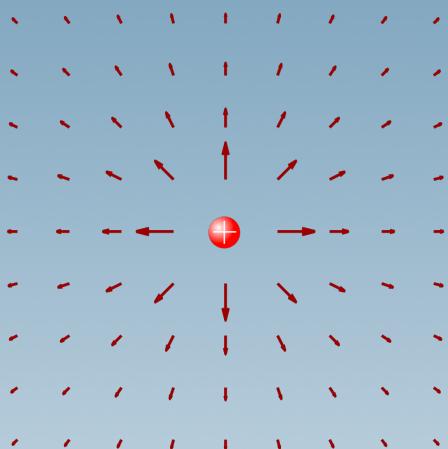


# Calcul de champs électrostatiques par la méthode des différences finies

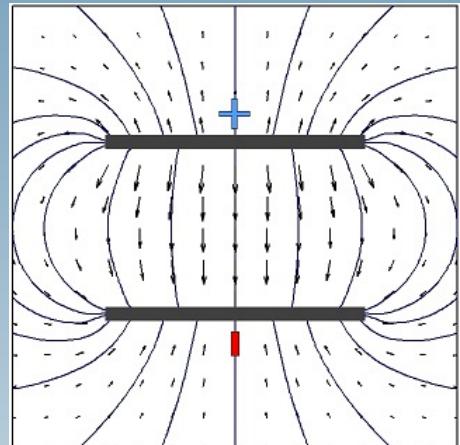
L3 EEA  
Projet - LU3EE103

Programmation et Méthodes Numériques en C

Novembre 2021 - Janvier 2022



**Zhifan SONG**  
**Melina HASNAOUI**  
**Emir TAS**  
**Yuankun FENG**



Professeur encadrant : Massimiliano CASALETTI



# SOMMAIRE

- 1) Présentation et théorie de base**
- 2) Description détaillée des méthodes numériques utilisées**
- 3) Implémentation des algorithmes**
  - \* Structuration des données
  - \* Algorithmes
- 4) Validation de chaque méthode implémentée**
  - \* Champ électrique dans le vide
  - \* Champ électrique autour d'un condensateur plan
- 5) Conclusion**
- 7) Bibliographie**

# PRÉSENTATION

Dans le cadre de l'UE de programmation et méthodes numériques en C, notre équipe est amené à réalisé un logiciel développé en langage C permettant de calculer dans un plan en deux dimensions de taille fixe, la distribution des potentiels électriques et le champ électrostatique associé générés par des distributions de charge ou présence d'un condensateur. L'observation de la présence de charges et des lignes de champs électriques se feront sur Matlab, mais les résultats sont calculés par notre programme écrit en langage C.

Tout d'abord, afin de bien assimiler le thème du projet, il faut poser les équations aux dérivées partielles du problème électrostatique.

On sait que le champ électrique vérifie les équations suivantes :

$$E = -\nabla V \quad \text{div} E = \frac{\rho}{\epsilon_0} \Rightarrow -\text{div}(\nabla V) = -\Delta V = \frac{\rho}{\epsilon_0}$$

Lorsqu'on considère des charges dans le vide, on combine ces deux équations de manière à obtenir : l'équation de Poisson-Laplace aux dérivées partielles :

$$\frac{\partial^2 V(x, y)}{\partial x^2} + \frac{\partial^2 V(x, y)}{\partial y^2} = -\frac{\rho(x, y)}{\epsilon_0}$$

## MÉTHODES NUMÉRIQUES UTILISÉES

Pour pouvoir résoudre cette équation aux dérivées partielles, nous devons utiliser la méthode numérique des différences finies.

