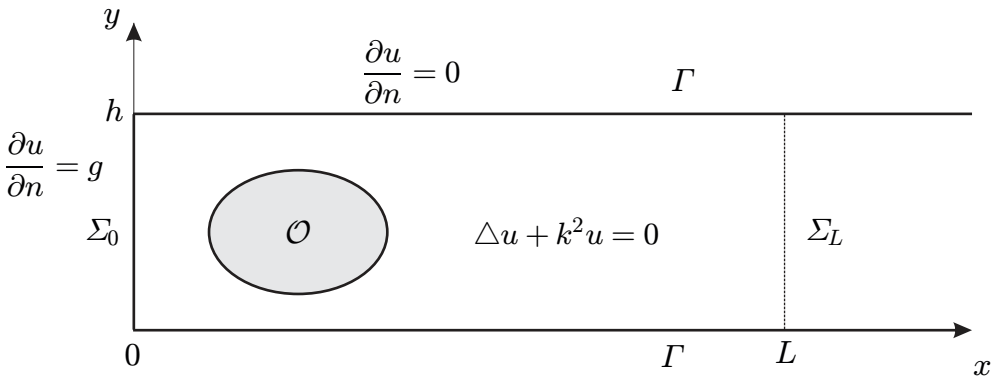
# Propagation dans un guide d'ondes acoustiques

E. Lunéville

2024

L'objet de ce TP est de tester différentes techniques de résolution numérique de la propagation d'ondes en régime harmonique dans un guide fermé. Par souci de simplicité, on s'intéresse au cas d'un guide 2D accoustique semi-infini localement perturbé (pertubation située dans le domaine $]0, L[\times[0, h]$).




Plus précisément, on cherche à résoudre le problème harmonique suivant :

$$\boxed{\begin{array}{ll} \triangle u + k^2 u = 0 & \text{sur } \Omega = \mathbb{R}_*^+ \times ]0, h[ \backslash \bar{\mathcal{O}} \\ \dfrac{\partial u}{\partial n} = 0 & \text{sur } \Gamma \cup \partial\mathcal{O} \\ \dfrac{\partial u}{\partial n} = g & \text{sur } \Sigma_0 \\ u(x,y) = \displaystyle\sum_{n\geq 0} \alpha_n \, e^{i\beta_n x} \varphi_n(y) & \forall x > L \quad \textit{(condition de rayonnement)} \end{array}} \tag{1}$$

où

$$\beta_n = \sqrt{k^2 - \left(\frac{n\pi}{h}\right)^2} \text{ avec } \mathcal{I}m\,\beta_n \geq 0 \text{ et } \mathcal{R}e\,\beta_n \geq 0$$
$$\varphi_n(y) = a_n \cos(\frac{n\pi y}{h}) \text{ avec } a_n = \sqrt{\frac{2}{h}} \text{ si } n > 0 \text{ et } a_0 = \sqrt{\frac{1}{h}}.$$

On rappelle qu'à l'extérieur de la zone perturbée, la solution se représente à l'aide d'une série ($\Sigma_a$ une section quelconque du guide avec $a > L$):

$$u(x,y) = \sum_{n\geq 0} \left( \int_{\Sigma_a} u\varphi_n \right) e^{i\beta_n (x-a)} \varphi_n(y) \quad \forall x \geq a, \forall y \in ]0, h[. \tag{2}$$

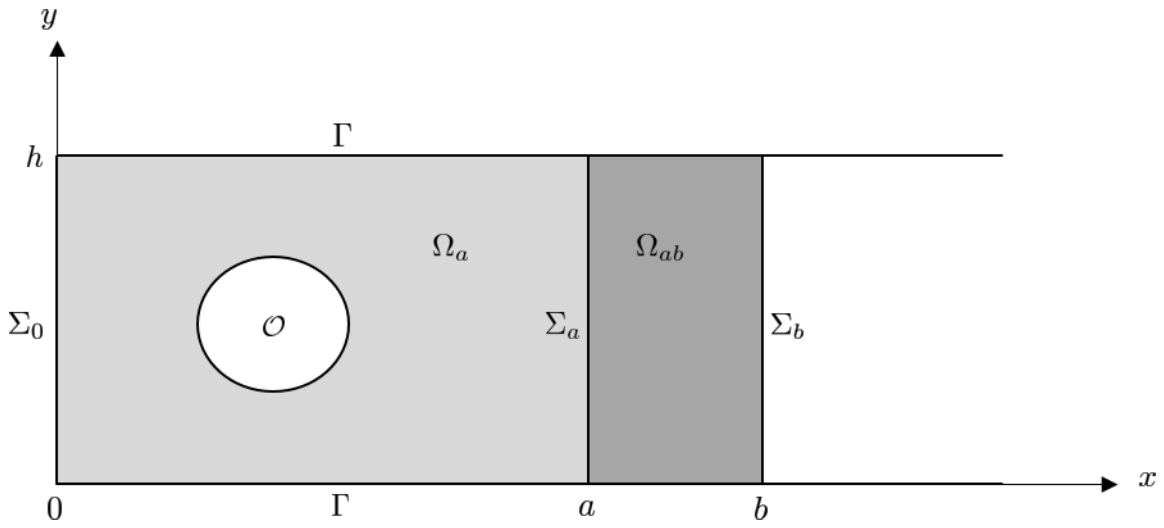
On va successivement s'intéresser aux méthodes suivantes :

