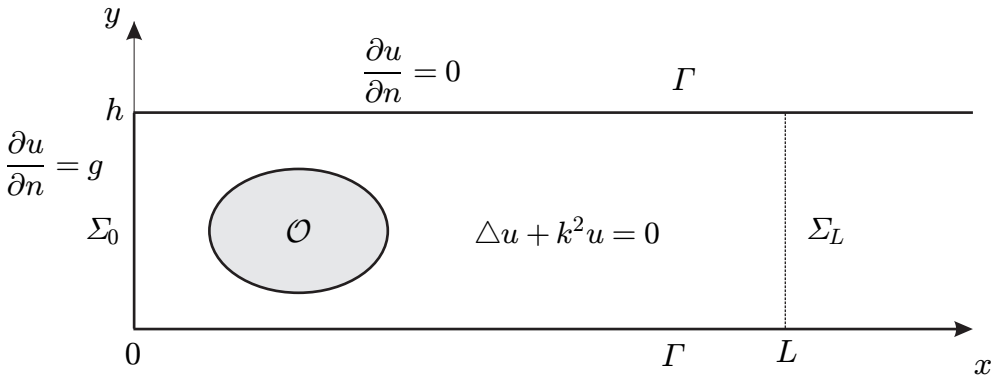
XLiFE++ TP2
Problème de HelmHoltz dans un guide d'ondes

E. Lunéville

2015

L'objet de ce TP est de tester différentes techniques de résolution numérique de la propagation d'ondes en régime harmonique dans un guide d'onde 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 :

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

où

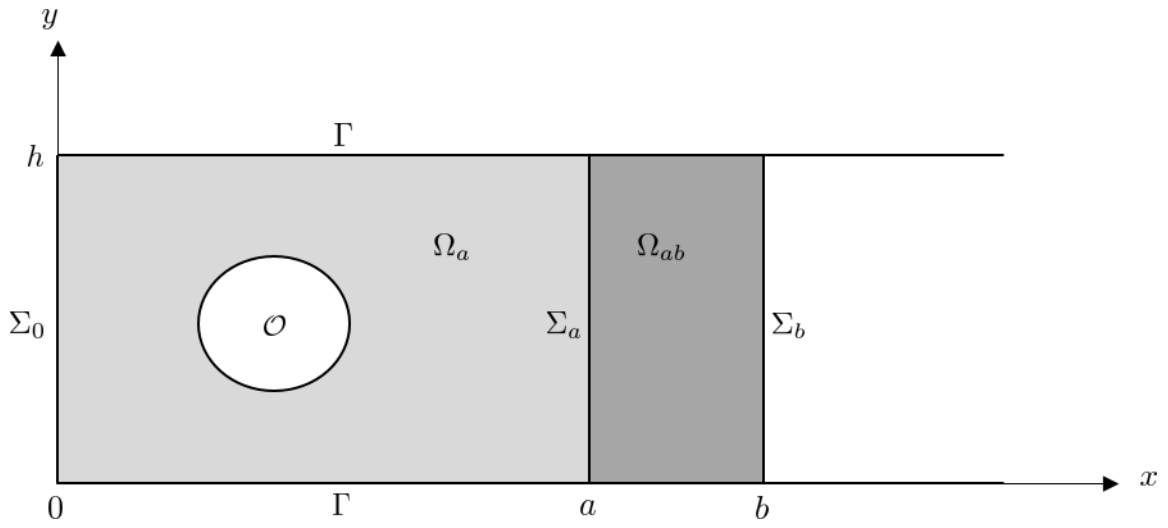
$$\beta_n = \sqrt{k^2 - \left(\frac{n\pi}{h}\right)^2} \text{ avec } \mathcal{I}m\,\beta_n \geq 0 \text{ et } \mathcal{R}e\,\beta_n \geq 0$$
$$\varphi_n(y) = a_n \cos(\frac{n\pi y}{h}) \text{ avec } a_n = \sqrt{\frac{2}{h}} \text{ si } n > 0 \text{ et } a_0 = \sqrt{\frac{1}{h}}.$$

On rappelle qu'à l'extérieur de la zone perturbée, la solution se représente à l'aide d'une série (Σ_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
couplées à une approximation par éléments finis.

