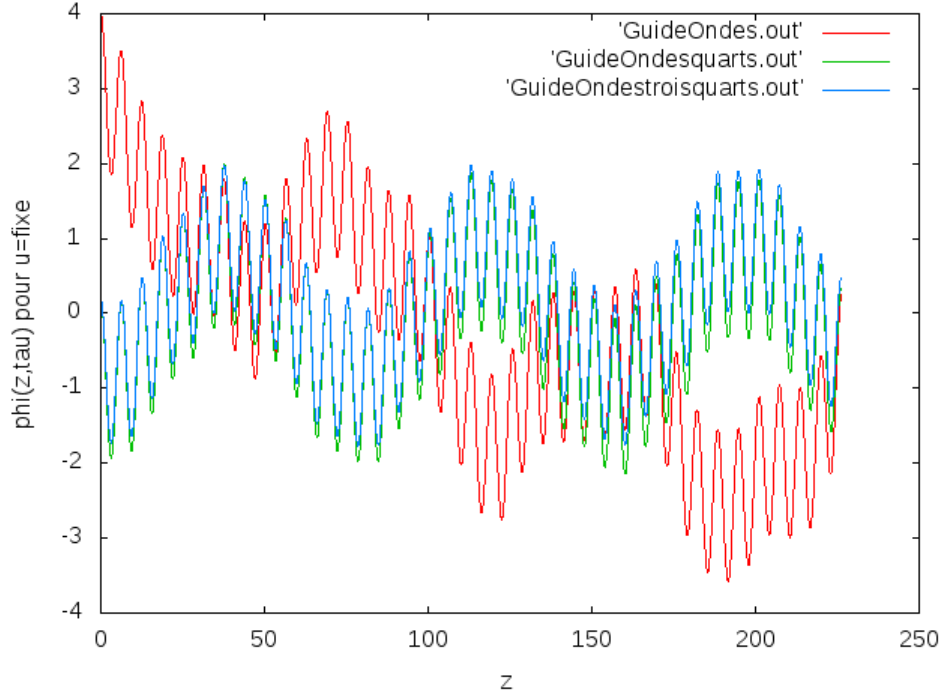


FIGURE 11 – L’amplitude de l’onde le long z pour $u = 0.25, 0.5$ et 0.75 , $\omega\tau = 2\pi$ et $\omega_n = 2.3\omega_{\text{coup}}$.

7 Quelques remarques sur le traitement informatique

L’expression (103) implique que l’on n’a pas besoin de connaître *toutes* les valeurs antérieures en τ où différentes de celles de u ou de z qui sont d’intérêt.

Ainsi, si l’on cherche à afficher le profil de l’onde, comme fonction de z , à u et τ fixes, il suffit d’entrer dans l’expression (103) les valeurs appropriées.

Dans la fig. 11 on affiche $\phi(u = 0.25, z, \tau = 2\pi/\omega_*)$, $\phi(u = 0.5, z, \tau = 2\pi/\omega_*)$, $\phi(u = 0.75, z, \tau = 2\pi/\omega_*)$, pour $\omega_n = 2.3\omega_{\text{coup}}$.

Exercice 17. Choisir comme condition initiale un mode propre et tracer les courbes correspondantes à la fig. 11.

Solution : Un mode propre consiste à une fonction $\phi(u, z, \tau)$, qui correspond à une onde stationnaire le long u — pour les conditions aux limites Dirichlet et une onde progressive le long z . Un tel mode sera, nécessairement, de la forme

$$\phi(u, z, \tau) = f(u) \sin(kz - \omega\tau) \quad (104)$$

où $f(u)$ est fonction propre de $-d^2/du^2$ et ω et k satisfont la relation de dispersion correspondante. Par conséquent on constate que ce sera un sinus pur en z , pour u et τ fixes—ce qui nous

permet d'interpréter les courbes de la fig. [B.4](#) à compléter.

Exercice 18. Facultatif : *Etudier plus systématiquement la propagation le long z en variant les paramètres.*

Solution :

8 Conclusions

A Description des codes informatiques

A.1 Les codes en C

Les courbes ont été obtenues en utilisant les codes, qui sont inclus dans l'annexe [B](#). Ici on présentera la procédure à suivre pour les utiliser.

Les codes sont écrits en langage C et doivent être compilés en réalisant la liaison avec la bibliothèque mathématique ; sous Unix ou Linux ceci veut dire qu'en ligne de commande on tape

```
gcc -O -o exe prog_tp0.c -lm
```

pour résoudre l'équation avec conditions aux limites de type Robin et de manière identique pour les conditions aux limites Dirichlet et Neumann.

L'exécution en ligne de commande est, alors, réalisée par la commande

```
./exe > exe_Robin.out
128
0.1
500.0
```

La première donnée, un entier, est le nombre de termes dans la série, où l'on développe la solution ; la deuxième donnée, un nombre réel, est la largeur de la Gaussienne, qui est la condition initiale que l'on emploie ; et la troisième donnée, aussi un nombre réel, est le paramètre Γ des conditions aux limites Robin.

On obtient, ainsi, un fichier de résultats, `exe_Robin.out`, qui comprend trois colonnes : τ , u et $[\phi(u, \tau)]_N$.

Pour le cas de la propagation en 2+1 dimensions—cf. le code, présenté en sous-section [B.4](#)—l'exécution est la suivante :

```
./exe > exe_Robin.out
128
0.1
5.0
1.0
```

La dernière donnée est la valeur de ω_* .

