TP1 : Représentation intégrale pour l'équation de Helmholtz 2D

Clément Baillet

Le but de ce TP était de commencer à apprivoiser la mise en oeuvre d'un solveur BEM dans le cadre de l'équation d'Helmholtz en 2D. Nous sommes dans le cas de la diffraction d'une onde incidente plane, i.e. $u^{\text{inc}} = \mathrm{e}^{-i\mathbf{k}\cdot\mathbf{x}}$, par un disque centré en 0 et de rayon a. Pour ce faire, on commencera par coder un générateur de maillage en Python pour la frontière de ce disque, que nous noterons Γ et qui correspond au cercle de centre 0 et de rayon a. Face à l'absence de point de départ, nous avons sollicité ChatGPT pour la génération du maillage, en lui fournissant la requête suivante :

Prompt

Comment répondre à la question : "générez un maillage du bord du disque du centre 0 et de rayon a : d'abord les noeuds du maillage puis les extrémités de chaque segment. Quel est le nombre de segments (en fonction du nombre de noeuds du maillage) ?", en sachant que j'aimerais le coder en Python.

Voici le code obtenu en retour :

```python
def generate_nodes(N, a):
    angles = np.linspace(0, 2*np.pi, N, endpoint=False)
# angles uniformes
    x = a * np.cos(angles)
    y = a * np.sin(angles)
    return np.column_stack((x, y))

def generate_segments_indices(N):
    return [(k, (k+1) mod N) for k in range(N)]
```

Afin de vérifier le code, nous l'avons simplement relu ligne par ligne, afin de voir s'il faisait sens, puis nous l'avons exécuté (il y avait également une partie de code généré permettant d'afficher le maillage, mais nous ne l'avons pas inclus ici). Le code ayant été relu et le résultat final étant probant, nous sommes passés à la suite.

On peut maintenant procéder au calcul de la dérivée normale de la fonction $-u^+ - u^{\text{inc}}$, que l'on notera p. On cherche ici à connaître son expression explicite car elle nous permettra par la suite de calculer numériquement u^+, avant qu'on la détermine elle-même numériquement. Pour ce faire, il faut rappeler la formule du gradient en coordonnées polaires :

$$\nabla = \frac{\partial}{\partial r}\vec{u_r} + \frac{1}{r}\frac{\partial}{\partial \theta}\vec{u_\theta}$$

où $\vec{u_r}$ et $\vec{u_\theta}$ sont les vecteurs de la base mobile. Une fois ceci en tête, le calcul de la dérivée normale de p sur Γ devient direct, puisque $\nabla_{\vec{n}} = \nabla \cdot \vec{u_r} = \frac{\partial}{\partial r}$. On trouve donc, en dérivant sous le signe somme (possible en se plaçant sur un anneau compact autour de Γ) :

$$\nabla_{\vec{n}} p(a,\theta) = \sum_{n\in\mathbb{Z}} \left((-i)^n \frac{J_n(ka)}{H_n^{(1)}(ka)} \quad \underbrace{\frac{\mathrm{d}}{\mathrm{d}r}(H_n^{(1)})(ka)}_{=\frac{k}{2}(H_{n-1}^{(1)}(ka)-H_{n+1}^{(1)}(ka))} \quad \mathrm{e}^{in\theta} \right) + ik\cos\theta \mathrm{e}^{-ika\cos\theta}.$$

On la code en Python en se servant du module scipy et en important les fonctions de Bessel et de Hankel du premier ordre, puis en effectuant une boucle for pour calculer les termes successifs de la somme (on démarre la somme à $n = -60$ et l'arrête à $n = 60$). La figure 1 ci-dessous montre le graphe des parties réelle et imaginaire de p sur Γ en fonction de θ :

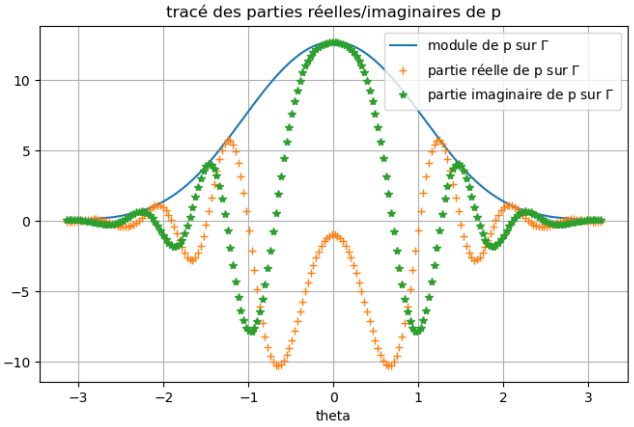

Figure 1: Partie réelle/imaginaire de p pour $k = 2\pi$, $N = 500$ et $a = 1.0$,

Maintenant que nous sommes capables de calculer numériquement p pour des cercles de rayon arbitraire, nous pouvons passer au calcul numérique de u^+ : pour ce faire, nous utilisons le fait que sur le domaine extérieur Ω_e, nous avons la représentation intégrale