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



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 $\forall v \in H^1(\Omega_a)$

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

– A l'aide de XLiFE++, programmer la résolution par éléments finis P1 et P2 de ce problème. On testera le programme en calculant l'erreur L2 commise dans le cas où la solution est le mode plan.
– Trouver des situations mettant en défaut l'approximation onde plane.
– Remplacer la condition de Neumann $\partial_n u = g$ sur Σ_0 par une condition de Dirichlet $u = f$ sur Σ_0.

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 α 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 Ω_b :

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

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

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

En pratique on prend α constant.

– Programmer la résolution par éléments finis P1 et P2 de ce problème.
– Etudier l'influence du coefficient α

Annexe

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.

Principaux objets XLiFE++

– 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,_nnodes=Numbers(na,ny),
             _domain_name="Omega_a",
             _side_names=Strings("Gamma_a","Sigma_a","Gamma_a","Sigma_0"));
```

ny, na indiquant un nombre de points sur les bords du rectangle,
le disque de centre $(0.5, 0.5)$ et de rayon 0.1 :

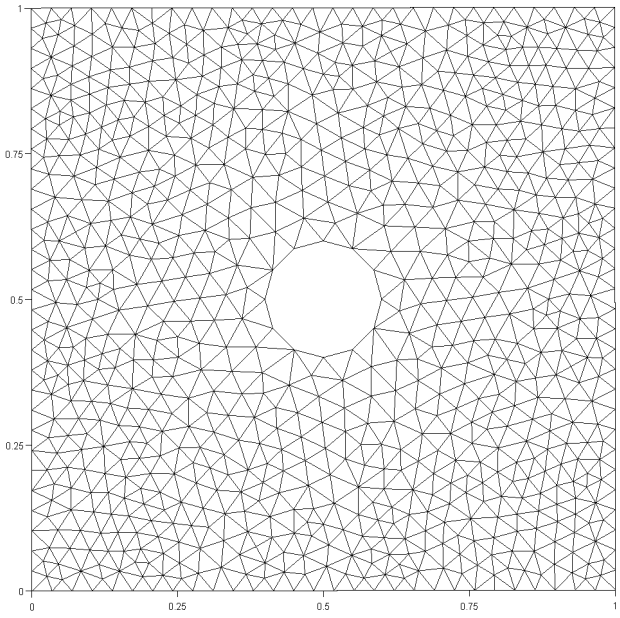
```
Number nd=Number(0.5*ny*pi*r/h);
Disk D(_center=Point(0.5,0.5),_radius=0.1,_nnodes=nd);
```

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

```
Mesh mail(R1,_triangle,1,_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 mail2(Ra-D,_triangle,1,_gmsh);
```



– des domaines géométriques, par exemple les domaines de calcul Ω, Σ_0 et Σ_a, appelés respectivement "Omega", "Sigma_0" et "Sigma_a" dans le maillage sont identifiés en tant qu'objet par les commandes :

```
Domain Omega = mail.domain("Omega");
Domain Sigma_a = mail.domain("Sigma_a");
Domain Sigma_0 = mail.domain("Sigma_0");
```

– des espaces d'approximation, par exemple l'espace d'approximation par éléments finis de Lagrange P^2 sur le domaine Ω est instancié de la façon suivante :

```
Space V(Omega,_P2,"V",true);
Unknown u(V,"u");
TestFunction v(u,"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,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 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 les formes (bi)linéaires de XLiFE++ car les fonctions de bases de l'espace sont à valeurs réelles. Néanmoins le résultat du calcul pourra être complexe !

– 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 soucier.

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);
```

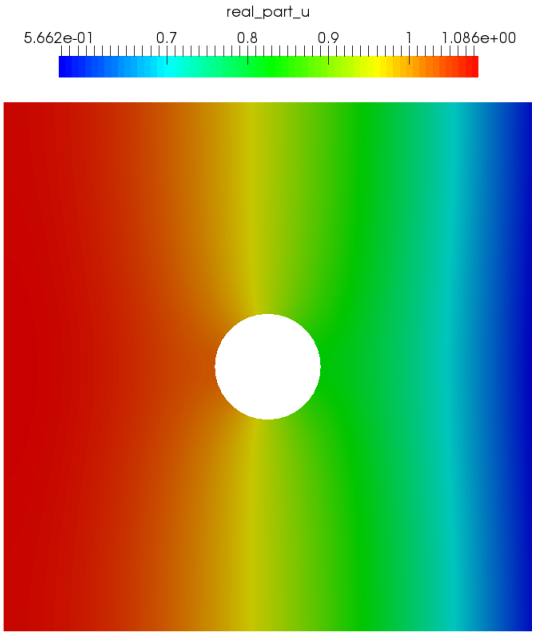
Ici U est le vecteur des composantes (U_i) de la solution approchée u_h :

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

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

