INP Toulouse - ENSEEIHT

Bureau d'étude

Méthodes numériques pour les problèmes de diffraction

BE méthodes intégrales

Rédigé par :
AZZA Ayoub
YACOUBI Mohamed Reda

Encadré par :
POIRIER Jean-René

Année 2022-2023

Table des matières

Introduction

Ce BE s'inscrit dans la continuité du BE de méthode numérique. Notre objectif consiste à développer les opérateurs intégraux en deux dimensions, et par la suite étudier les performances de chacun. Le travail a été divisé en deux parties principales : la première concerne le travail théorique, et la seconde vise à développer un code Fortran.

1 Travail théorique

Le but de cette première partie est d'établir la résolution du problème de diffraction 2D en mode TM.

$$\begin{cases} u = E_z, u^i = E_z^{inc} & \\ E_z^{tot} = 0 & \text{sur } \Gamma \\ E_z^{diff} = -E_z^{inc} & \text{sur } \Gamma \end{cases} \tag{1.1}$$

En utilisant la représentaion du champ par un potentiel de double couche :

$$u^d(r') = \int_\Gamma \partial_{n_r} G(r, r')\varphi(r) d\gamma(r) \qquad \forall r' \notin \Gamma$$

Donc

$$u^d(r') = D(\varphi(r'))$$

En utilisant le théorème de discontinuité du potentiel double couche à la traversé de Γ, et en exploitant la condition aux limites sur le bord :

$$u^{tot}(r') = 0 = u^d(r') + u^i(r') \qquad \forall r' \in \Gamma$$

On trouve :

$$\lim_{r' \to \Gamma} \int_\Gamma \partial_{n_r} G(r, r')\varphi(r) d\gamma(r) = -u^i(r') \qquad \forall r' \in \Gamma$$

Ce qui donne bien le résultat :

$$\frac{\varphi(r')}{2} + \int_\Gamma \partial_{n_r} G(r, r')\varphi(r) d\gamma(r) = -u^i(r') \qquad \forall r' \in \Gamma$$

$$\frac{\varphi(r')}{2} + D_0(\varphi)(r') = -u^i(r') \qquad \forall r' \in \Gamma$$

Formulation variationnelle :

$$< D_0(\varphi), \varphi' >= - < u^i(r'), \varphi') \qquad \forall \varphi' \in V$$

Discrétisation :

On prend $\varphi = \sum_{i=1}^{N} \varphi_i B_i$ et $\varphi' = B_j \quad j = 1...N$ ce qui donne :

$$\sum_{i=1}^{N} \varphi_i < D_0(B_i) + \frac{1}{2} B_i, B_j >= - < u^i, B_j > \quad j = 1...N$$

D'où la matrice $A_{ij} =< D_0(B_i) + \frac{1}{2}B_i, B_j >=< D_0(B_i), B_j > + \frac{1}{2} < B_i, B_j >$
Et le scond membre $b_j = - < u^i, B_j >$
Approximation de l'intégrale :
Pour $i \neq j$:

$$< D_0(B_i), B_j > + \frac{1}{2} < B_i, B_j >=< D_0(B_i), B_j >= \int_{\Gamma_i}\int_{\Gamma_j} \partial_{n_r} G(r, r') B_i(r') B_j(r) d\gamma(r) d\gamma(r')$$

Donc :

$$A_{ij} =< D_0(B_i), B_j > \sim \partial_{n_r} G(r_{m_i}, r'_{m_j}) |\Gamma_i||\Gamma_j|$$

Sachant que :

$$\begin{cases} G(r, r') = -\frac{j}{4} H_0^{(2)}(k||r - r'||) \\ \partial_{n'_r} G(r, r') = \frac{1}{|\Gamma'_n|} \frac{jkH_1^2(k||r-r'||)}{4(k||r-r'||)} (\Delta y'(x' - x) - \Delta x'(y' - y)) \\ n_r = \frac{1}{||r_n - r_{n+1}||} (\Delta y, \Delta x)^t \end{cases} \tag{1.2}$$

avec $r = (x, y)$; $\quad r' = (x', y')$; $\quad \Delta y = y_{n+1} - y_n$; $\quad \Delta x = x_{n+1} - x_n$
Et :

$$b_j = - < u^i(r), B_j(r') >= - \int_{\Gamma_j} u^i(r) B_j(r') d\gamma(r') \sim -u^i(r'_{m_j}) |\Gamma_j|$$