Premièrement, on applique le développement de Taylor :

$$f(t_0 + \Delta t) = f(t_0) + f'(t_0)\Delta t + \frac{f''(t_0)}{2}\Delta t^2 + O(\Delta t^3) \quad (A)$$

$$f(t_0 - \Delta t) = f(t_0) - f'(t_0)\Delta t + \frac{f''(t_0)}{2}\Delta t^2 + O(\Delta t^3) \quad (B)$$

$$(A) + (B) \rightarrow f''(t_0) = \frac{f(t_0 + \Delta t) - 2f(t_0) + f(t_0 - \Delta t)}{\Delta t^2} + O(\Delta t^2)$$

Ce moyen va nous permettre de discréteriser l'espace, en remplaçant terme à terme, les correspondances entre la méthode numérique du **développement de Taylor** et notre équation initiale.

Tout d'abord, nous allons appliquer ces méthodes sur le principe de notre problème. À partir du développement de Taylor précédent, nous allons considérer que le pas  $h = \Delta_x = \Delta_y = \Delta$  et on a donc :

$$\frac{V_{x+\Delta,y} - 2V_{x,y} + V_{x-\Delta,y}}{\Delta_x^2} + \frac{V_{x,y+\Delta} - 2V_{x,y} + V_{x,y-\Delta}}{\Delta_y^2} = -\frac{\rho_{x,y}}{\epsilon_0}$$

$$\text{ce qui nous donne : } V_{x+\Delta,y} + V_{x-\Delta,y} + V_{x,y+\Delta} + V_{x,y-\Delta} - 4V_{x,y} = -\Delta^2 \frac{\rho_{x,y}}{\epsilon_0}$$

Sachant que les conditions aux limites de Dirichlet nous indiquent que les potentiels valent 0 pour chaque côté. On complète donc notre matrice pour chaque point du domaine  $\Omega$ , avec **la méthode de Jacobi** qui permet de calculer le potentiel aux points précédents.

$$Jacobi : V_{i,j} = \frac{1}{4}[V_{i-1,j} + V_{i+1,j} + V_{i,j-1} + V_{i,j+1} + h^2 \frac{\rho_{i,j}}{\epsilon_0}]$$

Cependant après ce calcul nous avons des termes, qui sont nos potentiels aux points courants qui n'ont pas pu être calculés. De ce fait, une méthode complémentaire à Jacobi va intervenir pour ce calcul, il s'agit donc de **la méthode de Gauss-Seidel** qui consiste à reprendre le même algorithme que Jacobi, pour lequel la matrice des potentiels est balayé selon les indices i et j croissants.

Donc lorsqu'on calcul  $V_{k+1}(i, j)$ , on aura déjà obtenu  $V_{k+1}(i-1, j)$  et  $V_{k+1}(i, j-1)$  grâce à la méthode précédente.

$$Gauss - Seidel : \forall (i, j) \in \Omega, V_{k+1}(i, j) = \frac{V_k(i+1, j) + V_k(i-1, j) + V_k(i, j-1) + V_k(i, j+1)}{4} + \frac{h^2 \rho(i, j)}{4\epsilon_0}$$

Nous pourrons utiliser une autre méthode: sur-relaxation successive qui consiste à faire la moyenne pondérée des deux méthodes vues précédemment (Jacobi et Gauss-Seidel), dont le résultat va converger beaucoup plus vite par rapport aux 2 méthodes précédentes.

$$Sur - relaxation : V_{k+1}(i, j) = (1 - \omega) \underbrace{V_k(i, j)}_{Jacobi} + \omega \cdot \underbrace{\frac{1}{4}[V_{i-1,j} + V_{i+1,j} + V_{i,j-1} + V_{i,j+1} + \Delta^2 \frac{\rho_{i,j}}{\epsilon_0}]}_{Gauss Seidel}$$

Effectivement, en informatique, on ne peut calculer qu'un nombre fini de matrice et donc il faut imposer une condition d'arrêt à notre algorithme.

On admettra que lorsque  $\frac{\|V_{k+1} - V_k\|_\infty}{\|V_{k+1}\|_\infty} < \epsilon$ , l'algorithme sortira de la boucle en considérant que la valeur souhaitée a été atteinte.

