LECOURTIER Frédérique 20 mars 2023

Suivi Stage PhiFEM : 06/02/2023 - 28/07/2023

Table des matières

1 Semaine 1 : 06/02/2023 - 10/02/2023

Résumé

Pendant cette première semaine, j'ai du me familiariser avec le code "phifem" écrit par Vincent Vigon et les codes de génération des données fournit par Killian. L'idée étant de comprendre comment générer les données avec FEniCS pour ensuite les faire apprendre par un FNO. Après la réunion du 07/02, il semblerait que le sujet du stage porte sur l'entrainement d'un FNO puis la correction/certification des prédictions.

1.1 Génération des données

Le code fournit par Killian ("Data_Generation_moving_ellipse_poisson") a pour but de faire varier la levelset. J'ai donc repris ce code afin de générer dans un premier temps les données solution d'un problème de Poisson avec condition de Dirichlet homogène ("DataGen_PhiFEM_f_gaussienne").

On considère Ω le cercle de rayon $\sqrt{2}/4$ et de centre $(0.5, 0.5)$ avec $\Phi(x,y) = -1/8 + (x-1/2)^2 + (y-1/2)^2$ et le domaine fictif $O = (0,1)^2$.

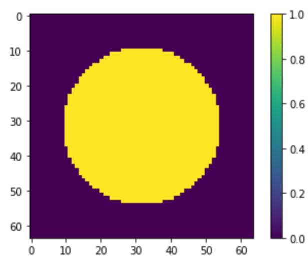
On souhaite résoudre

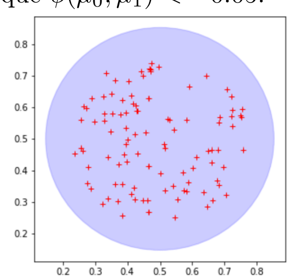
$$\begin{cases} -\Delta u &= f\,, \quad \text{dans } \Omega\,, \\ u &= 0\,, \quad \text{sur } \Gamma\,, \end{cases}$$

où

$$f(x,y) = \exp\left(-\frac{(x-\mu_0)^2 + (y-\mu_1)^2}{2\sigma^2}\right)\,,$$

avec $\sigma \sim \mathcal{U}([0.1, 0.6])$ et $\mu_0, \mu_1 \sim \mathcal{U}([0.5 - \sqrt{2}/4, 0.5 + \sqrt{2}/4])$ à condition que $\phi(\mu_0, \mu_1) < -0.05$.





La fonction *create_ data* renvoie le nombre donné de F et les paramètres associés choisis uniformément.

Formulation faible :

$$\int_{\Omega_h} \nabla(\bar{\phi}w)\nabla(\bar{\phi}v) - \int_{\partial\Omega_h} \frac{\partial}{\partial n}(\bar{\phi}w)\bar{\phi}v + G_h(w,v) = \int_{\Omega_h} f\bar{\phi}v + G_h^{rhs}(v)$$

avec

$$G_h(w,v) = \sigma h \sum_{E\in\mathcal{F}_h^\Gamma} \int_E \left[\frac{\partial}{\partial n}(\bar{\phi}w)\right]\left[\frac{\partial}{\partial n}(\bar{\phi}v)\right] + \sigma h^2 \sum_{T\in\mathcal{T}_h^\Gamma} \int_T \Delta(\bar{\phi}w)\Delta(\bar{\phi}v)$$

et

$$G_h^{rhs}(v) = -\sigma h^2 \sum_{T\in\mathcal{T}_h^\Gamma} \int_T f\Delta(\bar{\phi}v)$$

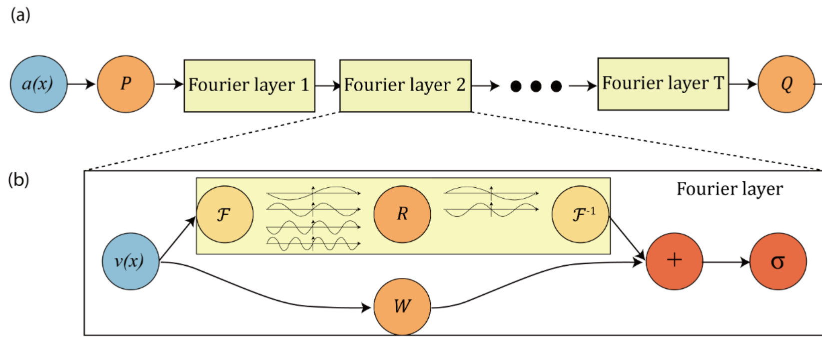
On utilise FEniCS (solveur d'EDP) pour résoudre le problème.

On va finalement stocker les résultats au format npy dans les fichiers "F.npy", "agentParams.npy" et "U.npy"

1.2 FNO

De la même manière que pour la génération des données, il a fallu reprendre le code de Vincent Vigon ("phifem") afin de l'adapter au problème considéré (phifem_f_gaussienne).

Une première étape fut donc la lecture d'article sur les FNO (Fourier Neural Operator). Voici un schéma descriptif de ce type de réseau de neurones.



(a) The full architecture of neural operator: start from input a. 1. Lift to a higher dimension channel space by a neural network P. 2. Apply four layers of integral operators and activation functions. 3. Project back to the target dimension by a neural network Q. Output u. **(b) Fourier layers**: Start from input v. On top: apply the Fourier transform $\mathcal{F}$; a linear transform R on the lower Fourier modes and filters out the higher modes; then apply the inverse Fourier transform $\mathcal{F}^{-1}$. On the bottom: apply a local linear transform W.

L'idée étant que le réseau nous retourne u.

Remarque. *ERREUR : il doit nous retourner w car on connait déjà la levelset et pour la correction, ça n'a pas de sens de faire* $\phi u = \phi^2 w$.

