TP IPT n°2
Résolution numérique de l'équation de Laplace

Dans le cas d'un problème d'électrostatique, on définit le potentiel électrostatique $V(M)$ en un point M tel que $\vec{E}(M) = -\overrightarrow{\text{grad}}V(M)$. Dans une zone vide de charges ($\rho = 0$), l'équation de Maxwell-Gauss donne :

$$\Delta V(M) = 0$$

dite équation de Laplace. Cette équation joue un rôle important en mathématique et dans plusieurs domaines de la physique. On rencontre en effet fréquemment des situations dans lesquelles un champ scalaire $V(M)$ est recherché dans un domaine $\mathcal{D}$ avec les contraintes suivantes :

- $V(M)$ satisfait à l'équation de Laplace en tout point de $\mathcal{D}$,
- la valeur de $V(M)$ est fixée sur les limites de $\mathcal{D}$.

On parle dans ce cas du problème de Laplace. Parmi les propriétés remarquables, citons l'existence d'une unique solution pour tout problème de Laplace : ainsi, si une solution a été obtenue en formulant des hypothèses à priori, on peut affirmer qu'elle est bien la solution du problème.

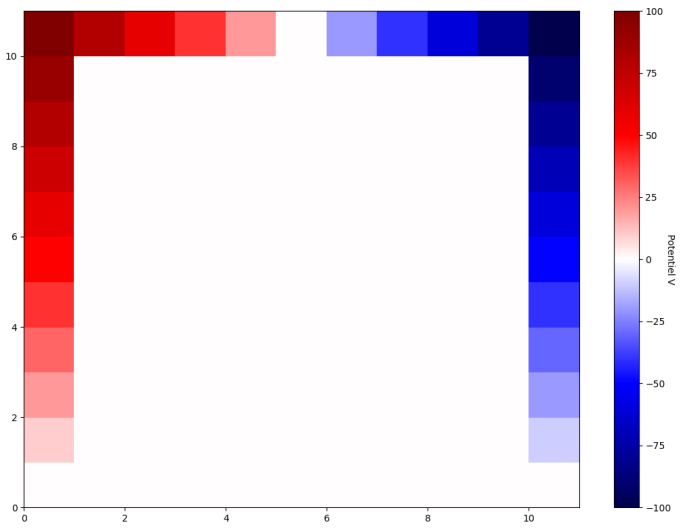
Le but de ce TP est d'utiliser une méthode numérique pour résoudre un problème de Laplace. En vue de faciliter la visualisation et de mettre en évidence le principe, on se place en coordonnées cartésiennes (O, x, y) de dimension 2.

1 - Discrétisation spatiale

La première étape de la résolution numérique est la discrétisation régulière du domaine d'étude en N cellules unitaires selon chaque dimension. Ainsi, le potentiel $V(M) = V(x, y)$ est décrit dans un tableau N par N, chaque cellule étant caractérisée par la valeur discrétisée $V[i, j]$. On propose les conditions aux limites suivantes :

- le potentiel est nul pour toutes cellules d'abscisse $i = 0$,
- il croît de manière affine, de 0 à 100 le long de la bordure verticale $j = 0$ et entre 0 et -100 le long de la bordure verticale $j = N - 1$,
- il évolue de façon affine entre 100 et -100 sur la bordure horizontale supérieure $i = N - 1$.

Ces conditions limites sont résumées sur la figure ci-contre (maillage 11×11).



⚛ À partir de 4 développements de Taylor de $V[i, j]$, déterminer une approximation du Laplacien de $V[i, j]$. On rappelle que notre équation est discrétisée sur une grille unitaire, on prendra donc un pas spatial $h = 1$ dans les deux directions x et y.

2 - Résolution numérique itérative

Pour déterminer l'ensemble des valeurs du potentiel, on part d'une distribution quelconque de celui-ci, c'est-à-dire d'un tableau qui ne correspond pas à l'équation de Laplace, et on procède par itérations. Étape par étape, on fait alors évoluer les valeurs contenues dans les cellules, en veillant à maintenir les conditions aux limites inchangées. Un moyen simple consiste à déterminer chaque nouvelle distribution du potentiel $V'[i,j]$ à partir de la ditribution courante $V[i,j]$ selon l'équation récurrente :

$$V'[i,j] = \frac{1}{4}\left(V[i-1,j] + V[i+1,j] + V[i,j-1] + V[i,j+1]\right)$$

Cette mise à jour ne concerne pas les cellules situées sur les bords qui doivent respecter les conditions limites. On note que l'évolution cesse lorsque la distribution du potentiel respecte l'équation de Laplace, car alors $V'[i,j] = V[i,j]$. Le problème de la convergence n'est pas examiné ici : on se contente d'affirmer que l'évolution temporelle (étape par étape) revient à faire intervenir un phénomène de diffusion. Du point de vue algorithmique, on stocke le potentiel dans un tableau `array` de flottants de dimension $N \times N$ noté `pot`. On définit également les fonctions :

- `inittab()` d'initialisation du tableau de valeurs respectant les conditions aux limites ;
- `itera(tab)` d'itération qui, recevant en argument une distribution de potentiel, renvoie une nouvelle distribution dans laquelle les cellules ont évolués selon l'équation récurrente ;
- `compare(tab1,tab2)` de comparaison qui renvoie le plus grand écart constaté (en valeur absolue) entre les cellules correspondantes de deux tableaux ;
- `afficher(pot)` qui permet de tracer en couleur l'évolution spatiale du champ `pot`.