Pour calculer la valeur optimale de  $\omega$ , on va faire le calcul suivant :

$$\omega = \frac{8 - \sqrt{64 - 16t^2}}{t^2} \quad \text{avec} \quad t = \cos\left(\frac{\pi}{N_x}\right) + \cos\left(\frac{\pi}{N_y}\right)$$

$N_x$  et  $N_y$  sont les dimensions de notre domaine  $\Omega$ .

Par la suite, après avoir obtenu le potentiel  $V$  à chaque point, il faut calculer le champ électrique  $E = -\text{grad}(V) = -\nabla V$ .

$$E_x = -\frac{\partial V}{\partial x} = -\left(\frac{(V_{i+1,j} - V_{i,j})}{\Delta}\right)$$

$$E_y = -\frac{\partial V}{\partial y} = -\left(\frac{(V_{i,j+1} - V_{i,j})}{\Delta}\right)$$

$$E = \sqrt{E_x^2 + E_y^2}$$

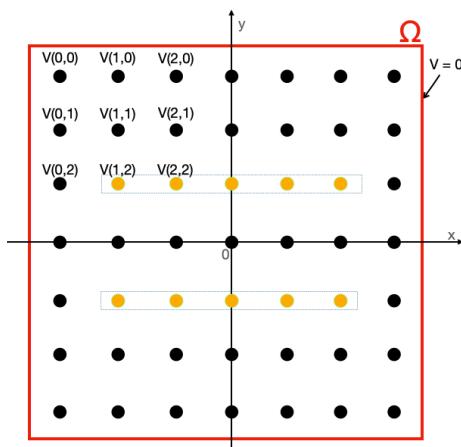


Figure 1. Exemple d'une grille de la méthode des différences finies

# IMPLÉMENTATION EN LANGAGE C

## Structuration de données

Nous allons maintenant implémenter les algorithmes permettant de résoudre la problématique. Pour cela, nous allons d'abord introduire la structuration des données. Seule la partie 2 concernant la distribution des potentiels et le champ électrostatique dans et autour d'un condensateur plan est présentée ci-dessous. Pour étudier l'influence des charges dans la partie 1, il suffit de supprimer les lignes de code mettant le condensateur et d'ajouter des valeurs dans la matrice de charges.

Les variables:

**dimX** et **dimY** : les dimensions;

**tension\_plaque\_up** : la valeur de tension de la plaque en haut du condensateur, et **tension\_plaque\_down** pour celle en bas;

**longueur\_plaque**: la longueur des deux plaques du condensateur (nombres de points);

**position\_plaque**: où on met le condensateur par rapport au point milieu de l'axe X;

**h**: le nombre de pas;

**k, kgr, kgs**: trois compteurs permettant de compter les nombres d'itérations pour les trois méthodes numériques utilisées (Jacobi, Gauss-Seidel, Sur-relaxation successive);

**lp**: la moitié de la longueur de la plaque; **mpx** et **mpy**: le point milieu des axes X et Y; **pp1** et **pp2** : la position des plaques en bas et en haut par rapport à l'axe X;

**eps0** et **epsr**: permittivité du vide et permittivité relative choisie;

### La bibliothèque 'système.h'

Pour les potentiels et les charges, on déclare une structure '**System**' qui contient les dimensions **Nx, Ny**, une matrice de charges **charge** et une matrice de potentiels **tension**. On écrit donc une fonction **System creerSystem ( int dimX ,int dimY )** qui nous permet d'initialiser et renvoie une structure de type **System** qui crée et initialise à zéro une matrice de potentiels ainsi qu'une matrice de charges. Cette fonction permet aussi d'initialiser tous les points sur les bords de l'espace à zéro (conditions de limite). La fonction **void freeSystem ( System \*s )** correspondante a donc l'objectif de vider les deux matrices créées par l'allocation dynamique. Les autres fonctions sont celles que l'on écrit communément: **void writefileMatrix (char \*fName, double\*\* v, int dimX, int dimY)** pour écrire une matrice dans un fichier texte, **double\*\* creerMatrice (int dimX, int dimY)** pour allouer dynamiquement une matrice, **void freeMatrice (double \*\*\*mat, int dimX)** pour vider une matrice. On y ajoute également deux fonctions suivantes: **double\*\* subtractMatrix ( double\*\* mat1, double\*\*mat2, int dimX, int dimY)** permettant de soustraire deux matrices et **double norm (double\*\* v, int dimX, int dimY)** qui calcule la norme d'une matrice. Ces deux fonctions nous permettront de calculer à chaque itération l'écart pour comparer avec le critère d'arrêt afin de sortir de la boucle 'while'. On met ces fonctions dans une bibliothèque 'système.h'.