1.3 Correction

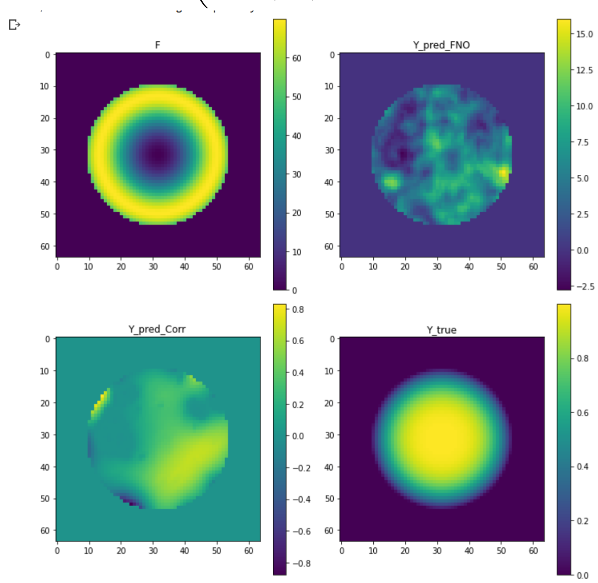
On veut en fait résoudre le même problème sauf que cette fois-ci, on définit une nouvelle levelset $\bar{\phi} = \phi u$ (où u est la sortie du FNO).

On résout alors le nouveau problème $z = \bar{\phi} C$:

$$\begin{cases} -\Delta z &= f\,, \quad \text{dans } \Omega\,, \\ z &= 0\,, \quad \text{sur } \Gamma\,, \end{cases}$$

Remarque. *L'idée étant d'appliquer la correction sur un millage plus grossier que le réseau. Ainsi FNO+Corr plus précis que Phifem classique et plus rapide.*

Les résultats obtenus (en prenant ici $u_{ex} = cos\left(\frac{\pi}{2} \times \left(\frac{4}{\sqrt{2}}\right)^2 \times \left((x-0.5)^2 + (y-0.5)^2\right)\right)$) ne sont pas bons :



Remarque. *ERREUR : C'est logique ! ! ! Le réseau n'a pas été entrainé pour ça ! (+ pb remarque d'avant.)*

2 Semaine 2 : 13/02/2023 - 17/02/2023

Résumé

(Les résultats en semaine 1 ont été obtenus en début de semaine.)

Après la semaine dernière, on s'est dit qu'une bonne idée serait de prendre une solution manufacturée (analytique) afin de pouvoir comparer les erreurs avec FEM classique, PhiFEM, le FNO et le FNO+correction. On a choisi de prendre u comme une gaussienne et ainsi le f associé. Attention, il a fallut normaliser les F pour le FNO. On a également eut l'idée d'utiliser une solution sur-raffinée (à la place d'une solution exacte) mais ça n'a pas encore été testé.

2.1 Génération de données

On considère toujours Ω le cercle de rayon $\sqrt{2}/4$ et de centre $(0.5, 0.5)$ avec $\Phi(x,y) = -1/8+(x-1/2)^2+(y-1/2)^2$ et le domaine fictif $O = (0,1)^2$ ("DataGen_PhiFEM_u_gaussienne").

On souhaite résoudre

$$\begin{cases} -\Delta u & = f\,, \quad \text{dans } \Omega\,, \\ u & = g\,, \quad \text{sur } \Gamma\,, \end{cases}$$

Notre solution analytique est

$$u_{ex}(x,y) = \exp\left(-\frac{(x-\mu_0)^2+(y-\mu_1)^2}{2\sigma^2}\right)\,,$$

avec $\sigma \sim \mathcal{U}([0.1, 0.6])$ et $\mu_0, \mu_1 \sim \mathcal{U}([-0.9, 0.9])$.

Ainsi

$$f(x,y) = \left(\frac{2\sigma^2-(x-\mu_0)^2-(y-\mu_1)^2}{\sigma^4}\right) * \exp\left(-\frac{(x-\mu_0)^2+(y-\mu_1)^2}{2\sigma^2}\right)$$

et

$$g(x,y) = u_{ex}(x,y)$$

Pour l'apprentissage du FNO, on va normaliser f par $\max_f ||f||_{L^2(\Omega_h)}$

Formulation faible :

On utiliser une méthode directe pour inclure les conditions de Dirichlet homogène :

$$\int_{\Omega_h} \nabla(\bar{\phi}w)\nabla(\bar{\phi}v) - \int_{\partial\Omega_h} \frac{\partial}{\partial n}(\bar{\phi}w)\bar{\phi}v + G_h(w,v) = \int_{\Omega_h} f\bar{\phi}v - \left(\int_{\Omega_h} \nabla(g)\nabla(\bar{\phi}v) - \int_{\partial\Omega_h} \frac{\partial g}{\partial n}\bar{\phi}v\right) + G_h^{rhs}(v)$$

avec

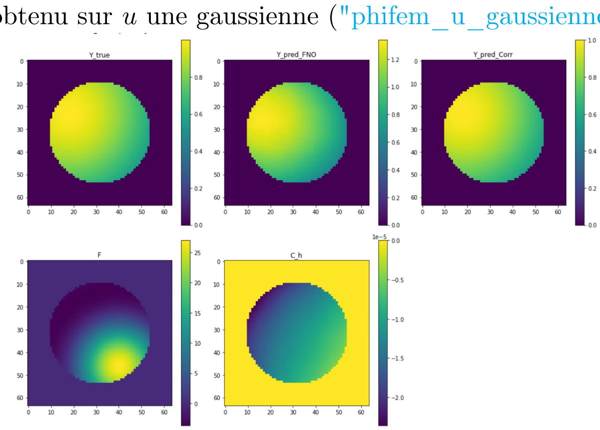
$$G_h(w,v) = \sigma h \sum_{E\in\mathcal{F}_h^\Gamma} \int_E \left[\frac{\partial}{\partial n}(\bar{\phi}w)\right]\left[\frac{\partial}{\partial n}(\bar{\phi}v)\right] + \sigma h^2 \sum_{T\in\mathcal{T}_h^\Gamma} \int_T \Delta(\bar{\phi}w)\Delta(\bar{\phi}v)$$

et

$$G_h^{rhs}(v) = -\sigma h^2 \sum_{T\in\mathcal{T}_h^\Gamma} \int_T f\Delta(\bar{\phi}v) - \sigma h \sum_{E\in\mathcal{F}_h^\Gamma} \int_E \left[\frac{\partial g}{\partial n}\right]\left[\frac{\partial}{\partial n}(\bar{\phi}v)\right] - \sigma h^2 \sum_{T\in\mathcal{T}_h^\Gamma} \int_T \Delta(g)\Delta(\bar{\phi}v)$$

2.2 Résultat

Voici un exemple de résultats obtenu sur u une gaussienne ("phifem_u_gaussienne").