A.2 Les codes en Java

Le code en Java définit une applet, qui réalise l'animation. Elle s'appuie sur les données, définies dans le fichier `Example5Applet.html`, pour les valeurs des paramètres, comme le taux de rafraîchissement, `fps`, le délai entre mise à jour de l'affichage, `delay` et la taille du canevas, `d.width` et `d.height`.

On constate que, si l'on varie ces paramètres, on trouve que à compléter.

B Les codes sources en C

B.1 Le code en C pour les conditions aux limites Robin

```
//Condiitions aux bords de type Robin
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>

#define sqr(x) ( (x)*(x) )
//#define sigma 0.01
#define PI acos(-1.0)
//#define Gamma 5.0
#define Astar sqrt( 2.0/ ( Gamma*(1.0-exp( -2.0/Gamma ) ) ) )
#define L 1.0

double thetan;

double sigma, Gamma;

void simpson(double s[], int N){//set the coefficients of the numerical integration proc
    int i;
    s[0] = 1./3.;
    for(i=1;i<=N;i=i+2){
        s[i]=4./3.;
    }
    for(i=2;i<N;i=i+2){
        s[i]=2./3.;
    }
    s[N+1]=1./3.;
}
```

```

void trapezes(double s[], int N){// set the coefficients of the numerical integration pr
    int i;
    s[0]=0.5;
    for(i=1;i<(N+1);i++){
        s[i]=1.0;
    }
    s[N+1]=0.5;
}

double integrate(double f[], double s[], double A, double B, int N){//numerical integrat
    int i; double somme=s[0]*f[0];

    for(i=1;i<=N;i++){
        somme = somme + s[i]*f[i];
    }
    somme = somme + s[N+1]*f[N+1];
    return( ((B-A)/(N+1))*somme);
}

double evaluestar(){
    return( -1.0/sqr(Gamma) );
}

double eigenfneg(double x){
    return(Astar*exp(-( x/L)/Gamma )) );
}

double eigenfnegprime(double x){
    return(-(Astar/(L*Gamma))*exp(-x/(L*Gamma)) );
}

double eigenvalue(int n){// Valeurs propres de la seconde dérivée spatiale
    return(sqr(n*PI/L));} //choix d'unités: v=1 et L=1

double eigenfunction(int n, double x){// fonctions propres de la seconde dérivée spatiale

    thetan = 2.0*atan(Gamma*n*PI);
    return( sqrt(2.0/L)*sin( (n*PI*x/L)-0.5*thetan) );
}

```

```

        // return(sqrt(2.0/L)*sin(n*PI*x/L)); // eigenfunctions for Dirichlet boundary con
    }

double eigenfunctionprime(int n, double x){

    thetan = 2.0*atan(Gamma*n*PI);
    return( sqrt(2.0/L)*(n*PI/L)*cos( (n*PI*x/L) - 0.5*thetan ) );
}

double initialcondition(double x){// condition initiale
    return(exp(-sqr(x-0.5*L)/(2.0*sqr(sigma)))/sqrt(2.0*PI*sqr(sigma))); //une gauss
// return(1.0); // une constante
// return(eigenfneg(x)); //fonction propre de la valeur propre négative
// return(eigenfunction(1,x)); //testing an eigenfunction as initial condition
}

main(){
    double somme;
    int n, i, j;
    int N;

    double u,tau;

    scanf("%d", &N);

    scanf("%lf %lf", &sigma, &Gamma);

    double cstar, c[N+2];

    double s[N+2], f[N+2];
    // simpson(s,N); // set the coefficients of the numerical integration for Simpson
    trapezes(s,N); // set the coefficients of the numerical integration for trapezes

    // Exercice 0: Retrouver que la fonction propre de valeur propre négative est corr

```

```

for(i=0;i<=(N+1);i++){
    u = (float)i/(float)(N+1);
    //      f[i] = sqr(Astar)*exp(-2.0*u/Gamma);
    f[i] = sqr(eigenfneg(u));
}

//    printf("%17.14g\n",integrate(f,s,0.0,1.0,N));

// Exercice 1: Retrouver que les fonctions propres de valeur propre positive sont correctes

for(n=1;n<(N+1);n++){

for(i=0;i<=(N+1);i++){
    u = (float)i/(float)(N+1);
    f[i] =  sqr(eigenfunction(n,u));
}
//    printf("%d\t%17.14g\n",n,integrate(f,s,0.0,1.0,N));

}

//Exercice 2: Orthogonalité des fonctions propres
//(a) Orthogonalité entre la fstar et les fn

for(n=1;n<(N+1);n++){
    for(i=0;i<=(N+1);i++){
u = (float)i/(float)(N+1);
f[i] = eigenfneg(u)*eigenfunction(n,u);
    }

    //    printf("%d\t%17.14g\n",n,integrate(f,s,0.0,1.0,N));

}

//Exercice 3: Reproduire la condition initiale

//Calcul du coefficient cstar