- l'approximation basse fréquence
- la méthode PML
- la méthode DtN
- la méthode DtN épais

couplées à une approximation par éléments finis.

# 1 Réduction à un domaine borné

Dans toute la suite on se référera à la configuration géométrique suivante :




## 1.1 Approximation basse fréquence

Lorsque $k < \frac{\pi}{h}$ il n'y a qu'un seul mode propagatif ($n = 0$, $\beta_0 = k$). Tous les autres modes décroissent exponentiellement. Si on se situe loin de la perturbation, la solution sera quasiment de la forme $a\, e^{i\,k\,x}$ et vérifiera donc :

$$\frac{\partial u}{\partial n} \approx i\,k\,u \text{ sur } \Sigma_a \ (a > L).$$

Cette observation conduit à introduire le problème approché suivant dans $\Omega_a = \Omega \cap ]0, a[\times]0, h[$ :

$$\begin{cases} \triangle u + k^2 u = 0 & \text{sur } \Omega_a \\ \dfrac{\partial u}{\partial n} = 0 & \text{sur } \Gamma \cup \partial\mathcal{O} \\ \dfrac{\partial u}{\partial n} = g & \text{sur } \Sigma_0 \\ \dfrac{\partial u}{\partial n} = i\,k\,u & \text{sur } \Sigma_a \ (a > L). \end{cases} \tag{3}$$

dont la formulation variationnelle est : trouver $u \in H^1(\Omega_a)$ telle que $\forall v \in H^1(\Omega_a)$

$$\int_{\Omega_a} \nabla u \nabla \bar{v} - k^2 \int_{\Omega_a} u\,\bar{v} - ik \int_{\Sigma_a} u\,\bar{v} = \int_{\Sigma_0} g\,\bar{v}.$$

## 1.2 Méthode PML

La méthode PML (Pefectly Match Layer) consiste à modifier de façon astucieuse les propriétés du milieu de telle sorte que tous les modes deviennent évanescents, permettant ainsi de créer une couche absorbante. L'astuce des PML réside dans le fait qu'elle ne crée pas de réflexion à l'interface des couches. En pratique, on modifie l'opérateur de Helmholtz de la façon suivante dans la couche PML :

$$\frac{\partial^2 u}{\partial y^2} + \alpha \frac{\partial}{\partial x}\left(\alpha \frac{\partial u}{\partial x}\right) + k^2 u = 0$$

où $\alpha$ est une fonction qui ne dépend que de $x$. Une condition suffisante pour que la couche soit absorbante est

$$\mathcal{R}e\,\alpha > 0 \text{ et } \mathcal{I}m\,\alpha < 0.$$

En introduisant la fonction :

$$\tilde{\alpha}(x) = \begin{cases} 1 & \text{si } x < a \\ \alpha(x) & \text{si } a < x < b \end{cases}$$

on considère finalement le problème PML posé dans $\Omega_b$:

$$\begin{cases} \dfrac{\partial^2 u}{\partial y^2} + \tilde{\alpha}\dfrac{\partial}{\partial x}\left(\tilde{\alpha}\dfrac{\partial u}{\partial x}\right) + k^2 u = 0 & \text{sur } \Omega_b = \Omega \cap ]0,b[\times]0,h[ \\ \dfrac{\partial u}{\partial n} = 0 & \text{sur } \Gamma \cup \partial\mathcal{O} \cup \Sigma_b \\ \dfrac{\partial u}{\partial n} = g & \text{sur } \Sigma_0 \\ \dfrac{\partial u}{\partial n} = 0 & \text{sur } \Sigma_b. \end{cases} \tag{4}$$