Conclusion

C'est en fin de semaine qu'on s'est rendue compte du problème du FNO : qu'il faut retourner w et pas u. Les modifications ont été faites mais les résultats ne semblent toujours pas convaincant (résultats supprimés malencontreusement).

En fait, on a remarqué que prendre $g = u_{ex}$ sur Ω entier posait problème pour la correction, car on obtient $w = 0$ et donc le problème de correction n'a plus de sens. Une solution à celà a été proposé par Emmanuel : il faudrait prendre $g = u_{ex}$ sur Γ et étendre la solution (par exemple en utilisant un nouveau réseau de neurones qui nous fournirait une solution lisse). Mais pour l'instant, on va simplement choisir une autre solution analytique, où on n'est pas obligé de prendre $g = u_{ex}$ sur tout le domaine.

On a également eut l'idée de se ramener à un problème homogène (passage du problème en f au problème en $\tilde{f} = f + \Delta g$)

Un autre problème constaté est qu'il faut utilisé le ϕ initial pour la génération des espaces $\mathcal{F}_h^\Gamma$ et $\mathcal{T}_h^\Gamma$.

3 Semaine 3 : 20/02/2023 - 24/02/2023

Résumé

Pour cette semaine, on considère une nouvelle solution analytique (solution trigonométrique : $\sin * \cos$). Avec cette nouvelle solution, on va pouvoir prendre g différent de u_{ex} sur Ω comme souhaité pendant la semaine précédente.

A défaut de pouvoir faire tourner les entrainements (car plus d'units dispo sur colab) et en attente d'une solution avec v100, on va s'intéresser uniquement au problème de correction où on prendra comme $\bar{\phi}$ notre u_{ex} ($-g$ si non homogène). En lui donnant comme nouvelle levelset notre solution exacte, C doit être très proche de 1 (à l'erreur machine).

3.1 Génération des données

On considère toujours Ω le cercle de rayon $\sqrt{2}/4$ et de centre $(0.5, 0.5)$ avec $\Phi(x,y) = -1/8+(x-1/2)^2+(y-1/2)^2$ et le domaine fictif $O = (0,1)^2$.

On souhaite résoudre

$$\begin{cases} -\Delta u &= f\,, \quad \text{dans } \Omega\,, \\ u &= g\,, \quad \text{sur } \Gamma\,, \end{cases}$$

Notre solution analytique est

$$u_{ex}(x,y) = \frac{1}{\sin\left(k_1 \frac{\pi}{2}\right)} \times \sin\left(k_1 \frac{\pi}{2} \left(\frac{4}{\sqrt{2}}\right)^2 \left((x-0.5)^2+(y-0.5)^2\right)\right) \times \cos(k_2(x^2+y^2))\,,$$

avec $k_1, k_2 \sim \mathcal{U}([0.1, 0.5])$.
Ainsi

$$g(x,y) = \cos(k_2(x^2+y^2))$$

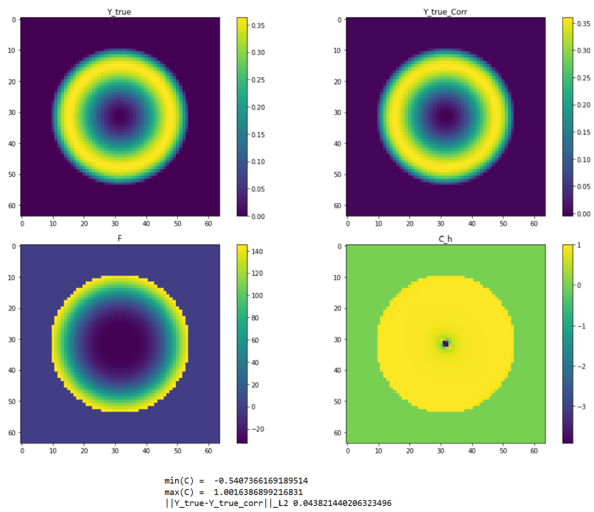
On considérera également la solution analytique au problème homogène :

$$u_{ex}(x,y) = \frac{1}{\sin\left(k_1 \frac{\pi}{2}\right)} \times \sin\left(k_1 \frac{\pi}{2} \left(\frac{4}{\sqrt{2}}\right)^2 \left((x-0.5)^2+(y-0.5)^2\right)\right) \times \cos\left(\frac{\pi}{2} \left(\frac{4}{\sqrt{2}}\right)^2 \left((x-0.5)^2+(y-0.5)^2\right)\right)\,,$$

Les formulation faibles sont les mêmes que précédemment.