```

```

for(i=0;i<=(N+1);i++){
    u = (float)i/(float)(N+1);
    f[i] = initialcondition(u)*eigenfneg(u);
}
cstar = integrate(f,s,0.0,1.0,N);

//Calculs des coefficients c[n]

for(n=1;n<(N+1);n++){
    for(i=0;i<=(N+1);i++){
u = (float)i/(float)(N+1);
f[i] = initialcondition(u)*eigenfunction(n,u);
    }
    c[n] = integrate(f,s,0.0,1.0,N);
}

//Test que la condition initiale est correctement reproduite

for(i=0;i<=(N+1);i++){
    u = (float)i/(float)(N+1);
    somme = cstar*eigenfneg(u);
    for(n=1;n<(N+1);n++){
somme = somme + c[n]*eigenfunction(n,u);
    }
    //          printf("%g\t%g\t%g\n",u,initialcondition(u),somme);
}

//Exercice 4: Evolution dans le temps, pour le cas spécial  $\Omega = 0$ 

double w[N+2];
for(n=1;n<(N+1);n++){
    w[n] = sqrt( sqrt(n*PI) + sqrt(1.0/Gamma) );
}

double T1 = 2.0*PI/w[1], TN = 2.0*PI/w[N];
/* Le courant de bord de l'énergie pour le cas où la condition initiale est
    une fonction propre, appartenant au spectre positif

*/

//Pour préciser un seul mode

```

```

//      scanf("%d", &n);

double Jbord, dphidtau0,dphidtau1,dphidu0,dphidu1;

for(tau=0.0; tau < 2*T1; tau = tau + TN){
    //Jbord pour le cas de la condition initiale, qui serait un seul mode, qu'appartene
    //      Jbord = (w[n]/Gamma)*sin(w[n]*tau)*( sqrt(eigenfunction(n,0.0))-sqrt(eigenfu

    //Le calcul de Jbord, dans le cas d'une condition initiale générale
    dphidtau0 = 0.0; dphidtau1 = 0.0;
    dphidu0 = cstar*eigenfneg(0.0); dphidu1 = cstar*eigenfneg(1.0);

    for(n=1;n<(N+1);n++){
dphidtau0 = dphidtau0 + c[n]*w[n]*sin(w[n]*tau)*eigenfunction(n,0.0);
dphidtau1 = dphidtau1 + c[n]*w[n]*sin(w[n]*tau)*eigenfunction(n,1.0);

dphidu0 = dphidu0 + c[n]*cos(w[n]*tau)*eigenfunction(n,0.0);
dphidu1 = dphidu1 + c[n]*cos(w[n]*tau)*eigenfunction(n,1.0);

    }

    dphidtau0 = -dphidtau0;
    dphidtau1 = -dphidtau1;

    dphidu0 = -dphidu0/Gamma;
    dphidu1 = -dphidu1/Gamma;

    Jbord = dphidu0*dphidtau0 - dphidu1*dphidtau1;

    //      printf("%g\t%g\n",tau, Jbord);

}

for(tau=0.0; tau <2*T1; tau = tau + 10.0*TN){

    /*
Calculer la valeur de la solution à chaque bord, c.à.d. phi(0.,tau) et
phi(1.,tau)
    */

```

```

//=====
/*
u = 0.0;
somme = cstar*eigenfneg(u);
for(n=1;n<(N+1);n++){
somme = somme + c[n]*cos(w[n]*tau)*eigenfunction(n,u);
}

    printf("%g\t%g\t",tau, somme);

u = 1.0;
somme = cstar*eigenfneg(u);
for(n=1;n<(N+1);n++){
somme = somme + c[n]*cos(w[n]*tau)*eigenfunction(n,u);
}
printf("%g\n",somme);
*/
//=====
    for(i=0;i<=(N+1);i++){
u = (float)i/(float)(N+1);
somme = cstar*eigenfneg(u);

for(n=1;n<(N+1);n++){
    somme = somme + c[n]*cos(w[n]*tau)*eigenfunction(n,u);
}
printf("%g\t%g\t%g\n",tau, u, somme);
    }

}

}

```

B.2 Le code en C pour les conditions aux limites Dirichlet

codeDir

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>

#define sqr(x) ( (x)*(x) )

```

```

//#define sigma 0.01
#define PI acos(-1.0)
//#define Gamma 5.0
#define Astar sqrt( 2.0/ ( Gamma*(1.0-exp( -2.0/Gamma ) ) ) )
#define L 1.0

double  thetan;

double sigma, Gamma, beta; // beta < = -1/Gamma^2

void simpson(double s[], int N){//set the coefficients of the numerical integration process
    int i;
    s[0] = 1./3.;
    for(i=1;i<=N;i=i+2){
        s[i]=4./3.;
    }
    for(i=2;i<N;i=i+2){
        s[i]=2./3.;
    }
    s[N+1]=1./3.;
}

void trapezes(double s[], int N){// set the coefficients of the numerical integration process
    int i;
    s[0]=0.5;
    for(i=1;i<(N+1);i++){
        s[i]=1.0;
    }
    s[N+1]=0.5;
}

double integrate(double f[], double s[], double A, double B, int N){//numerical integration
    int i; double somme=s[0]*f[0];

    for(i=1;i<=N;i++){
        somme = somme + s[i]*f[i];
    }
    somme = somme + s[N+1]*f[N+1];
    return( ((B-A)/(N+1))*somme);
}

```

```

double evaluestar(){
    return( -1.0/sqr(Gamma) );
}

double eigenfneg(double x){
    return(Astar*exp(-( x/L)/Gamma )) );
}

double eigenfnegprime(double x){
    return(-(Astar/(L*Gamma))*exp(-x/(L*Gamma)) );
}

double eigenvalue(int n){// Valeurs propres de la seconde dérivée spatiale
    return(sqr(n*PI/L));} //choix d'unités: v=1 et L=1

double eigenfunction(int n, double x){// fonctions propres de la seconde dérivée spatiale

    thetan = 2.0*atan(Gamma*n*PI);
    return( sqrt(2.0/L)*sin( (n*PI*x/L)-0.5*thetan) );