Donc :

$$b_j = e^{i(k_x x_{m_j} - k_y y_{m_j})} |\Gamma_j|$$

Pour $i = j$ cette approximation n'est plus possible et le terme $< D_0(B_i), B_j >$ est négligeable devant $\frac{1}{2} < B_i, B_j >$, on a donc :

$$A_{ii} = \frac{1}{2} < B_i, B_i >= \frac{1}{2}$$

2 Développement d'un code Fortran

Tout d'abord, Avant d'entamer la partie programmation, on vérifie bien que la valeur du champ Ediff théorique et celui calculé à partir des méthodes numérique sont presque identiques (l'erreur entre les deux est négligeable)

E = 0.255802505452501 + 0.258187796523942i
Ediff = 0.256971914249610 + 0.256247582817562i

on crée un module maillage, où on définit une structure segment avec les différents attributs nécessaires à savoir : le milieu du segments, la longueur ainsi que la normale.

- Implémentation du second membre :

```
1 do i=1,n-1
2   b(i)=exp(j*(kx*s(i)%milieu%x-ky*s(i)%milieu%y))*s(i)%longueur
3 print*,b(i)
```

On vérifie bien que la valeur du second membre trouvé à travers Matlab est presque égale a celle trouvée par Fortran. On a fait le test sur 10 valeurs de la liste de b, où on a bien trouvé les valeurs attendues.

- Implémentation du calcul de la matrice A :

```
do u=1,n-1
  do v=1,n-1
    if (u==v) then
      A(u,u)=0.5*s(u)%longueur
    else
      z=k*longueur(s(u)%milieu,s(v)%milieu)
      hankel=besj1(z)+(0,1)*besy1(z)
      l=-((s(u)%p2%y-s(u)%p1%y)*(s(u)%milieu%x-s(v)%milieu%x)-(s(u)%p2%x-s(u)%p1%
          x)*(s(u)%milieu%y-s(v)%milieu%y))
      A(u,v)=-s(v)%longueur*hankel/z*j/4.0*k**2*l
    end if
  end do
end do
```

Maintenant, on teste les outils qui nous permettent d'utiliser les fonctions de résolution du système linéaire, et on les intègre par la suite afin qu'on puisse résoudre notre système linéaire.
On lance le calcul de solution par l'une des méthodes (par exemple gmres) et on vérifie bien la compatibilité des deux résultats (Matlab & Fortran).
Rappel : Notre champ E est définit à partir de l'opérateur potentiel double couche extérieur : D(x ; y ; x0 ; y0 ; k ; u).

```
do u=1,180
 theta=(u-1)/real(180)
 x0=infinity*cos(theta*pi)
 y0=infinity*sin(theta*pi)
 E=0.0
  do v=1,n-1
   rho0=sqrt((s(v)%milieu%x-x0)**2+(s(v)%milieu%y-y0)**2)
   w=(s(v)%p2%x-s(v)%p1%x)*(s(v)%milieu%y-y0)
   E=E+sol(v)*(besj1(k*rho0)+j*besy1(k*rho0))*((s(v)%p2%y-s(v)%p1%y)*(s(v)%
       milieu%x-x0)- w)/rho0
  end do
    E=-k*E*j/4.0
    serdb(u)=10*log10(2*pi*infinity*abs(E)**2)
end do
```

Le script précèdent correspond au code permettant de calculer la SER bistatique. Après L'exécution de ce dernier, on a eu des résultats différents à celui de Matlab même que le script implémenté était correcte. Du coup on a pensé que le problème vient de la valeur de π sur Fortran vu qu'on a mis une valeur approchée. Donc, on a changé la valeur de π sur Matlab à 3.14159 ce qui nous a permet d'avoir par la suite des résultats pour la SER avec une grande précision. On a essayé également de tracer la SER monostatique, mais les deux graphes obtenus ne coïncident pas parfaitement.