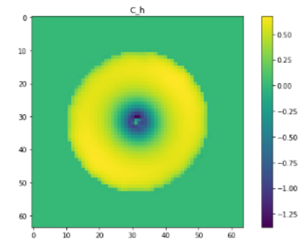
3.2 Correction

Dans un premier temps, on a pris notre nouvel levelset comme étant notre solution exacte sur une image 64*64 puis on interpoler cette levelset ("u_trigo_homogene"). En prenant $\phi = u_{ex}$, on espère obtenir C très proche de 1 (en augmentant le degré d'interpolation, on espère atteindre l'erreur machine). Les résultats obtenus n'étaient pas bon :

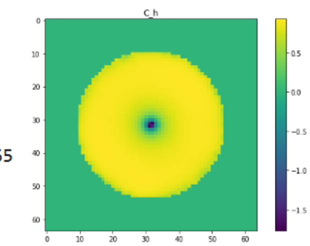


En fin de semaine, on s'est rendue compte que l'interpolation de ϕ posait problème pour la correction. Lorsque $\bar{\phi} = u_{ex}$, on obtient bien l'erreur machine mais en lui donnant notre levelset sous la forme d'une image nb_vert×nb_vert (ce qui est plus proche de ce que le réseau fournira), les résultats deviennent incohérents. On a donc tenter deux approches ("u_trigo_homogene_modif") : griddata (de scipy) et extrapolate (de FEniCS). L'option du extrapolate n'a pas fonctionné (nan ?). Pour l'autre option avec le griddata, on a constaté que interpoler avec tous les points de l'image était trop lourd, on a donc décidé de prendre pour chaque point (x,y) un certains nombres de points d'interpolations autour de lui. Voici les résultats obtenus avec différents nb_vert et différents nombres de points d'interpolation :

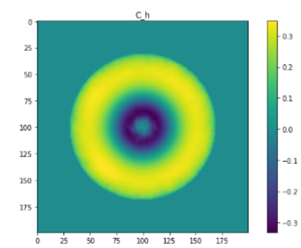
```
nb_vert =  64  ; nb_pts_inter =  20
min(C) =  -1.3784305135502903
max(C) =  0.6775565517686649
||Y_true-Y_true_corr||_L2 0.4496541682615938
```




```
nb_vert =  64  ; nb_pts_inter =  50
min(C) =  -1.767988511885549
max(C) =  0.9318459960999035
||Y_true-Y_true_corr||_L2 0.10616354494514865
```




```
nb_vert =  200  ; nb_pts_inter =  20
min(C) =  -0.3312977177567371
max(C) =  0.34758316279438156
||Y_true-Y_true_corr||_L2 0.7271864833222375
```



Ces résultats ont été obtenus la semaine suivante.

Conclusion

L'idée pour la semaine prochaine et d'effectuer quelques tests pour essayer de comprendre ce qui pose problème avec l'interpolation. Une fois ce type de problème réglé, on espère pouvoir entrainer le modèle sur la v100.

4 Semaine 4 : 27/02/2023 - 03/03/2023

Résumé

On cherche à comprendre les problèmes d'interpolation de phi dans le cadre de la correction sur la solution exacte du problème de poisson avec condition de dirichlet homogène. Après la réunion du 28/02 avec Emmanuel et Vanessa, les points suivants sont à traiter :

- tester avec solution analytique + perturbation !
- enlever les termes de stabilisation afin de voir si le problème est dans la dérivée seconde de phi
- visualiser le ϕ et le ϕ' EF et le comparer au ϕ et ϕ' analytique
- comprendre le problème du extrapolate

Après la réception d'un ordinateur, on a passé la moitié de la semaine à essayer d'installer ce dont on avait besoin. En fin de semaine, les installations ont enfin été faite et je peux maintenant générer des données et entraîner le modèle sur ce nouveau PC.

4.1 Génération des données

On considère toujours Ω le cercle de rayon $\sqrt{2}/4$ et de centre $(0.5, 0.5)$ avec $\Phi(x,y) = -1/8+(x-1/2)^2+(y-1/2)^2$ et le domaine fictif $O = (0,1)^2$.

On souhaite résoudre

$$\begin{cases} -\Delta u & = f\,, \quad \text{dans } \Omega\,, \\ u & = 0\,, \quad \text{sur } \Gamma\,, \end{cases}$$

Notre solution analytique est

$$u_{ex}(x,y) = \frac{1}{\sin\left(k_1\frac{\pi}{2}\right)} \times \sin\left(k_1\frac{\pi}{2}\left(\frac{4}{\sqrt{2}}\right)^2\left((x-0.5)^2+(y-0.5)^2\right)\right) \times \cos\left(\frac{\pi}{2}\left(\frac{4}{\sqrt{2}}\right)^2\left((x-0.5)^2+(y-0.5)^2\right)\right)\,,$$

avec $k_1 \sim \mathcal{U}([0.1, 0.5])$.

4.2 Correction

On cherche à comprendre les problèmes d'interpolation de phi dans le cadre de la correction sur la solution exacte du problème de poisson avec condition de Dirichlet homogène ("tests_interpolation"). On va effectuer plusieurs tests :

Solution analytique + Perturbation.

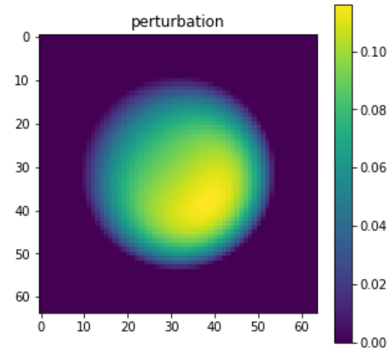
Pour simuler la solution fournit par le FNO on va considérer comme nouvelle level-set notre solution analytique plus une petite perturbation. La perturbation choisie est définie par :

$$P(x,y) = \epsilon \times \sin(k_1(x+y)) \times \cos(4\pi((x-0.5)^2+(y-0.5)^2))$$