    // return(sqrt(2.0/L)*sin(n*PI*x/L)); // eigenfunctions for Dirichlet boundary conditions
}

double eigenfunctionprime(int n, double x){

    thetan = 2.0*atan(Gamma*n*PI);
    return( sqrt(2.0/L)*(n*PI/L)*cos( (n*PI*x/L) - 0.5*thetan ) );
}

double initialcondition(double x){// condition initiale
    return(exp(-sqr(x-0.5*L)/(2.0*sqr(sigma)))/sqr(2.0*PI*sqr(sigma))); //une gaussienne
// return(1.0); // une constante
// return(eigenfneg(x)); //fonction propre de la valeur propre négative
// return(eigenfunction(1,x)); //testing an eigenfunction as initial condition
}

```

```

main(){
    double somme;
    int n, i, j;
    int N;

    double u,tau;

    scanf("%d", &N);

    //    scanf("%lf %lf", &sigma, &Gamma);
    scanf("%lf", &sigma); Gamma = 0.0;
    scanf("%lf", &beta);

    double cstar, c[N+2];

    double s[N+2], f[N+2];
    //    simpson(s,N); // set the coefficients of the numerical integration for Simpson
    trapezes(s,N); // set the coefficients of the numerical integration for trapezes

    // Exercice 0: Retrouver que la fonction propre de valeur propre négative est correcte

    for(i=0;i<=(N+1);i++){
        u = (float)i/(float)(N+1);
        //        f[i] = sqr(Astar)*exp(-2.0*u/Gamma);
        //        f[i] = sqr(eigenfneg(u));
    }

    //    printf("%17.14g\n",integrate(f,s,0.0,1.0,N));

    // Exercice 1: Retrouver que les fonctions propres de valeur propre positive sont correctes

    for(n=1;n<(N+1);n++){

        for(i=0;i<=(N+1);i++){
            u = (float)i/(float)(N+1);

```

```

    f[i] =  sqr(eigenfunction(n,u));
}
//    printf("%d\t%17.14g\n",n,integrate(f,s,0.0,1.0,N));

}

//Exercice 2: Orthogonalité des fonctions propres
//(a) Orthogonalité entre la fstar et les fn

for(n=1;n<(N+1);n++){
    for(i=0;i<=(N+1);i++){
u = (float)i/(float)(N+1);
// f[i] = eigenfneg(u)*eigenfunction(n,u);
    }

    //    printf("%d\t%17.14g\n",n,integrate(f,s,0.0,1.0,N));

}

//Exercice 3: Reproduire la condition initiale

//Calcul du coefficient cstar

for(i=0;i<=(N+1);i++){
    u = (float)i/(float)(N+1);
    //    f[i] = initialcondition(u)*eigenfneg(u);
}
//    cstar = integrate(f,s,0.0,1.0,N);

//Calculs des coefficients c[n]

for(n=1;n<(N+1);n++){
    for(i=0;i<=(N+1);i++){
u = (float)i/(float)(N+1);
f[i] = initialcondition(u)*eigenfunction(n,u);
    }
    c[n] = integrate(f,s,0.0,1.0,N);
}

//Test que la condition initiale est correctement reproduite

```

```

for(i=0;i<=(N+1);i++){
    u = (float)i/(float)(N+1);
    //      somme = cstar*eigenfneg(u);
    somme = 0.0;
    for(n=1;n<(N+1);n++){
somme = somme + c[n]*eigenfunction(n,u);
    }
    //      printf("%g\t%g\t%g\n",u,initialcondition(u),somme);
}

//Exercice 4: Evolution dans le temps, pour le cas spécial  $\Omega = 0$ 

double w[N+2];
for(n=1;n<(N+1);n++){
    //      w[n] = sqrt( sqr(n*PI) + sqr(1.0/Gamma) );
    //      w[n] = n*PI;
    w[n] = sqrt( sqr(n*PI) - beta ); // beta < 0
}

double T1 = 2.0*PI/w[1], TN = 2.0*PI/w[N];
//      printf("%g\t%g\n",T1, TN);
/* Le courant de bord de l'énergie pour le cas où la condition initiale est
    une fonction propre, appartenant au spectre positif

*/

//Pour préciser un seul mode
//      scanf("%d", &n);

double Jbord, dphidtau0,dphidtau1,dphidu0,dphidu1;

for(tau=0.0; tau < 2*T1; tau = tau + 10*TN){
    //Jbord pour le cas de la condition initiale, qui serait un seul mode, qu'apparte
    //      Jbord = (w[n]/Gamma)*sin(w[n]*tau)*( sqr(eigenfunction(n,0.0))-sqr(eigenfu

    //Le calcul de Jbord, dans le cas d'une condition initiale générale
    dphidtau0 = 0.0; dphidtau1 = 0.0;
    dphidu0 = cstar*eigenfneg(0.0); dphidu1 = cstar*eigenfneg(1.0);

    for(n=1;n<(N+1);n++){

```

```

dphidtau0 = dphidtau0 + c[n]*w[n]*sin(w[n]*tau)*eigenfunction(n,0.0);
dphidtau1 = dphidtau1 + c[n]*w[n]*sin(w[n]*tau)*eigenfunction(n,1.0);

dphidu0 = dphidu0 + c[n]*cos(w[n]*tau)*eigenfunction(n,0.0);
dphidu1 = dphidu1 + c[n]*cos(w[n]*tau)*eigenfunction(n,1.0);