### La bibliothèque 'methodes.h'

On écrit 3 fonctions **System MethodeJacobi / MethodeGaussSeidel / MethodeRelaxation** (respectivement) (**int dimX, int dimY, int length\_plate, int position\_plate, double tension\_plaque\_up, double tension\_plaque\_down, double h, int \*k**) dont le contenu est les 3 algorithmes que l'on détaillera après. On les mets dans la

bibliothèque 'méthodes.h'.

### La bibliothèque 'Champ.h'

La dernière bibliothèque, 'Champ.h' contient une structure de champ électrique **ChampE**, qui contient les dimensions **Nx, Ny** ainsi que deux matrices **Ex et Ey** qui sont calculées à partir du gradient des potentiels. Dans cette bibliothèque, on implémente une fonction **ChampE creerChamp( int dimX, int dimY )** qui initialise un champ électrique à zéros pour tous les points, une fonction **void freeChamp( ChampE \*e )** correspondante pour vider les matrices allouées. La fonction **ChampE gradientV ( int dimX, int dimY, System v, double h )** permet de calculer le gradient de V pour ensuite être envoyée dans la fonction **double\*\* amplitudeE( ChampE champ )** qui permet de calculer les amplitudes du champ électrique. Avec toutes ces fonctions, on peut donc calculer la capacité du condensateur caractérisé par nous par les méthodes numériques pour comparer avec la valeur réelle.

### Le 'main.c'

On crée ensuite trois variables de type **System** v1,v2,v3 respectivement qui prennent en entrée les 3 systèmes de potentiels calculés par les 3 méthodes numériques. De même, on crée 3 variables de type **ChampE** qui prennent en entrée les 3 champs électriques pour les 3 méthodes choisies calculées. 3 matrices E1,E2,E3 sont ensuite créées pour prendre les amplitudes des 3 champs électriques. On pourra donc ensuite calculer la valeur de la capacité par méthode numérique et calculer la valeur réelle pour les comparer.

On utilise la fonction **writefileMatrix** pour exporter les systèmes de potentiels et de champs électriques vers les fichiers textes, pour que l'on puisse ensuite les imposer dans Matlab pour visualiser la distribution de potentiels ainsi que les lignes de champs électrique.

Au final, on utilise la fonction **freeSystem** et **freeChamp** pour vider toutes les matrices allouées après avoir écrit nos données dans les fichiers textes.

## Algorithmes

Dans cette partie, on s'intéresse uniquement aux trois fonctions écrites dans la bibliothèque 'méthodes.h'

Prenons un exemple de dimension 102x102 (dont 4 points de coins et 100x100 à l'intérieur).

Pour chaque méthode, on utilise une boucle 'while' dans laquelle on met deux boucles 'for' afin de faire les calculs matriciels. Les variables i et j varient de 1 à Nx-2 et Ny-2 car on exclut les potentiels de coins.

Il faut imposer la tension des deux plaques du condensateur à chaque itération pour figer le condensateur.

Avant chaque itération on utilise une matrice Vn/VnGs/VnR respectivement pour mémoriser la dernière matrice des potentiels calculés, il faut faire attention à recréer la matrice v1/v2/v3.tension en utilisant la fonction **creerMatrice** pour l'initialiser car on est en train de coder en C au lieu de Matlab, sinon les valeurs

sont erronées.