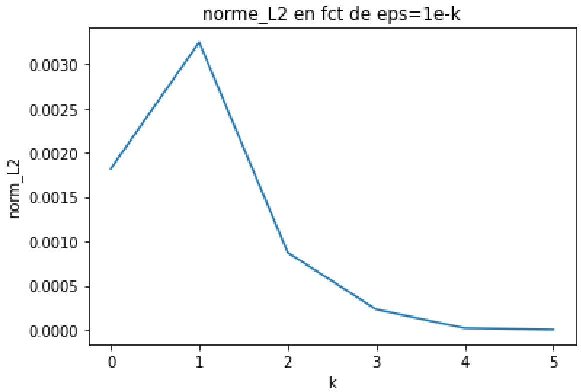
On a alors

$$\Phi(x,y) = u_{ex}(x,y) + P(x,y)$$

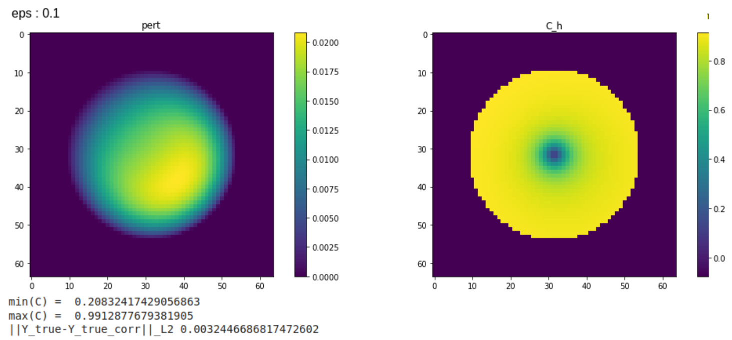
En prenant $\epsilon = 1$, on a :

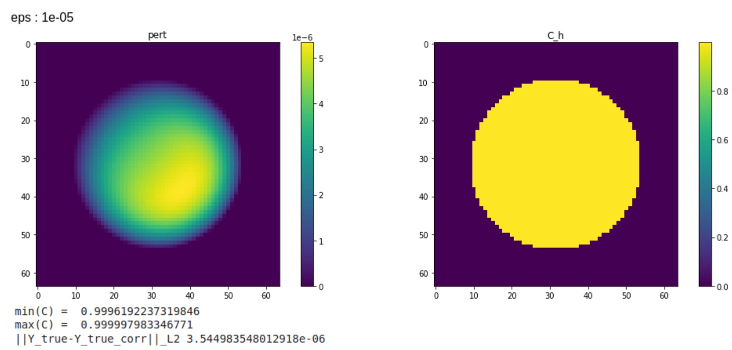


On obtient en normes L_2 les différentes erreurs en fonction de $\epsilon = 1e - k$ entre la solution initiale fournit Φ et le solution après correction ΦC :



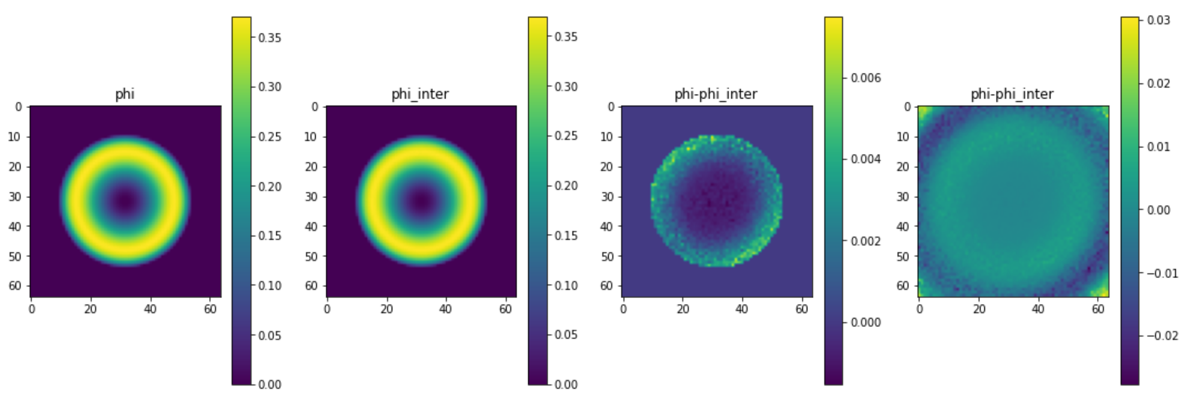
Il semblerait que plus la perturbation est petite et plus l'erreur est petite. De plus, C est de plus en plus proche de 1 :





Griddata

On a comparé Φ exact et notre Φ interpolé avec griddata (pour nb_vert=64 et nb_pts=20). Voici les résultats obtenus :



Il semblerait que sur le bord du cercle, les résultats soit moins bons. On a pas cherché à comprendre plus loin car on va utiliser la fonction extrapolate de FEniCS.