dont la formulation variationnelle est : trouver $u \in H^1(\Omega)$ telle que $\forall v \in H^1(\Omega)$

$$\int_{\Omega_b} \frac{1}{\tilde{\alpha}} \frac{\partial u}{\partial y}\frac{\partial \bar{v}}{\partial y} + \int_{\Omega_b} \tilde{\alpha}\frac{\partial u}{\partial x}\frac{\partial \bar{v}}{\partial x} - k^2 \int_{\Omega_b} \frac{1}{\tilde{\alpha}} u\,\bar{v} = \int_{\Sigma_0} g\,\bar{v}.$$

En pratique on prend $\alpha$ constant.

### 1.3 Méthode DtN

La représentation en série de la solution extérieure (2) permet de construire la condition de raccord dite Dirichlet to Neumann qui raccorde les valeurs et dérivées de la solution à l'intérieur avec les valeurs et dérivées de la solution à l'extérieur :

$$\frac{\partial u}{\partial n} = Tu \overset{def}{=} \sum_{n\geq 0} i\,\beta_n \left(\int_{\Sigma_a} u\varphi_n\right)\varphi_n(y) \text{ sur } \Sigma_a \ (a > L).$$

En pratique la série définissant l'opérateur $T$ est tronquée au rang $N$ :

$$T_n u = \sum_{0\leq n\leq N} i\,\beta_n \left(\int_{\Sigma_a} u\varphi_n\right)\varphi_n(y) \text{ sur } \Sigma_a \ (a > L).$$

Ce qui conduit au problème approché par DtN :

$$\begin{cases} \triangle u + k^2 u = 0 & \text{sur } \Omega_a \\ \dfrac{\partial u}{\partial n} = 0 & \text{sur } \Gamma \cup \partial\mathcal{O} \\ \dfrac{\partial u}{\partial n} = g & \text{sur } \Sigma_0 \\ \dfrac{\partial u}{\partial n} = T_N u & \text{sur } \Sigma_a \ (a > L). \end{cases} \tag{5}$$

dont la formulation variationnelle est : trouver $u \in H^1(\Omega_a)$ telle que $\forall v \in H^1(\Omega_a)$

$$\int_{\Omega_a} \nabla u \nabla \bar{v} - k^2 \int_{\Omega_a} u\,\bar{v} - i \sum_{0\leq n\leq N} \beta_n \int_{\Sigma_a} u(x)\,\varphi_n(x)\,dx \int_{\Sigma_a} \bar{v}(y)\,\varphi_n(y)\,dy = \int_{\Sigma_0} g\,\bar{v}.$$

### 1.4 Méthode DtN épais

La méthode DtN épais consiste à utiliser la formule de représentation (2) pour construire un opérateur DtN sur une autre section $\Sigma_b$ $(b > a)$. On a en effet

$$\frac{\partial u}{\partial x}(b,y) = \sum_{n\geq 0} i\,\beta_n \left(\int_{\Sigma_a} u\varphi_n\right) e^{i\beta_n(b-a)}\varphi_n(y).$$

En tronquant à nouveau au rang $N$ :

$$\tilde{T}_N u = \sum_{n\geq 0} i\,\beta_n e^{i\beta_n(b-a)} \left(\int_{\Sigma_a} u\varphi_n\right) \varphi_n(y) \text{ sur } \Sigma_b \ (b > a).$$

on introduit le problème approché :

$$\begin{cases} \triangle u + k^2 u = 0 & \text{sur } \Omega_b \\ \dfrac{\partial u}{\partial n} = 0 & \text{sur } \Gamma \cup \partial\mathcal{O} \\ \dfrac{\partial u}{\partial n} = g & \text{sur } \Sigma_0 \\ \dfrac{\partial u}{\partial n} = \tilde{T}_N u & \text{sur } \Sigma_b. \end{cases} \tag{6}$$

dont la formulation variationnelle est : trouver $u \in H^1(\Omega_a)$ telle que $\forall v \in H^1(\Omega_a)$

$$\int_{\Omega_b} \nabla u \nabla \bar{v} - k^2 \int_{\Omega_b} u\,\bar{v} - i \sum_{0\leq n\leq N} \beta_n \int_{\Sigma_a} u(x)\,\varphi_n(x)\,dx \int_{\Sigma_b} \bar{v}(y)\,\varphi_n(y)\,dy = \int_{\Sigma_0} g\,\bar{v}.$$

# 2 Approximation par éléments finis

Afin de résoudre les problèmes précédents, on utilise une méthode d'éléments finis de Lagrange. Dans le cadre de ce TP, on s'appuiera sur la librairie XLiFE++ (eXtended Library of Finite Element in C++) qui permet de manipuler simplement les principaux objets intervenant dans une méthode d'éléments finis.