    }

    dphidtau0 = -dphidtau0;
    dphidtau1 = -dphidtau1;

    dphidu0 = -dphidu0/Gamma;
    dphidu1 = -dphidu1/Gamma;

    Jbord = dphidu0*dphidtau0 - dphidu1*dphidtau1;

    //          printf("%g\t%g\n",tau, Jbord);

}

for(tau=0.0; tau <2*T1; tau = tau + 0.01*TN){

    /*
    Calculer la valeur de la solution à chaque bord, c.à.d. phi(0.,tau) et
    phi(1.,tau)
    */
    //=====
    //=====
        for(i=0;i<=(N+1);i++){
    u = (float)i/(float)(N+1);
    // somme = cstar*eigenfneg(u);
    somme = 0.0;
    for(n=1;n<(N+1);n++){
        somme = somme + c[n]*cos(w[n]*tau)*eigenfunction(n,u);
    }
    printf("%g\t%g\t%g\n",tau, u, somme);
    }
}

```

```

    }

}

```

B.3 Le code en C pour les conditions aux limites Neumann

//Code pour le cas de conditions aux bords Neumann

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>

#define sqr(x) ( (x)*(x) )
//#define sigma 0.01
#define PI acos(-1.0)
//#define Gamma 5.0
#define Astar 1.0
#define L 1.0

double thetan;

double sigma, Gamma, beta; // beta < = -1/Gamma^2

void simpson(double s[], int N){//set the coefficients of the numerical integration procedure
    int i;
    s[0] = 1./3.;
    for(i=1;i<=N;i=i+2){
        s[i]=4./3.;
    }
    for(i=2;i<N;i=i+2){
        s[i]=2./3.;
    }
    s[N+1]=1./3.;
}

void trapezes(double s[], int N){// set the coefficients of the numerical integration procedure
    int i;
    s[0]=0.5;

```

```

    for(i=1;i<(N+1);i++){
        s[i]=1.0;
    }
    s[N+1]=0.5;
}

double integrate(double f[], double s[], double A, double B, int N){//numerical integration
    int i; double somme=s[0]*f[0];

    for(i=1;i<=N;i++){
        somme = somme + s[i]*f[i];
    }
    somme = somme + s[N+1]*f[N+1];
    return( ((B-A)/(N+1))*somme);
}

double evaluestar(){
    return( 0.0 );
}

double eigenfzero(double x){
    return(1.0);
}

double eigenfnegprime(double x){
    return(-(Astar/(L*Gamma))*exp(-x/(L*Gamma)) );
}

double eigenvalue(int n){// Valeurs propres de la seconde dérivée spatiale
    return(sqrt(n*PI/L));} //choix d'unités: v=1 et L=1

double eigenfunction(int n, double x){// fonctions propres de la seconde dérivée spatiale

    //  thetan = 2.0*atan(Gamma*n*PI);
    return( sqrt(2.0/L)*cos( (n*PI*x/L) ) );
}

double eigenfunctionprime(int n, double x){

    thetan = 2.0*atan(Gamma*n*PI);

```

```

    return( sqrt(2.0/L)*(n*PI/L)*cos( (n*PI*x/L) - 0.5*thetan ) );
}

double initialcondition(double x){// condition initiale
    return(exp(-sqr(x-0.5*L)/(2.0*sqr(sigma)))/sqrt(2.0*PI*sqr(sigma))); //une gauss
// return(1.0); // une constante
// return(eigenfneg(x));//fonction propre de la valeur propre négative
// return(eigenfunction(1,x)); //testing an eigenfunction as initial condition
}

main(){
    double somme;
    int n, i, j;
    int N;

    double u,tau;

    scanf("%d", &N);

    //    scanf("%lf %lf", &sigma, &Gamma);
    scanf("%lf", &sigma);
    scanf("%lf", &beta); //beta doit être négatif!
    Gamma = sqrt(-1.0/beta);

    double cstar, c[N+2];

    double s[N+2], f[N+2];
    //    simpson(s,N); // set the coefficients of the numerical integration for Simpson
    trapezes(s,N); // set the coefficients of the numerical integration for trapezes

    // Exercice 0: Retrouver que la fonction propre de valeur propre zéro est correcte

    for(i=0;i<=(N+1);i++){
        u = (float)i/(float)(N+1);
        f[i] = sqr(eigenfzero(u));
    }
}

```

```

    }

    //      printf("%17.14g\n",integrate(f,s,0.0,1.0,N));

// Exercice 1: Retrouver que les fonctions propres de valeur propre positive sont correctes

    for(n=1;n<(N+1);n++){

        for(i=0;i<=(N+1);i++){
            u = (float)i/(float)(N+1);
            f[i] =  sqr(eigenfunction(n,u));
        }
        //      printf("%d\t%17.14g\n",n,integrate(f,s,0.0,1.0,N));

    }

//Exercice 2: Orthogonalité des fonctions propres
// (a) Orthogonalité entre la fstar et les fn

    for(n=1;n<(N+1);n++){
        for(i=0;i<=(N+1);i++){
u = (float)i/(float)(N+1);
f[i] = eigenfzero(u)*eigenfunction(n,u);
        }

        //      printf("%d\t%17.14g\n",n,integrate(f,s,0.0,1.0,N));

    }

//Exercice 3: Reproduire la condition initiale

//Calcul du coefficient cstar

    for(i=0;i<=(N+1);i++){
        u = (float)i/(float)(N+1);
        f[i] = initialcondition(u)*eigenfzero(u);
    }
    cstar = integrate(f,s,0.0,1.0,N);