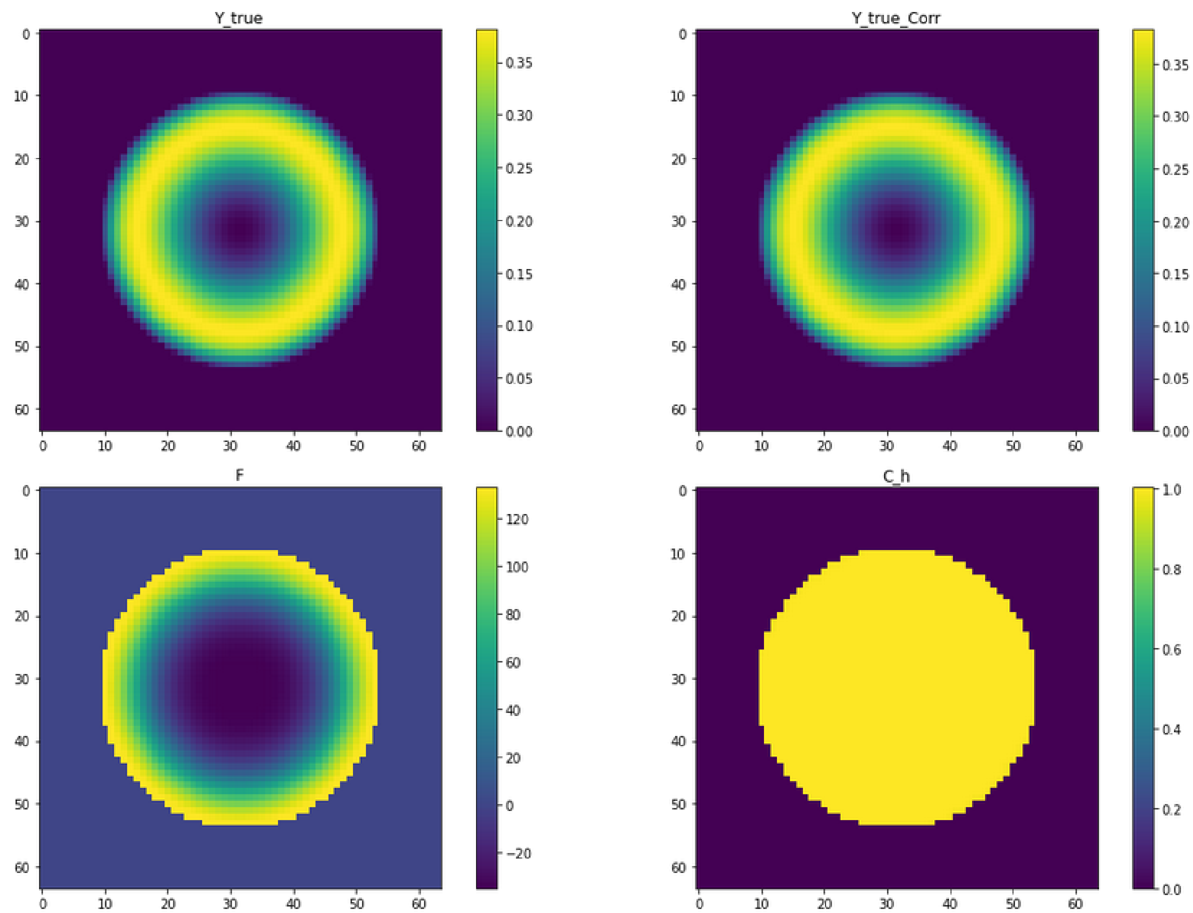
Extrapolate de FEniCS

En milieu de semaine, on s'est rendu compte que le problème avec la foncion extrapolate que l'on avai eut la semaine dernière était que l'on pouvait n'incrémenter le degré d'interpolation que de 1.

```python
def get_phi_fct(Y, degree = 2):
    Y_np = Y[0,:,:,0]

    nb_vert = Y_np.shape[0]
    coeff = 1

    boxmesh = RectangleMesh(Point(0, 0), Point(1,1), nb_vert-1, nb_vert-1)

    V = FunctionSpace(boxmesh, "CG", 1)
    v2d = vertex_to_dof_map(V)

    phi_fct = Function(V)
    phi_fct.vector()[v2d] = Y_np.flatten()

    for i in range(2,degree+1):
        V2 = FunctionSpace(boxmesh, "CG", i)
        phi_fct2 = Function(V2)
        phi_fct2.extrapolate(phi_fct)
        phi_fct = phi_fct2

    return phi_fct
```

Voici les résultats obtenus pour une degré 2 :

```
degree =  2
num of cell in the ghost penalty: 302
||phi_ex-phi_inter||_L2 0.0002493552722685459
solver duration: 22.493684768676758
```




```
min(C) =  0.994576987356184
max(C) =  1.0032304281199327
||Y_true-Y_true_corr||_L2 0.00044278483969670816
```

Conclusion

La semaine prochaine, on peut enfin entraîner le modèle car les installations ont été faites sur le nouveau pc.

5 Semaine 5 : 06/03/2023 - 10/03/2023

Résumé

Pendant cette semaine, on a globalement cherché à comprendre les problèmes liés à l'interpolation de ϕ. On s'est donc concentré sur la précision de la correction lorsque l'on prend la solution analytique en entrée. On a testé avec deux solutions analytiques : la solution trigonométrique considérées précédemment et une solution polynomiale. La partie où l'on va utilisé le FNO sera considéré dans un second temps.

Dans toute la suite, les erreurs calculées en norme L2 sont les erreurs relatives calculées de la manière suivante :

$$||y_{true} - y_{pred}||^2_{L^2(\Omega_h),rel} = \frac{\int_{\Omega_h}(y_{true} - y_{pred})^2}{\int_{\Omega_h} y_{true}^2}$$

5.1 Solution analytique trigonométrique

On considère la solution analytique (au problème de Poisson avec conditions de Dirichlet homogène) :

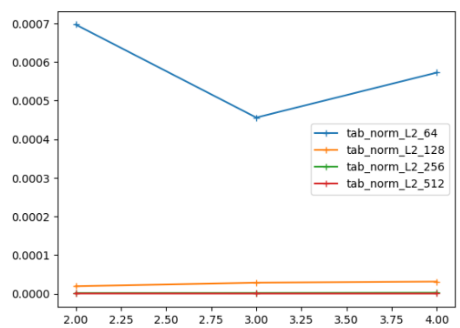
$$u_{ex}(x,y) = \frac{1}{\sin\left(k_1\frac{\pi}{2}\right)} \times \sin\left(k_1\frac{\pi}{2}\left(\frac{4}{\sqrt{2}}\right)^2\left((x-0.5)^2+(y-0.5)^2\right)\right) \times \cos\left(\frac{\pi}{2}\left(\frac{4}{\sqrt{2}}\right)^2\left((x-0.5)^2+(y-0.5)^2\right)\right),$$

avec $k_1 \sim \mathcal{U}([0.1, 1])$.

Extrapolate en faisant varier nb_vert

On va calculer l'erreur en norme L2 pour différents degré d'extrapolation (de 2 à 4) et pour différents nb_vert. On calculera les facteurs multiplicatifs entre les résultats obtenus pour différentes valeurs de nb_vert.

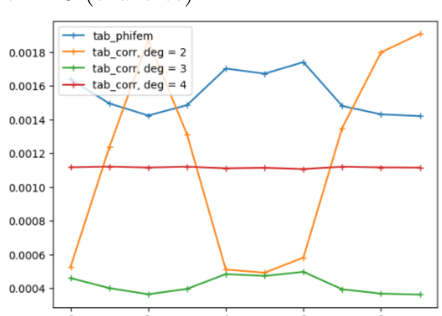
Attention : Les résultats obtenus pour chaque valeur de nb_vert n'ont pas été calculés pour les mêmes valeurs du paramètres k_1.