```
saveToFile("solU",U,_vtu);
```

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



Calcul d'une erreur L2

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 de perturbation 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)
{
...
}
```

Correction

On a réuni dans un même code le calcul la formulation basse fréquence et la formulation PML.

```
#include "xlife++.h"
using namespace xlifepp;
using namespace std;

Complex i(0,1);
Real k=5;
Real h=1;
//data function on Sigma0
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);  //normal derivative on Sigma0 of mode n
}
//exact solution
Complex uex(const Point& P, Parameters& pa = defaultParameters)
{
Real x=P(1), y=P(2);
Number n=0;
Complex betan=sqrt(Complex(k*k-n*n*pi*pi/(h*h)));
return sqrt(2./h)*cos(n*pi*y/h)*exp(i*betan*x);
}

int main(int argc, char** argv)
{
init(_lang=en); // mandatory initialization of xlifepp
verboseLevel(1);

//define some parameters
Real a=1,  // abcissa of Sigma_a
b=1.2,     // abcissa of Sigma_b
k2=k*k;
Parameters params;
params << Parameter(h,"h")<< Parameter(k,"k");

//create Mesh
Number ny=30, na=Number(ny*a/h), nb=2*Number(ny*(b-a)/h);
Rectangle Ra(_xmin=0,_xmax=a,_ymin=0, _ymax=h,_nnodes=Numbers(na,ny),
_domain_name="Omega_a",_side_names=Strings("Gamma_a","Sigma_a","Gamma_a","Sigma_0"));
Rectangle Rb(_xmin=a,_xmax=b,_ymin=0, _ymax=h,_nnodes=Numbers(nb,ny),
_domain_name="Omega_b",_side_names=Strings("Gamma_b","Sigma_b","Gamma_b","Sigma_a"));

bool doError = false;
Mesh meshWG;
if(doError)  meshWG = Mesh(Ra+Rb,_triangle,1,_gmsh);  //for exact solution
else
{
Real r=0.1;
Number nd=Number(0.5*ny* pi* r/h);
Disk D1(_center=Point(0.5,0.5),_radius=r,_nnodes=nd,_domain_name="D1");
meshWG = Mesh(Ra+Rb-D1,_triangle,1,_gmsh);
}
Domain Omega_a = meshWG.domain("Omega_a"), Omega_b = meshWG.domain("Omega_b");
Domain Sigma_0 = meshWG.domain("Sigma_0"),
Sigma_a = meshWG.domain("Sigma_a"),
Sigma_b = meshWG.domain("Sigma_b");
```

```
// create interpolation space on Omega
Domain Omega=merge(Omega_a,Omega_b,"Omega");
Space V(Omega,_P2,"V",true);
Unknown u(V,"u"); TestFunction v(u,"v");

//create mass matrix
TermMatrix M1(intg(Omega_a,u*v));
compute(M1);
TermVector Uex(u,Omega_a,uex);

elapsedTime("mesh and space creation");

//------------------   solve low frequency approximation ------------------
// create system and solve it
BilinearForm alf = intg(Omega_a,grad(u)|grad(v)) - k2*intg(Omega_a,u*v)
- i*k*intg(Sigma_a,u*v);
LinearForm f= intg(Sigma_0,g*v);
TermMatrix Alf(alf, "Alf");
TermVector B(f,"B");
TermVector Ulf=directSolve(Alf,B);
elapsedTime("solve low frequency approximation");
saveToFile("Ulf",Ulf,_vtu);
if(doError) //compute L2 error
{
TermVector Erlf=Ulf-Uex;
Real erlf=sqrt(abs((M1*Erlf|Erlf)));
cout<<" low frequency approximation L2 error = "<<erlf<<eol;
}

//------------------ solve using PML method ------------------
Complex alpha=(1-i)/20;     //PML coefficient
Complex am=1./alpha;
//create system and solve it
BilinearForm apml =  intg(Omega_a,grad(u)|grad(v)) - k2*intg(Omega_a,u*v)
            +  am*intg(Omega_b,dy(u)|dy(v)) + alpha*intg(Omega_b,dx(u)|dx(v))
            - (k2*am)*intg(Omega_b,u*v);
TermMatrix Apml(apml,"Apml");
TermVector Upml=directSolve(Apml,B);
elapsedTime("solve PML approximation");
saveToFile("Upml",Upml,_vtu);
if(doError) //compute L2 error
{
TermVector Erpml=Upml-Uex;
Real erpml=sqrt(abs((M1*Erpml|Erpml)));
cout<<" PML approximation L2 error = "<<erpml<<eol;
}
```