```

```

//Calculs des coefficients c[n]

for(n=1;n<(N+1);n++){
    for(i=0;i<=(N+1);i++){
u = (float)i/(float)(N+1);
f[i] = initialcondition(u)*eigenfunction(n,u);
    }
    c[n] = integrate(f,s,0.0,1.0,N);
}

//Test que la condition initiale est correctement reproduite

for(i=0;i<=(N+1);i++){
    u = (float)i/(float)(N+1);
    somme = cstar*eigenfzero(u);
    for(n=1;n<(N+1);n++){
somme = somme + c[n]*eigenfunction(n,u);
    }
    //      printf("%g\t%g\t%g\n",u,initialcondition(u),somme);
}

//Exercice 4: Evolution dans le temps, pour le cas spécial  $\Omega = 0$ 

double w[N+2];
for(n=1;n<(N+1);n++){
    //      w[n] = sqrt( sqr(n*PI) + sqr(1.0/Gamma) );
    //      w[n] = n*PI;
    w[n] = sqrt( sqr(n*PI) - beta ); // beta < 0
}

double T1 = 2.0*PI/w[1], TN = 2.0*PI/w[N];
//      printf("%g\t%g\n",T1, TN);
/* Le courant de bord de l'énergie pour le cas où la condition initiale est
    une fonction propre, appartenant au spectre positif

*/

//Pour préciser un seul mode
//      scanf("%d", &n);

```

```

double Jbord, dphidtau0,dphidtau1,dphidu0,dphidu1;

for(tau=0.0; tau < 2*T1; tau = tau + 10*TN){
    //Jbord pour le cas de la condition initiale, qui serait un seul mode, qu'appartient
    //      Jbord = (w[n]/Gamma)*sin(w[n]*tau)*( sqrt(eigenfunction(n,0.0))-sqrt(eigenfunction(n,1.0)));

    //Le calcul de Jbord, dans le cas d'une condition initiale générale
    dphidtau0 = 0.0; dphidtau1 = 0.0;
    dphidu0 = cstar*eigenfzero(0.0); dphidu1 = cstar*eigenfzero(1.0);

    for(n=1;n<(N+1);n++){
        dphidtau0 = dphidtau0 + c[n]*w[n]*sin(w[n]*tau)*eigenfunction(n,0.0);
        dphidtau1 = dphidtau1 + c[n]*w[n]*sin(w[n]*tau)*eigenfunction(n,1.0);

        dphidu0 = dphidu0 + c[n]*cos(w[n]*tau)*eigenfunction(n,0.0);
        dphidu1 = dphidu1 + c[n]*cos(w[n]*tau)*eigenfunction(n,1.0);

    }

    dphidtau0 = -dphidtau0;
    dphidtau1 = -dphidtau1;

    dphidu0 = -dphidu0/Gamma;
    dphidu1 = -dphidu1/Gamma;

    Jbord = dphidu0*dphidtau0 - dphidu1*dphidtau1;

    //      printf("%g\t%g\n",tau, Jbord);

}

for(tau=0.0; tau <2*T1; tau = tau + 10.0*TN){

    /*
    Calculer la solution
    */
    //=====
    //=====
    for(i=0;i<=(N+1);i++){

```

```

u = (float)i/(float)(N+1);
somme = cstar*eigenfzero(u)*cos(tau/Gamma);
for(n=1;n<(N+1);n++){
    somme = somme + c[n]*cos(w[n]*tau)*eigenfunction(n,u);
}
printf("%g\t%g\t%g\n",tau, u, somme);
    }

}

}

```

B.4 Le code en C pour la solution de l'équation d'ondes en 2+1 dimensions

code2plus1

```

//Condiitions aux bords de type Robin: Généralization pour le cas de deux dimensions spat
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>

#define sqr(x) ( (x)*(x) )
//#define sigma 0.01
#define PI acos(-1.0)
//#define Gamma 5.0
#define Astar sqrt( 2.0/ ( Gamma*(1.0-exp( -2.0/Gamma ) ) ) )
#define L 1.0

double thetan;

double sigma, Gamma;

void simpson(double s[], int N){//set the coefficients of the numerical integration proc
    int i;
    s[0] = 1./3.;
    for(i=1;i<=N;i=i+2){
        s[i]=4./3.;
    }
}

```

```

    for(i=2;i<N;i=i+2){
        s[i]=2./3.;
    }
    s[N+1]=1./3.;
}

void trapezes(double s[], int N){// set the coefficients of the numerical integration pr
    int i;
    s[0]=0.5;
    for(i=1;i<(N+1);i++){
        s[i]=1.0;
    }
    s[N+1]=0.5;
}

double integrate(double f[], double s[], double A, double B, int N){//numerical integrat
    int i; double somme=s[0]*f[0];

    for(i=1;i<=N;i++){
        somme = somme + s[i]*f[i];
    }
    somme = somme + s[N+1]*f[N+1];
    return( ((B-A)/(N+1))*somme);
}

double evaluestar(){
    return( -1.0/sqr(Gamma) );
}

double eigenfneg(double x){
    return(Astar*exp(-( x/L)/Gamma )) );
}

double eigenfnegprime(double x){
    return(-(Astar/(L*Gamma))*exp(-x/(L*Gamma)) );
}

double eigenvalue(int n){// Valeurs propres de la seconde dérivée spatiale
    return(sqr(n*PI/L));} //choix d'unités: v=1 et L=1