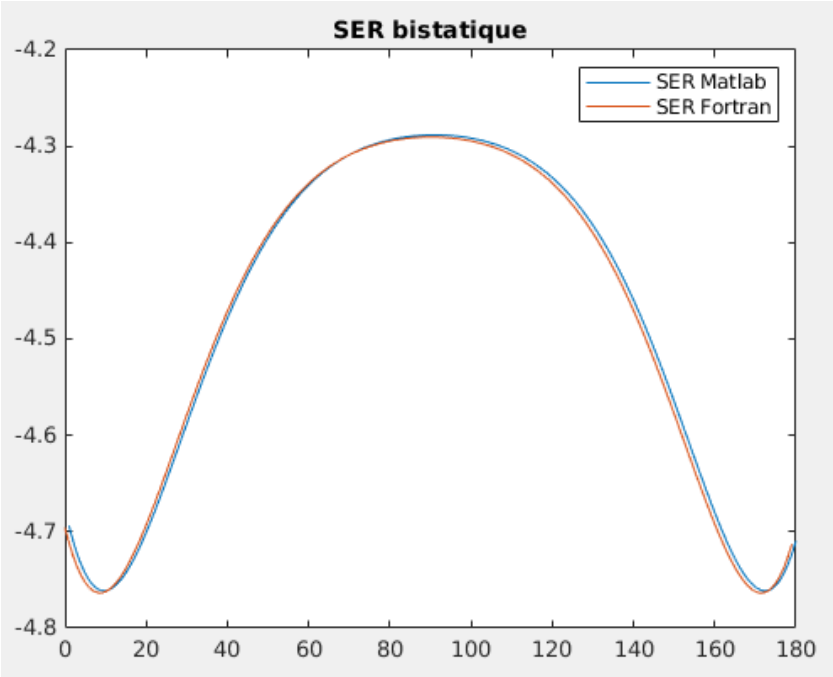

Figure 1 – SER Bistatique pour la géometrie cylindrique par la méthode gauss

```
1  do u=1,180
2   theta=(u-1)/real(180)
3   kx=k*sin(theta*pi)
4   ky=k*cos(theta*pi)
5   x0=infinity*cos(theta*pi)
6   y0=infinity*sin(theta*pi)
7    do v=1,n-1
8     b(v)=exp(j*(kx*s(v)%milieu%x-ky*s(v)%milieu%y))*s(v)%longueur
9    end do
10    call gauss(A,n-1,n-1,b,sol,0,P)
11    Em=(0.0,0.0)
12    do v=1,n-1
13      rho0=sqrt((s(v)%milieu%x-x0)**2+(s(v)%milieu%y-y0)**2)
14      w=(s(v)%p2%x-s(v)%p1%x)*(s(v)%milieu%y-y0)
15      Em=Em+sol(v)*(besj1(k*rho0)+j*besy1(k*rho0))*((s(v)%p2%y- s(v)%p1%y)*(s(v)%
            milieu%x-x0)- w)/rho0
16      Em=-k*Em*j/4.0
17      serdbm(i)=10*log10(2*pi*infinity*abs(Em)**2)
18    end do
```

3 Conclusion

Durant ce BE, on a pu établir la résolution du problème de diffraction 2D en mode TM, et par la suite on a développé notre code Fortran en commençant par la définition des différents fonctions ainsi que les subroutines, jusqu'à la trace de SER.

4 Annexe

Readme :
Pour bien comprendre le code, on a défini sur ‘maillage.f90’ nos différents fonctions à savoir la normale, le milieu, la longueur qui caractérisent nos segments. Et sur le ‘main.f90’ qui représente notre programme principale, on a fait appel aux sous programmes maillage et solveur, où on a implémenté également le second membre ainsi que la matrice A. Pour exécuter le code on se place sur un terminal, où on tape la commande ‘make’ (il ne permet également de vérifier qu’il n y a pas d’erreur dans notre code), et après, on tappe la commande ‘./main’ pour exécuter notre ‘main’ et on saisi par la suite la valeur du fréquence ainsi que l’angle d’incidence. Le fichier ‘routines.f90’ contient la définition des méthodes de résolution et celui de ‘makefile’ représente le fichier d’exécution du programme.