## 2.1 Librairie XLiFE++

Plus précisément, la librairie XLiFE++ gère :

- des **objets géométriques**, par exemple le rectangle $]0, a[\times]0, h[$ est défini de la façon suivante:

```
Number ny=30, na=Number(ny*a/h);
Rectangle Ra(_xmin=0,_xmax=a,_ymin=0, _ymax=h, _hsteps=hs, //hs pas de maillage
             _domain_name="Omega_a",
             _side_names=Strings("Gamma_a","Sigma_a","Gamma_a","Sigma_0"));
```

  le disque de centre $(0.5, 0.5)$ et de rayon 0.1 :

```
Number nd=Number(0.5*ny*pi*r/h);
Disk D(_center=Point(0.5,0.5),_radius=0.1,_hsteps=hs);
```

- des **maillages**, par exemple la triangulation basée sur les points du bord du rectangle précédent est :

```
Mesh ma(Ra, _shape=_triangle, _order=1, _geneator=_gmsh);
```

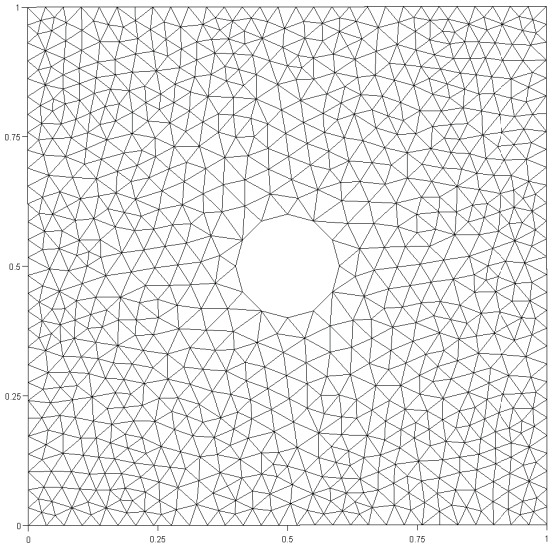
  la clé _gmsh indique que l'on utilise le mailleur GMSH et la triangulation du rectangle Ra privé du disque D est obtenue par la commande :

```
Mesh md(Ra-D, _shape=_triangle, _order=1, _geneator=_gmsh);
```

- des **domaines géométriques**, par exemple les domaines de calcul $\Omega_a$, $\Sigma_0$ et $\Sigma_a$, appelés respectivement "Omega_a", "Sigma_0" et "Sigma_a" dans le maillage sont identifiés en tant qu'objet par les commandes :

```
Domain Omega_a = ma.domain("Omega_a");
Domain Sigma_a = ma.domain("Sigma_a");
Domain Sigma_0 = ma.domain("Sigma_0");
```

- des **espaces d'approximation**, par exemple l'espace d'approximation par éléments finis de Lagrange $P^2$ sur le domaine $\Omega$ est instancié de la façon suivante :




```
Space V(_domain = Omega_a, _interpolation = P2, _name = "V");
Unknown u(V, _name="u");
TestFunction v(u, _name="v");
```

Les objets **inconnue** et **fonction test** sont des éléments symboliques de l'espace d'approximation $V$.

- des **formes bilinéaires** ou **linéaires**, par exemple la forme bilinéaire attachée au problème (3) :

$$a(u,v) = \int_{\Omega_a} \nabla u \nabla \bar{v} - k^2 \int_{\Omega_a} u\,\bar{v} - ik \int_{\Sigma_a} u\,\bar{v}$$

se définit de la façon suivante:

```
BilinearForm a = intg(Omega_a, grad(u)|grad(v)) - k*k*intg(Omega, u*v)
               - i*k*intg(Sigma_a, u*v);
```

et la forme linéaire

$$l(v) = \int_{\Sigma_0} g\,\bar{v}$$

se traduit par

```
LinearForm lg = intg(Sigma_0, g*v);
```

où g est une fonction C++ "ordinaire" mais standardisée, définie par l'utilisateur avant le programme *main*, par exemple :

```
Complex g(const Point& P, Parameters& pa = defaultParameters)
{ Real y=P(2);    Number n=0;
  Complex betan=sqrt(Complex(k*k-n*n*pi*pi/(h*h)));
  return -i*betan*sqrt(2./h)*cos(n*pi*y/h);
}
```