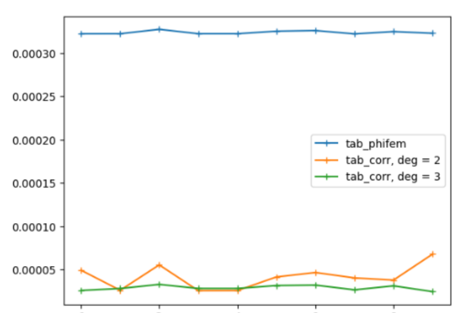



```
tab_norm_L2_64 :  [0.0006967652600855427, 0.00045597122067689814, 0.0005724634964613069]

tab_norm_L2_128 :  [1.922158215675266e-05, 2.823458823293494e-05, 3.11003321120869e-05]
facteur 64-128 :  [36.24911073 16.14938447 18.40698982]

tab_norm_L2_256 :  [1.4710081923791521e-06, 1.6045669223902737e-06, 2.1645064826912723e-06]
facteur 64-256 :  [473.66511192 284.1708964  264.47760773]
facteur 128-256 :  [13.06694433 17.5963918  14.36832477]

tab_norm_L2_512 :  [2.1037824077334138e-07, 1.0974522716743716e-07, 1.1186391627112441e-07]
facteur 64-512 :  [3311.96447657 4154.81595369 5117.49915025]
facteur 128-512 :  [ 91.36677865 257.27395133 278.01933947]
facteur 256-512 :  [ 6.99220693 14.62083558 19.34946098]
```

Comparaison avec PhiFEM

On veut comparer les erreurs en normes L2 pour différents degré d'interpolation. On fait une moyenne des résultats obtenus sur 10 valeurs de paramètres k_1 (nb_data=10). Les comparaisons entre PhiFEM et la correction (pour différents degré d'interpolation) sont effectuées pour les mêmes valeurs de paramètres. On testera avec nb_vert=64 (à gauche) et nb_vert=128 (à droite).






```
[0.00154893 0.00154893 0.00154893]
[0.00115899 0.00042031 0.00111569]
facteur :  [1.33644568 3.68519288 1.38831814]
```

```
[0.00032393 0.00032393]
[4.15350306e-05 2.88233974e-05]
facteur :  [ 7.79897413 11.23846107]
```

Stratégie P1 fin -> Pk grossier

Méthode précédente (avec le extrapolate) : On part d'une grille nb_vert × nb_vert. On associe alors les valeurs au noeuds aux valeurs des degré de liberté $\mathbb{P}^1$ puis on va extrapoler en $\mathbb{P}^k$ nb_vert × nb_vert.

$$[\bar{\phi} = \phi w]_{ij\ grossier} \longrightarrow \mathbb{P}^1{}_{grossier} \xrightarrow{extra} \mathbb{P}^k{}_{grossier}$$

On va considérer ici une nouvelle méthode : On part d'une grille fine nb_vert_fine × nb_vert_fine. On associe alors les valeurs aux noeuds aux valeurs des degrés de liberté $\mathbb{P}^1$ (nb_vert_fine × nb_vert_fine) puis on interpole en $\mathbb{P}^k$ nb_vert_coarse × nb_vert_coarse.

$$[\bar{\phi} = \phi w]_{ij\ fin} \longrightarrow \mathbb{P}^1{}_{fin} \xrightarrow{inter} \mathbb{P}^k{}_{grossier}$$

On effectuera plusieurs comparaisons (avec nb_data=10) :

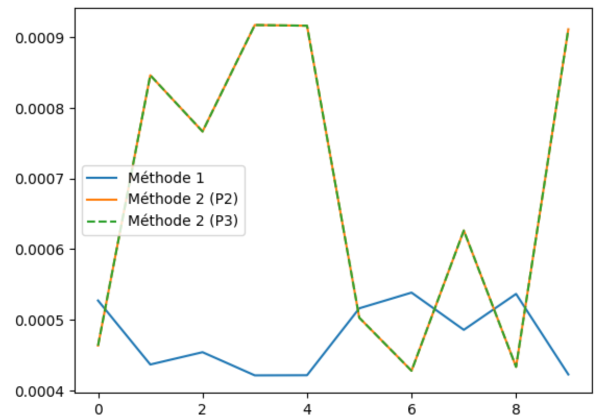
- **1er test :**

On va comparer les méthodes suivantes :

$$[\bar{\phi} = \phi w]_{ij\ 64} \longrightarrow \mathbb{P}^1{}_{64} \xrightarrow{extra^3} \mathbb{P}^3{}_{64}$$

$$[\bar{\phi} = \phi w]_{ij\ 256} \longrightarrow \mathbb{P}^1{}_{256} \xrightarrow{inter} \mathbb{P}^2{}_{64}$$

$$[\bar{\phi} = \phi w]_{ij\ 256} \longrightarrow \mathbb{P}^1{}_{256} \xrightarrow{inter} \mathbb{P}^3{}_{64}$$



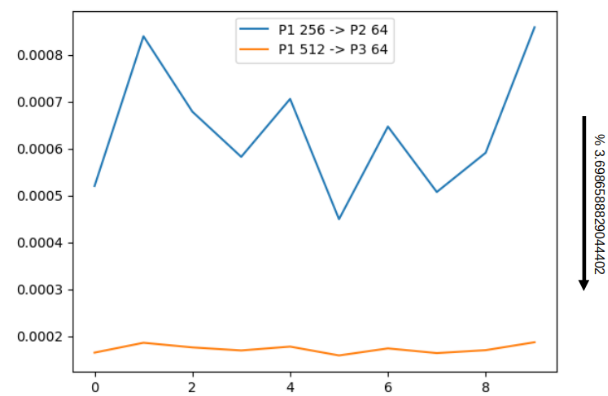
(Les courbes orange et verte sont très proches.)