$$u^+(\mathbf{x}) = \int_\Gamma G(\mathbf{x},\mathbf{y}) p(\mathbf{y}) \mathrm{d}\Gamma(\mathbf{y})$$

comme obtenue en cours, où

$$G(\mathbf{x},\mathbf{y}) = i/4 H_0^{(1)}(k\|\mathbf{x}-\mathbf{y}\|)$$

est la fonction de Green de notre problème. Pour calculer cette intégrale, on l'approxime sur un maillage du cercle Γ, en supposant que la valeur de p sur chaque segment correspond à sa valeur sur le milieu :

$$u^+(\mathbf{x_i}) = \sum_{e\in\text{segments}} p_e \int_{\Gamma_e} G(\mathbf{x_i},\mathbf{y}) \mathrm{d}\Gamma_e(\mathbf{y})$$

où $\mathbf{x}_i$ correspond à une discrétisation d'une partie du domaine extérieur. On note que le calcul revient à effectuer un produit matriciel entre la matrice des intégrales et le vecteur colonne des valeurs p_e. Pour calculer les intégrales, on va utiliser une quadrature de Gauss-Legendre :

$$\int_{-1}^{1} f(x)\mathrm{d}x \approx \sum_{i=1}^{n} w_i f(x_i)$$

où w_i sont des poids et x_i sont les racines du n^{e} polynôme de Legendre. Pour le coder, il est au préalable nécessaire de faire un changement de variable afin de pouvoir calculer des intégrales sur n'importe quel segment. Pour passer au calcul en 2D, on paramètre simplement le segment I entre (x_1, y_1) et (x_2, y_2) par

$$\gamma(t) = (x_1(1-t) + x_2 t, y_1(1-t) + y_2 t),$$

puis on utilise la formule suivante permettant de calculer les intégrales sur un contour :

$$\int_I f = \int_0^1 f \circ \gamma(t) \|\gamma'(t)\| \mathrm{d}t.$$

Le code de cette fonction intégrale consistant simplement en une boucle `for` qui reproduit explicitement la formule mentionnée ci-dessus, nous ne fournirons pas son code (qui a été testé et approuvé sur différentes fonctions usuelles, telles que des fonctions affines ou quadratiques, en comparant avec la fonction `quad`). Le code de la représentation intégrale consiste au final en différentes fonctions auxiliaires permettant de calculer les intégrales sur les segments dans notre maillage, et d'une fonction qui calcule la matrice des intégrales, le vecteur colonne et le produit des deux :

```
def representationIntegrale(N,points,poids,q,
                            G,segments,nodes,a,k):
    l = len(q)
    M = np.zeros((l,N),dtype=complex)
    V = np.zeros(N,dtype=complex)
    for e in segments: #calcul des milieux p_e
        j = e[0]
        (x,y) = milieu(e,nodes)
        z = x + 1j*y
        r = np.abs(z)
        theta = np.angle(z)
        V[j] = p(a,r,theta,k)
    for i in range(l): #calcul de la matrice
    #contenant les integrales
        n1 = q[i]
        for e in segments:
            j=e[0]
            j2=e[1]
            M[i][j] = integrale2D(
            (lambda X,Y: G(n1,(X,Y),k)),
            nodes[j],nodes[j2],points,poids)
    U = np.matmul(M,V)
    return U
```

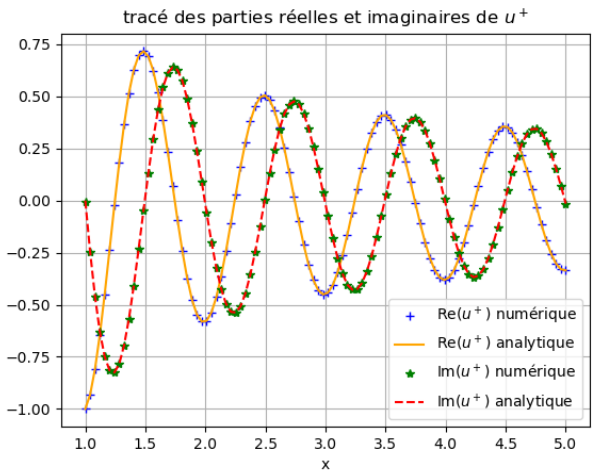

Figure 2: Tracé des parties réelles et imaginaires de u^+ pour $k = 2\pi$, $N = 500$, $a = 1.0$, $N_{\text{xi}} = 100$ et $nb_{\text{Gauss}} = 2$