### Algorithme Méthode de Jacobi:

**Tant que** ( e > critère d'arrêt)

```
Vn = V // mémorisation de la matrice calculée  
V = creerMatrice ( dimX, dimY ) // initialiser la matrice  
(*k) ++ // incrémentation du compteur  
Pour i = 1 à dimX -2 faire  
    Pour j = 1 à dimY -2 faire  
        Pour (i et j) ∈ plaque faire  
            V <- tension_plaque_up et tension_plaque_down // imposer condo  
            fin pour  
            V [ i ][ j ] = 0.25 * ( Vn[ i+1 ][ j ] + Vn[ i-1 ][ j ] + Vn[ i ][ j-1 ] + Vn[ i ][ j+1 ] +  
            h*h*v1.charge[ i ][ j ]/eps0 )  
            fin pour  
        fin pour
```

e = norme de la matrice V // calcul norme matrice

**Fin**

**Retourner** V // renvoyer la matrice

La ligne en violet nous permet d'itérer avec la méthode de Jacobi qui n'utilise que la dernière matrice calculée pour faire la mise à jour. Cette méthode demande beaucoup plus d'itérations pour converger au critère d'arrêt demandé.

### Algorithme Méthode de Gauss-Seidel

```
V [ i ][ j ] = 0.25 * ( Vn[ i+1 ][ j ] + V[ i-1 ][ j ] + V[ i ][ j-1 ] + Vn[ i ][ j+1 ] +  
h*h*v1.charge[ i ][ j ]/eps0 )
```

Pour la méthode de Gauss-Seidel, seule la ligne en violet change, comme expliqué précédemment, pour les V(i-1 ou j-1), on utilise la matrice (k+1) calculée donc la matrice V parce que l'on possède déjà ces données. Cela accélère évidemment l'algorithme et il va donc converger plus vite que la méthode de Jacobi.

### Algorithme Méthode Sur-Relaxation Successive

La première étape consiste à définir le résiduel R(i,j), que V ne satisfasse pas l'équation de départ :

$$R(i, j) = \frac{1}{4}[V(i - 1, j) + V(i + 1, j) + V(i, j - 1) + V(i, j + 1) + h^2 \frac{\rho(i, j)}{\epsilon_0}] - V(i, j)$$

L'étape suivante consiste à boucler sur chaque échantillon de V (i, j) et à ajouter un facteur de correction défini par R(i, j). On multiplie R par un facteur de relaxation ω, ce qui nous permet d'avoir une convergence plus rapide. Ce processus est ensuite répété sur de nombreuses itérations jusqu'à ce que le résidu est en dessous d'une valeur d'erreur inférieure au critère d'arrêt. Pour la kième itération de la boucle, on a donc

$$V^{k+1}(i, j) = V^k(i, j) + \omega R^k(i, j)$$

On ajoute la variable omega = ... (formule) et on modifie simplement la ligne en violet:

```
V [ i ][ j ] = Vn [ i ][ j ] + omega * ( 0.25 * ( Vn[ i+1 ][ j ] + V[ i-1 ][ j ] + V[ i ][ j-1 ] +  
Vn[ i ][ j+1 ] + h*h*v1.charge[ i ][ j ]/eps0 ) - Vn [ i ][ j ] )
```

# VALIDATION DES MÉTHODES

## Champ électrique dans le vide

On définit un domaine  $\Omega$  de taille finie, par exemple  $102 \times 102$  (dont  $100 \times 100$  pour les points à l'intérieur du domaine). On crée et initialise un système, on met ensuite une/des charge(s) dans le domaine. Par exemple, on met une charge de valeur  $2e-9$  à la position  $[10,10]$ , on utilise la méthode de Gauss Seidel et la méthode de Sur-Relaxation Successive pour calculer la matrice de potentiel. On les importe dans Matlab pour les visualiser:

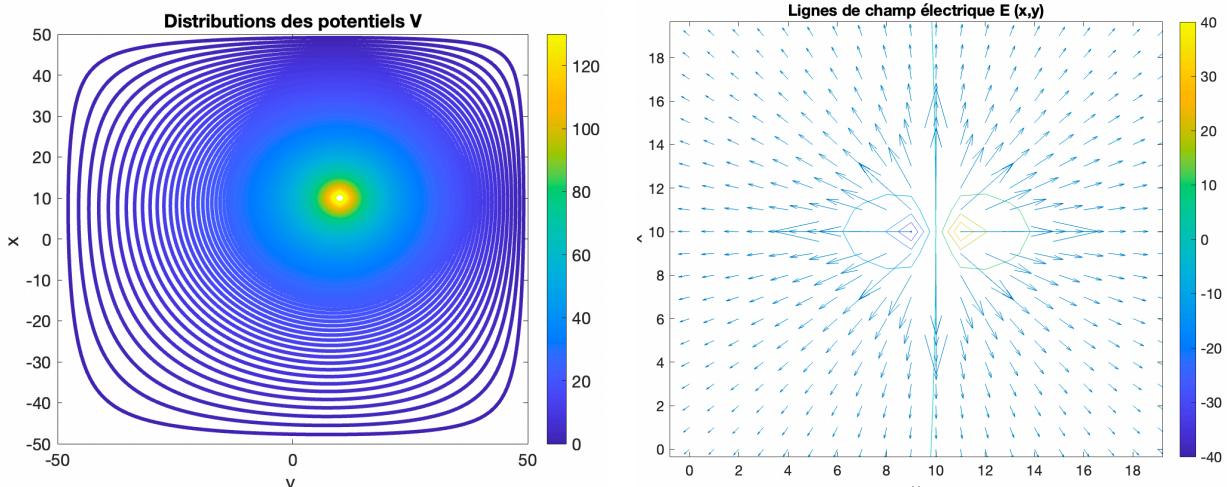


Figure 2. Distribution des potentiels et lignes de champs électriques pour une charge

On peut par exemple calculer le potentiel du point  $[12,12]$  en utilisant la formule:  $kQ/r$  et la comparer avec celle calculée par les méthodes numériques:

La valeur réelle: 63.551572

La valeur MN: 60.798114

Nombre d'iterations méthode Gauss-Seidel: 1040

Nombre d'iterations méthode Sur-Relaxation Successive : 85

On remarque que la valeur est proche de la valeur réelle et que la méthode de sur-relaxation successive converge largement plus vite que la méthode de Gauss Seidel. On constate aussi que la forme des lignes de champ correspond bien à celle créée par une charge. On peut aussi ajouter plus de charges (positives et négatives) dans d'autres endroits:

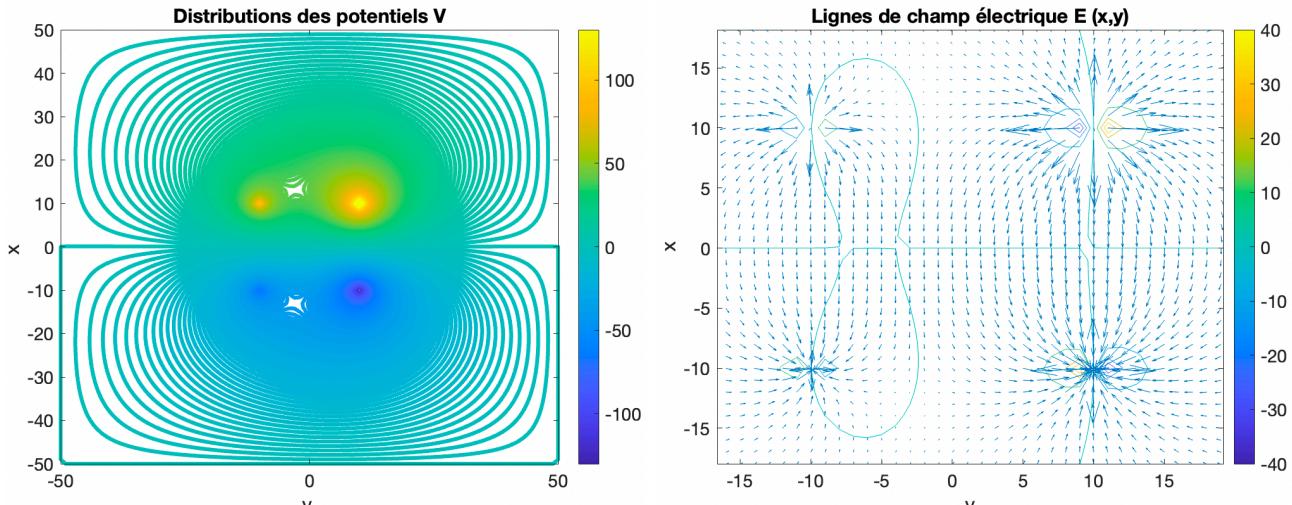


Figure 3. Distribution des potentiels et lignes de champs électriques - plusieurs charges

## Champ électrique autour d'un condensateur plan

On étudie par la suite le champ électrique dans et autour d'un condensateur plan sans charge. Prenons par exemple toujours un domaine de 102x102, un pas de 1, et deux plaques de longueur 51 points qui est à 25 points des frontières avec  $\pm 100$  V en haut/bas.

**Nombre d'iterations méthode Jacobi: 1238**  
**Nombre d'iterations méthode Gauss-Seidel: 780**  
**Nombre d'iterations méthode Sucessive over-relaxation : 110**

On exécute le programme et on remarque que la méthode de sur-relaxation successive n'a besoin que de 110 itérations pour converger vers la même erreur acceptable, cet algorithme est donc sans doute meilleur.

On importe donc les fichiers 'Vrelax.txt', 'Ex.txt', 'Ey.txt' et 'E.txt' dans Matlab pour visualisation:

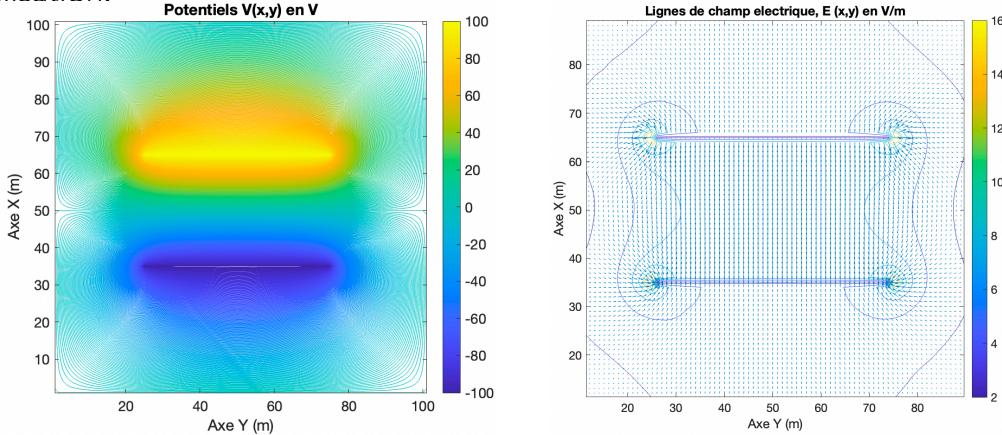


Figure 4. Distribution des potentiels et lignes de champs électriques - condensateur plan ( $\epsilon_r = 1, L = 51$ )

On augmente la permittivité relative  $\epsilon_r$ , on constate que la distribution des potentiels ne change pas mais l'amplitude du champ électrique augmente. Si on change les caractéristiques du condensateur, par exemple si on augmente la longueur des plaques, la distribution des potentiels change, les lignes de niveaux les plus élevés se dilatent vers le centre et évidemment il y a moins de lignes de champ électrique verticales de la plaque du haut vers la plaque du bas.