```

```

double eigenfunction(int n, double x){// fonctions propres de la seconde dérivée spatiale

    thetan = 2.0*atan(Gamma*n*PI);
    return( sqrt(2.0/L)*sin( (n*PI*x/L)-0.5*thetan) );

    // return(sqrt(2.0/L)*sin(n*PI*x/L)); // eigenfunctions for Dirichlet boundary conditions
}

double eigenfunctionprime(int n, double x){

    thetan = 2.0*atan(Gamma*n*PI);
    return( sqrt(2.0/L)*(n*PI/L)*cos( (n*PI*x/L) - 0.5*thetan ) );
}

double initialcondition(double x){// condition initiale
    return(exp(-sqr(x-0.5*L)/(2.0*sqr(sigma)))/sqrt(2.0*PI*sqr(sigma))); //une gaussienne
    // return(1.0); // une constante
    // return(eigenfneg(x)); //fonction propre de la valeur propre négative
    // return(eigenfunction(1,x)); //testing an eigenfunction as initial condition
}

main(){
    double somme;
    int n, i, j;
    int N;

    double u,tau;

    scanf("%d", &N);

    scanf("%lf %lf", &sigma, &Gamma);

    double cstar, c[N+2];

```

```

double s[N+2], f[N+2];
//      simpson(s,N); // set the coefficients of the numerical integration for Simpson
      trapezes(s,N); // set the coefficients of the numerical integration for trapezes

// Exercice 0: Retrouver que la fonction propre de valeur propre négative est correcte

for(i=0;i<=(N+1);i++){
    u = (float)i/(float)(N+1);
    //      f[i] = sqr(Astar)*exp(-2.0*u/Gamma);
    f[i] = sqr(eigenfneg(u));
}

//      printf("%17.14g\n",integrate(f,s,0.0,1.0,N));

// Exercice 1: Retrouver que les fonctions propres de valeur propre positive sont correctes

for(n=1;n<(N+1);n++){

for(i=0;i<=(N+1);i++){
    u = (float)i/(float)(N+1);
    f[i] =  sqr(eigenfunction(n,u));
}
//      printf("%d\t%17.14g\n",n,integrate(f,s,0.0,1.0,N));

}

//Exercice 2: Orthogonalité des fonctions propres
//(a) Orthogonalité entre la fstar et les fn

for(n=1;n<(N+1);n++){
    for(i=0;i<=(N+1);i++){
u = (float)i/(float)(N+1);
f[i] = eigenfneg(u)*eigenfunction(n,u);
    }

//      printf("%d\t%17.14g\n",n,integrate(f,s,0.0,1.0,N));

```

```

}

//Exercice 3: Reproduire la condition initiale

//Calcul du coefficient cstar

for(i=0;i<=(N+1);i++){
    u = (float)i/(float)(N+1);
    f[i] = initialcondition(u)*eigenfneg(u);
}
cstar = integrate(f,s,0.0,1.0,N);

//Calculs des coefficients c[n]

for(n=1;n<(N+1);n++){
    for(i=0;i<=(N+1);i++){
u = (float)i/(float)(N+1);
f[i] = initialcondition(u)*eigenfunction(n,u);
    }
    c[n] = integrate(f,s,0.0,1.0,N);
}

//Test que la condition initiale est correctement reproduite

for(i=0;i<=(N+1);i++){
    u = (float)i/(float)(N+1);
    somme = cstar*eigenfneg(u);
    for(n=1;n<(N+1);n++){
somme = somme + c[n]*eigenfunction(n,u);
    }
    //          printf("%g\t%g\t%g\n",u,initialcondition(u),somme);
}

//Exercice 4: Evolution dans le temps, pour le cas spécial  $\beta = -1/\Gamma^2$ 

double w[N+2], kstar, lambdastar, wstar, veck[N+2], lambda[N+2], wcoup, wcoupn2,z;
wcoup = sqrt( sqr(N*PI) + (1.0/sqr(Gamma)) );

scanf("%lf", &wstar); // On fait lire de l'entrée standard la fréquence wstar

```

```

kstar=wstar; lambdastar = 2.0*PI/kstar;

for(n=1;n<(N+1);n++){
    w[n] = 2.3*wcoup; //on fixe la valeur des fréquences au-delà du seuil de coupure

    wcoupn2 = sqr(n*PI)+(1.0/sqr(Gamma)); //La fréquence de coupure pour le mode n
    veck[n] = sqrt( sqr(w[n]) - wcoupn2 ); //Le vecteur d'onde du mode n
    lambda[n] = 2.0*PI/veck[n]; //La longueur d'onde du mode n
    //      printf("%d\t%g\t%g\n",n,veck[n],lambda[n]);
}

tau = 10.0*(2.0*PI/wstar); // On donne l'instant en unités de la période du mode st

double T1 = 2.0*PI/w[1], TN = 2.0*PI/w[N];
/* Le courant de bord de l'énergie pour le cas où la condition initiale est
    une fonction propre, appartenant au spectre positif

*/

//Pour préciser un seul mode
//      scanf("%d", &n);

double Jbord, dphidtau0,dphidtau1,dphidu0,dphidu1;

for(tau=0.0; tau < 2*T1; tau = tau + TN){
    //Jbord pour le cas de la condition initiale, qui serait un seul mode, qu'apparte
    //      Jbord = (w[n]/Gamma)*sin(w[n]*tau)*( sqr(eigenfunction(n,0.0))-sqr(eigenfu

    //Le calcul de Jbord, dans le cas d'une condition initiale générale
    dphidtau0 = 0.0; dphidtau1 = 0.0;
    dphidu0 = cstar*eigenfneg(0.0); dphidu1 = cstar*eigenfneg(1.0);

    for(n=1;n<(N+1);n++){
dphidtau0 = dphidtau0 + c[n]*w[n]*sin(w[n]*tau)*eigenfunction(n,0.0);
dphidtau1 = dphidtau1 + c[n]*w[n]*sin(w[n]*tau)*eigenfunction(n,1.0);

dphidu0 = dphidu0 + c[n]*cos(w[n]*tau)*eigenfunction(n,0.0);
dphidu1 = dphidu1 + c[n]*cos(w[n]*tau)*eigenfunction(n,1.0);

    }