Notez qu'il n'est pas nécessaire de conjuguer la fonction test $v$ dans la définition des formes (bi)linéaires de XLiFE++ car les fonctions de bases de l'espace sont à valeurs réelles. Néanmoins le résultat du calcul pourra être à valeurs complexes !

- des objets **matrice** ou **vecteur** qui sont les représentations algébriques de formes bilinéaires ou linéaires définies sur l'espace d'approximation, par exemple :

```
TermMatrix A(a,"A");
TermVector B(lg,"B");
```

Une fois ces objets définis, leur calcul est automatiquement réalisé; l'objet A contenant la matrice de coefficients $a(w_j, w_i)$ et l'objet B le vecteur de composantes $lg(w_i)$. Le stockage de la matrice est creux mais l'utilisateur n'a pas à s'en préoccuper.

On résout alors le système linéaire, par exemple à l'aide d'une méthode directe qui est choisie automatiquement, l'objet résultant étant du type TermVector:

```
TermVector U=directSolve(A,B);
```

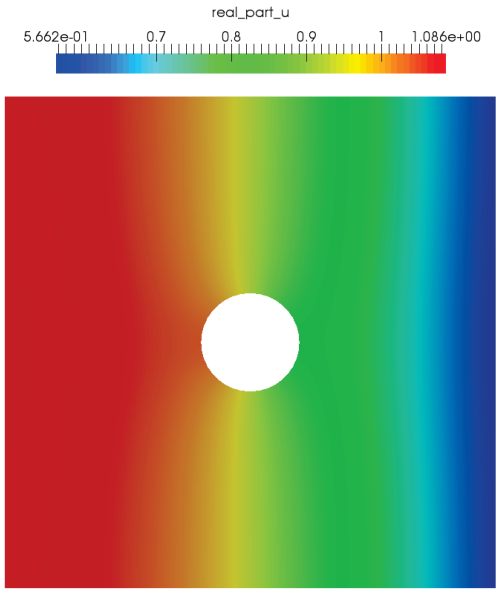
Ici $U$ est le vecteur des composantes $(U_i)$ de la solution approchée $u_h$ :

$$u_h(x,y) = \sum_i U_i w_i(x,y), \quad (w_i) \text{ fonctions de base.}$$

Le résultat U peut être sauvegardé dans un fichier au format vtu :

```
saveToFile("solU",U,_format=vtu);
```

qui pourra être visualisé à l'aide du logiciel graphique *paraview* :




## 2.2 Traitement des opérateurs DtN

Les opérateurs DtN intervenant dans les problèmes (5) et (6) donnent lieu à des termes particuliers dans les formulations variationnelles :

$$\sum_{0\le n\le N} \gamma_n \int_{\Sigma_a} u(x)\,\varphi_n(x)\,dx \int_{\Sigma_a} \bar{v}(y)\,\varphi_n(y)\,dy \text{ ou } \sum_{0\le n\le N} \gamma_n \int_{\Sigma_a} u(x)\,\varphi_n(x)\,dx \int_{\Sigma_b} \bar{v}(y)\,\varphi_n(y)\,dy.$$

Dans le contexte de XLiFE++, on réinterprète ces quantités sous la forme d'intégrales doubles à noyau:

$$\int_{\Sigma_a}\int_{\Sigma_a} u(x)\,K(x,y)\bar{v}(y)\,dx\,dy \text{ ou } \int_{\Sigma_b}\int_{\Sigma_a} u(x)\,K(x,y)\bar{v}(y)\,dx\,dy$$

avec

$$K(x,y) = \sum_{0\le n\le N} \gamma_n\,\varphi_n(x)\varphi_n(y).$$

Ces noyaux particuliers sont des objets de type TensorKernel dans XLiFE++ que l'on définit à partir d'espaces "spectraux" construits à partir de fonctions de bases analytiques :

```
Number N=5;  //nombre de modes
Space S(_domain = Sigma_a, _basis = Function(cosn, params), _dim = N,
        _name = "cos(n*pi*y)");      //espace spectral
Unknown phi(S,"phi");
Complexes gamma(N);
for(Number n=0; n<N; n++) gamma[n]=i*sqrt(Complex(k2-n*n*pi*pi/(h*h)));
TensorKernel K(phi,gamma);
```