⚛ Télécharger le code fourni en ligne et vérifiez que vous comprenez l'implémentation de la fonction `inittab()`. Lancez le script, vous devez obtenir un affichage identique à celui de la figure ci-dessus.

⚛ Codez la fonction `itera(tab)`. Cette fonction doit renvoyer un nouveau tableau après avoir modifié toutes les cellules grâce à la relation de récurrence. Attention à ne pas modifier le tableau d'entrée, mais à bien créer un nouveau tableau respectant les conditions limites.

⚛ Codez la fonction `compare(tab1,tab2)`. On rappelle que les opérations classiques (y compris la valeur absolue `abs()`) se font termes à termes entre deux arrays.

⚛ En quelques lignes de codes, implémentez la résolution complète du problème de Laplace. On continuera à itérer tant que l'écart maximal entre deux itérations est supérieur au critère d'arrêt `eps` (critère de stagnation). On affichera la solution finale ainsi que le nombre d'itérations effectuées.

Le procédé itératif utilisé ici est simple mais n'assure pas une convergence rapide. On propose d'exprimer ce problème sous la forme d'un système linéaire qu'on pourra ensuite résoudre de manière directe.

3 - Résolution numérique directe

En chaque élément du bord du domaine, les conditions aux limites nous donnent une équation, soit $4(N-1)$ équations. Chaque élément central donne également une équation (équation de récurrence), soit $(N-2)^2$ équations. Il y a donc au total $(N-2)^2 + 4(N-1) = N^2$ équations à résoudre, pour autant d'inconnues. Il faudrait montrer que les équations associées sont linéairement indépendantes pour s'assurer que la résolution est possible (on suppose que c'est le cas). On a donc un système de N^2 équations à N^2 inconnues à résoudre. On implémente le système associé sous la forme :

$$KV = B$$

où K est une matrice de taille $N^2 \times N^2$, B un vecteur de longueur N^2 représentant le second membre du problème linéaire (les conditions limites) et V un vecteur de taille N^2 qui représente le potentiel solution de notre problème. Pour mettre en place ce système d'équation, il faut être capable de décrire notre champ solution bidimentionnel de taille $N \times N$ sous la forme d'un vecteur de longueur N^2. On doit pour cela créer un "mapping" entre un champ solution `pot` et le vecteur solution V. On propose d'utiliser la numérotation globale illustrée ci-dessous :

⋮	⋮	⋮	⋮	$3N+3$
2	2	$N+2$	$2N+2$	$\cdots$
1	1	$N+1$	$2N+1$	$\cdots$
0	0	N	$2N$	$\cdots$
	0	1	2	$\cdots$

Dans cet exemple, la valeur du potentiel `pot[1,2]` correspondra à la composante `V[2N+1]`.

- Implémenter une fonction `indice(i,j)` qui associe à la position [i,j] dans le plan bidimensionnel sa coordonnée globale. La fonction inverse (`reconstruction(V)`) permettant de retrouver un champ 2D à partir d'un vecteur en coordonnée globales vous est donnée.

Les conditions aux limites sont interprétées comme des équations qui imposent simplement une valeur à un élément. Le script permettant de fixer les conditionss limites à gauche et à droite du domaine est déjà écrit.

- Sur le même principe, implémenter une boucle `for` pour mettre en place les conditions aux limites en haut et en bas du domaine.

Pour chaque élément central, on a une équation associée à la discrétisation de $\Delta V[i,j] = 0$ exprimée plus haut :

$$\frac{1}{4}\left(V[i-1,j] + V[i+1,j] + V[i,j-1] + V[i,j+1]\right) - V[i,j] = 0$$

- Remplissez les $(N-2)^2$ lignes "intérieures" de la matrice K pour résoudre ces équations.

La résolution du système linéaire peut se faire en utilisant le solveur linéaire de la librairie numpy avec : `V = np.linalg.solve(K,B)`.

- Résolvez le système linéaire construit précédemment pour obtenir la solution du problème.
- Vérifier que la résolution directe est bien plus efficace que la méthode itérative en travaillant par exemple sur un système de taille 101×101.
- S'il vous reste du temps, implémentez votre propre solveur en utilisant la méthode du pivot de Gauss.