Pour le nombre d'onde $k = 5$ (2 modes propagatifs) et une excitation correspondant au mode plan, on obtient pour chacune des méthodes les 3 premiers coefficients de diffraction suivants :

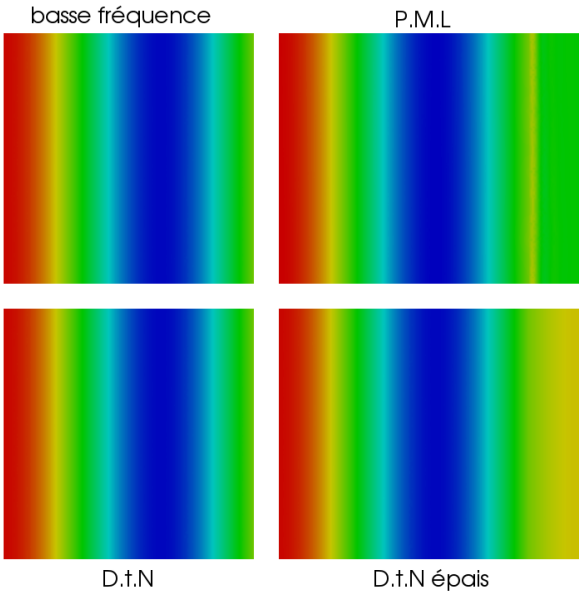
basse fréquence :	(0.401156,-1.35612)	(1.39069e-008,-1.92673e-009)	(-3.2017e-008,-5.28809e-008)
PML :	(0.407728,-1.35771)	(-0.00204145,0.00111319)	(0.00192726,-0.0029947)

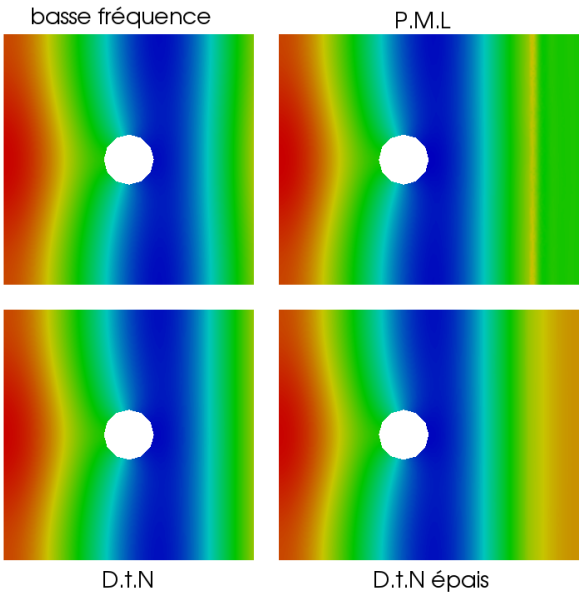
montrant ainsi la bonne qualité des approximations. On note toutefois que la méthode PML est moins précise.

Les figures suivantes montrent les parties réelles des champs obtenus par chacune des méthodes :

Il est facile de traiter des situations géométriques plus intéressantes en changeant par exemple le maillage. On obtient dans ce cas (obstacle circulaire centré) les résultats suivants :

basse fréquence :	(0.359186,-1.12153)	(3.71856e-008,-1.5774e-006)	(-0.042527,0.0329753)
PML :	(0.363767,-1.12183)	(0.00024973,-0.00229242)	(0.00619306,0.0399524)


basse fréquence
P.M.L
D.t.N
D.t.N épais


basse fréquence
P.M.L
D.t.N
D.t.N épais