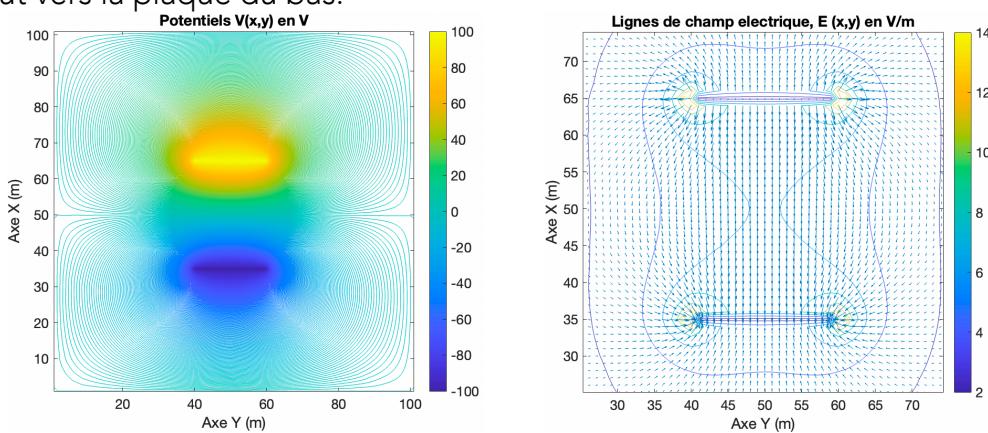


Figure 5. Distribution des potentiels et lignes de champs électriques - condensateur plan ( $\epsilon_r = 5, L = 21$ )

Enfin, on peut calculer la capacité du condensateur en utilisant la formule  $C = \epsilon \frac{\text{Longueur}}{\text{Hauteur}}$ . Pour la méthode numérique, on calcule l'énergie électrostatique

$$E_e = \iint \frac{1}{2} \epsilon E(x, y)^2 dx dy = \sum_i \sum_j \frac{1}{2} \epsilon E(i, j)^2 h^2$$

On a  $Ee = \frac{1}{2}C\Delta V^2 \Rightarrow C = \frac{2Ee}{\Delta V^2}$  ( $\Delta V$  correspond à la différence de potentiel entre 2 plaques du condensateur)

Pour  $\epsilon_r = 3$  et la longueur des plaques = 51, la distance entre deux plaque = 30 on a :

```
La valeur de la capacite calculee par la methode numerique :  
Cmn = 44.820404 pF  
La valeur reelle de la capacite:  
C = 45.156369 pF  
Program ended with exit code: 0
```

On constate que la valeur calculée par la méthode numérique est très proche de la valeur réelle du condensateur. On pourrait diminuer encore le critère d'arrêt pour avoir une valeur plus précise et encore plus proche de la valeur réelle.

## CONCLUSION

Nous avons implémenté trois algorithmes de méthodes numériques permettant de résoudre l'équation de Poisson. Nous avons pu étudier le principe de fonctionnement des trois algorithmes et les comparer. L'algorithme de sur-relaxation successive est la plus rapide pour trouver les résultats.

Nous pourrons améliorer le programme, par exemple en utilisant une liste pour les charges au lieu d'une matrice pour réduire l'impact de mémoire, étant donné que la matrice de charges est évidemment une matrice creuse.

## BIBLIOGRAPHIE

- [1] Solving the generalized Poisson equation using the Finite-Difference method (FDM). James R. Nagel, Department of Electrical and Computer Engineering, University of Utah, Salt Lake City, Utah.
- [2] Autour de l'équation de Poisson. CCINP 2017 PC - mai 2020.
- [3] Sadiku, Elements of Electromagnetics, 4th Edition, Oxford