```

```

dphidtau0 = -dphidtau0;
dphidtau1 = -dphidtau1;

dphidu0 = -dphidu0/Gamma;
dphidu1 = -dphidu1/Gamma;

Jbord = dphidu0*dphidtau0 - dphidu1*dphidtau1;

//          printf("%g\t%g\n",tau, Jbord);

}

for(z=0.0; z <30000*lambda[N]; z = z + 10.0*lambda[1]){

    /*
    Calculer la valeur de la solution à chaque bord, c.à.d. phi(0.,tau) et
    phi(1.,tau)
    */
    //=====
    /*
    u = 0.0;
    somme = cstar*eigenfneg(u);
    for(n=1;n<(N+1);n++){
    somme = somme + c[n]*cos(w[n]*tau)*eigenfunction(n,u);
    }
        printf("%g\t%g\t",tau, somme);

    u = 1.0;
    somme = cstar*eigenfneg(u);
    for(n=1;n<(N+1);n++){
    somme = somme + c[n]*cos(w[n]*tau)*eigenfunction(n,u);
    }
    printf("%g\n",somme);
    */
    //=====
    //          for(i=0;i<=(N+1);i++){
    //          u = (float)i/(float)(N+1);
    u = 0.75;
    somme = cstar*cos(kstar*z-wstar*tau)*eigenfneg(u);

```

```

        for(n=1;n<(N+1);n++){
somme = somme + c[n]*cos(veck[n]*z-w[n]*tau)*eigenfunction(n,u);
        }
        //          printf("%g\t%g\t%g\n",z, u, somme);
        printf("%g\t%g\n",z, somme);
        //      }

    }

}

```

C Les codes en Java

codeJava

Le fichier Example5Applet.java :

```

import java.awt.*;

/**
 * Example5Applet
 *
 * This is a template applet for animation.
 * It illustrates how to use a graphics context
 * to draw a sine wave to the screen with little
 * or no flashing.
 *
 * @author Arthur van Hoff
 */
public
class Example5Applet extends java.applet.Applet implements Runnable {
    int frame;
    int delay;
    Thread animator;

    /**
     * Initialize the applet and compute the delay between frames.
     */
    public void init() {
String str = getParameter("fps");

```

```

int fps = (str != null) ? Integer.parseInt(str) : 10;
delay = (fps > 0) ? (1000 / fps) : 100;
}

/**
 * This method is called when the applet becomes visible on
 * the screen. Create a thread and start it.
 */
public void start() {
animator = new Thread(this);
animator.start();
}

/**
 * This method is called by the thread that was created in
 * the start method. It does the main animation.
 */
public void run() {
// Remember the starting time
long tm = System.currentTimeMillis();
while (Thread.currentThread() == animator) {
    // Display the next frame of animation.
    repaint();

    // Delay depending on how far we are behind.
    try {
tm += delay;
Thread.sleep(Math.max(0, tm - System.currentTimeMillis()));
    } catch (InterruptedException e) {
break;
    }

    // Advance the frame
    frame++;
}

}

/**
 * This method is called when the applet is no longer
 * visible. Set the animator variable to null so that the
 * thread will exit before displaying the next frame.

```

```

        */
        public void stop() {
animator = null;
        }

        /**
         * Paint a frame of animation.
         */
        public void paint(Graphics g) {
update(g);
        }

        /**
         * Paint a frame of animation.
         */

        public double sqr(double x){
return((x)*(x));
        }
        public void update(Graphics g) {
Color bg = getBackground();
Dimension d = getSize();
int h = d.height / 2;
for (int x = 0 ; x < d.width ; x++) {
    int v = 10;

    int phase1 = (x-v*frame)%d.width + d.width/10;
    int phase2 = (x+v*frame)%d.width - d.width/10;
    double sigma1 = 70.0;
    double ph1 = sqr(phase1)/(double)(sigma1);
    double ph2 = sqr(phase2)/(double)(sigma1);
    int y1 = (int)(10*d.height*Math.exp(-ph1)/Math.sqrt((double)sigma1));
    int y2 = (int)(10*d.height*Math.exp(-ph2)/Math.sqrt((double)sigma1));

    int y = (y1 + y2)/2;

    g.setColor(bg);
    /*      g.drawLine(x,0,x,d.height-y1);
    g.drawLine(x,0,x,d.height-y2);
    g.setColor(Color.black);
    g.drawLine(x,d.height-y1,x,d.height);

```

```

        g.setColor(Color.red);
        g.drawLine(x,d.height-y2,x,d.height);
        */
        g.drawLine(x,0,x,d.height-y);
        g.setColor(Color.black);
        g.drawLine(x,d.height-y,x,d.height);

    }
    }
}

```

Le fichier Example5Applet.html :

```

<applet code=Example5Applet.class fps = 5 delay = 100 width = 1000 height = 400>
</applet>

```