où on a défini une liste de paramètres (*params*)

```
Parameters params;
Real h=1., k=2.;
params << Parameter(h,"h")<< << Parameter(k,"k");
```

utilisée par la fonction *cosn* définie par l'utilisateur:

```
Real cosn(const Point& P, Parameters& pa = defaultParameters)
{
  Real y=P(2);
  Real h=pa("h");                          //hauteur du guide
  int_t n=pa("basis index")-1;             //index de la fonction a calculer
  if(n==0) return std::sqrt(1./h);
  else     return std::sqrt(2./h)*std::cos(n*pi*y/h);
}
```

Noter que l'indice $n$ de la fonction spectrale à évaluer est obtenue depuis l'objet *pa* (Parameters) à l'aide de la clé "basis index"!

Les objets TensorKernel sont utilisés pour définir les intégrales doubles, par exemple:

```
BilinearForm a2 = intg(Sigma_a,Sigma_a,u*K*v);
```

### 2.3 Calcul des coefficients de diffraction

Afin de comparer les différentes méthodes, on peut également comparer les coefficients de diffraction, c'est-à-dire

$$a_n = \int_{\Sigma_a} u\,\varphi_n.$$

Pour les obtenir dans XLiFE++, il suffit de faire le produit de la matrice $C_{nj} = \int_{\Sigma_a} w_j\,\varphi_n$ avec le vecteur solution:

```
TermMatrix C(intg(Sigma_a,u*phi));
TermVector a=C*U;
```

### 2.4 Implémentation

• Programmer chacune des formulations variationnelles à l'aide de la librairie XliFE++. Afin de mettre au point le code, on pourra utiliser la situation où il n'y a pas d'obstacle de telle sorte que tout mode propagatif est une solution. On peut ainsi calculer la norme L2 de l'erreur :

```
TermMatrix M(intg(Omega,u*v));
TermVector Uex(u,Omega,uex);
TermVector E=U-Uex;
Real er=sqrt(abs((M*E|E)));
cout<<"L2 error = "<<er<<eol;
```

Ici `uex` est une fonction C++ définie par l'utilisateur:

```
Complex uex(const Point& P, Parameters& pa = defaultParameters)
{
  ...
}
```

• Conduire une série d'expériences numériques dans diverses situations pour identifier les "bons" paramètres de calcul (pas de maillage, nombre de modes dans les méthode DtN, coefficient et taille de la PML).

Afin de tester plusieurs configurations sans recompiler il est possible d'utiliser un fichier de données externe (e.g *data.text*) qui sera analysé via un objet XLiFE++ de type `Options` :

```
Real k, h, a, b, x_alpha;   // variables globales
Number n0;

int main(int argc, char **argv)
{
  init(argc, argv, _lang = fr);
  Options opts;
  opts.add("k",10.);
  opts.add("n0",1); opts.add("nbmode",5);  opts.add("alpha",1.);
  opts.add("a",3.); opts.add("b",4.);opts.add("h",1.);
  opts.add("step",0.05);
  opts.add("obs",0);
  opts.add("R",0.25); opts.add("C",Reals(0.5,0.5));
  opts.parse("data.txt");  // lit ces parametres
  // Parametres du probleme physique
  h = opts("h");                  // hauteur du guide
  k = opts("k");                  // nombre d'onde
  a = opts("a"); b = opts("b") ; // largeurs du guide
  n_0 = opts("n0");               // mode propagatif excitant en sigma_0
  x_alpha = opts("alpha"); ;      // facteur de la partie imaginaire du alpha PML
  Number N = opts("nbmode");      // nombre de modes DtN
  Real hs=opts("step");           // pas de maillage
  int obs=opts("obs");            // avec(1) ou sans(0) obstacle
  Real r = opts("R");             // rayon du disque
  Reals C = opts("C");            // centre du disque
  ...
}
```

Le fichier de données sera alors de la forme suivante :

```
h 1.
k 20.
n0 0
nbmode 5
alpha 0.1
b 4.
a 3.
step 0.05
obs 1
R 0.25
C 0.5 0.6
```

l'ordre des données étant indifférent et il peut même en manquer certaines; c'est la valeur par défaut qui sera alors utilisée.