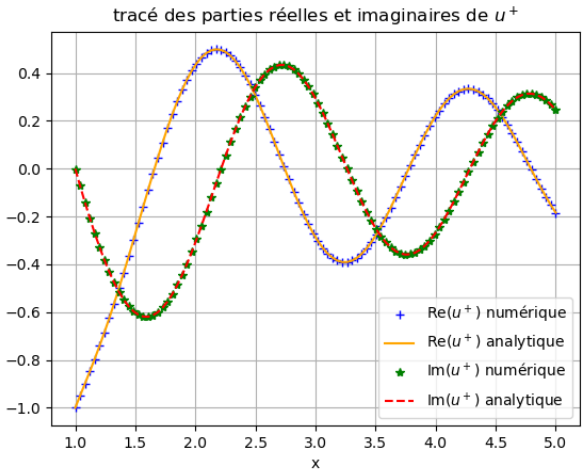

Figure 3: Tracé des parties réelles et imaginaires de u^+ pour $k = \pi$, $N = 200$, $a = 1.0$, $N_{\text{xi}} = 100$ et $nb_{\text{Gauss}} = 2$

Après avoir lancé le code et utiliser le module `matplotlib`, on obtient les graphes des figures 2 et 3. La première correspond à la fonction u^+ évaluée sur le segment horizontal allant de $(1, 0)$ à $(5, 0)$, et la seconde correspond à la fonction évaluée sur le segment vertical allant de $(0, 1)$ à $(0, 5)$.

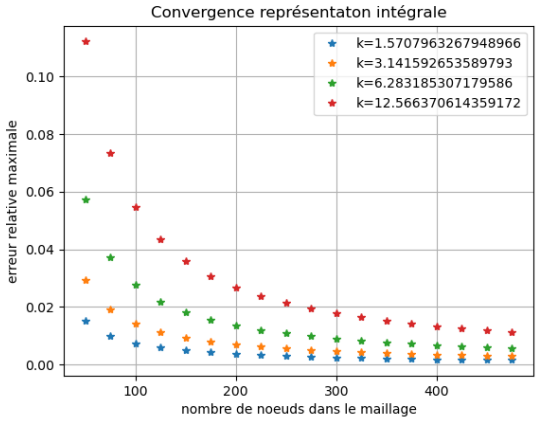

Figure 4: Erreur relative maximale de notre résolution intégrale pour k allant de $\pi/2$ à 4π, et pour un nombre de noeuds variant de 50 à 500

On souhaite maintenant vérifier la convergence de notre code, ainsi que l'erreur commise vis-à-vis de la solution analytique. Nous allons donc calculer l'erreur relative maximale commise pour différents nombres de noeuds dans notre maillage, ainsi que pour différentes fréquences (cf figure 4). On remarque que l'erreur relative diminue lorsque le maillage devient plus fin, et ce pour toutes les fréquences. On peut donc affirmer sans trop de risque que le code converge effectivement. Quant à la complexité en temps du code, en considérant que le calcul de la fonction p et que les calculs d'intégrales se font en temps constant, et que le produit matriciel entre le vecteur colonne des valeurs p_e et la matrice des intégrales se fait au plus en temps similaire au calcul de la dite matrice, alors la complexité est de l'ordre de

$$\mathcal{O}(N \times N_{\text{xi}}).$$

Conclusion

Pour conclure, lors de ce TP nous avons mis en place les premières étapes d'un solveur de type BEM (Boundary Element Method) pour l'équation de Helmholtz en deux dimensions. Après avoir généré un maillage du bord du domaine (le cercle Γ), nous avons implémenté le calcul explicite de la dérivée normale de $p = -u^+ - u^{\text{inc}}$ sur la frontière.

Ceci nous a permis d'ensuite approcher la représentation intégrale de la solution diffractée u^+ à l'aide d'une quadrature de Gauss–Legendre. Les comparaisons entre la solution numérique et la solution analytique ont montré une très bonne cohérence, et l'étude de convergence a confirmé que l'erreur relative diminue à mesure que le maillage est raffiné.

Ce travail a permis de nous familiariser avec les outils et les méthodes qui seront employées par la suite dans la création d'un vrai solveur BEM.