- **2ème test :**

On va comparer les méthodes suivantes :

$$[\bar{\phi} = \phi w]_{ij\ 256} \longrightarrow \mathbb{P}^1{}_{256} \xrightarrow{inter} \mathbb{P}^2{}_{64}$$

$$[\bar{\phi} = \phi w]_{ij\ 512} \longrightarrow \mathbb{P}^1{}_{512} \xrightarrow{inter} \mathbb{P}^3{}_{64}$$



- **3ème test :**

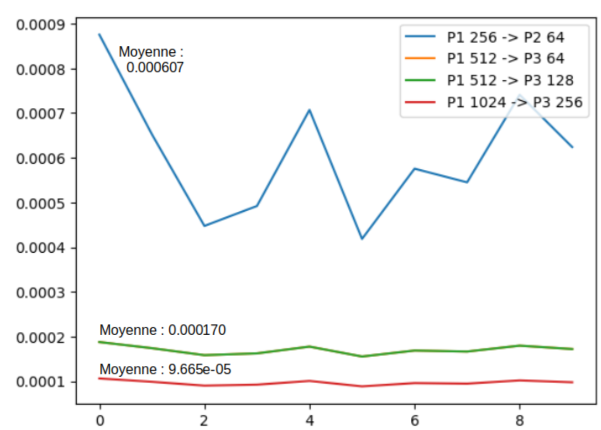
On va comparer les méthodes suivantes :

$$[\bar{\phi} = \phi w]_{ij\ 256} \longrightarrow \mathbb{P}^1{}_{256} \xrightarrow{inter} \mathbb{P}^2{}_{64}$$

$$[\bar{\phi} = \phi w]_{ij\ 512} \longrightarrow \mathbb{P}^1{}_{512} \xrightarrow{inter} \mathbb{P}^3{}_{64}$$

$$[\bar{\phi} = \phi w]_{ij\ 512} \longrightarrow \mathbb{P}^1{}_{512} \xrightarrow{inter} \mathbb{P}^3{}_{128}$$

$$[\bar{\phi} = \phi w]_{ij\ 1024} \longrightarrow \mathbb{P}^1{}_{1024} \xrightarrow{inter} \mathbb{P}^3{}_{256}$$



(Les courbes orange et verte sont très proches.)

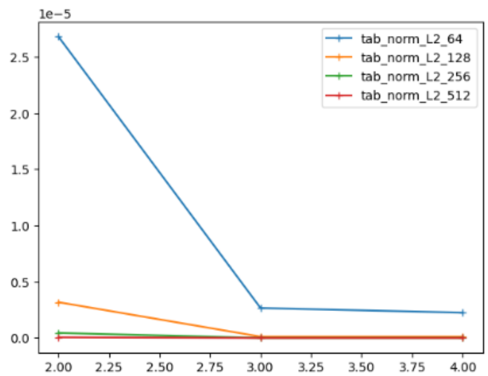
5.2 Solution analytique polynomiale

On considère la solution analytique (au problème de Poisson avec conditions de Dirichlet homogène) :

$$u_{ex}(x,y) = (1 + k_1 * x + k_2 * y + x^2 + y^2 + x^3 + y^3) \times \phi(x,y)$$

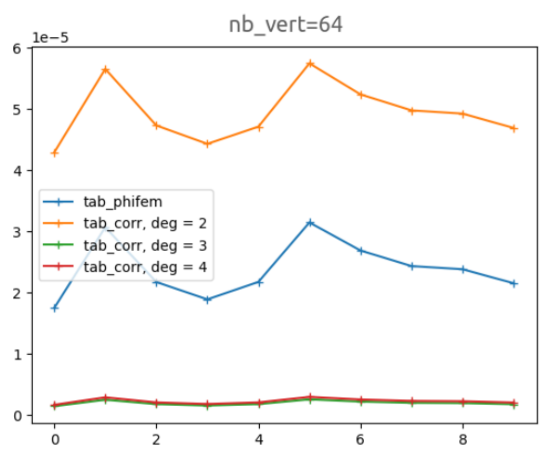
avec $k_1, k_2 \sim \mathcal{U}(1,5)$
On va effectuer exactement les mêmes tests que dans le cas trigonométrique.

Extrapolate en faisant varier nb_vert

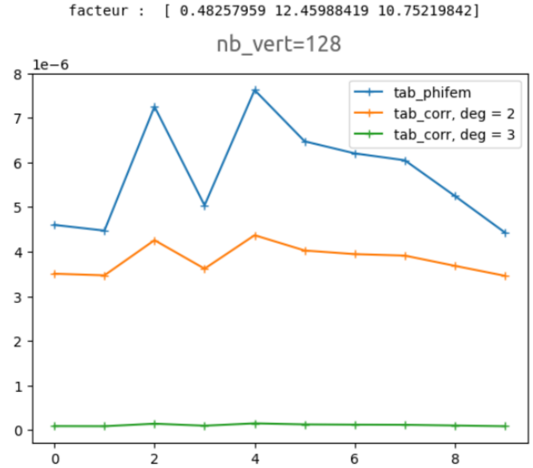



```
tab_norm_L2_64 :  [2.6795787869257804e-05, 2.6620051945242716e-06, 2.251929379799515e-06]

tab_norm_L2_128 :  [3.1816265809951308e-06, 1.1787885864954612e-07, 1.1458184372097631e-07]
facteur 64-128 :  [ 8.42204048 22.58254979 19.65345736]

tab_norm_L2_256 :  [4.45770025132257e-07, 5.792778233797276e-09, 8.63884715265421e-09]
facteur 64-256 :  [ 60.11123754 459.53859911 260.67475671]
facteur 128-256 :  [ 7.13737219 20.34927869 13.26355724]

tab_norm_L2_512 :  [5.892283603687297e-08, 3.271917059060034e-10, 5.05223428330941e-10]
facteur 64-512 :  [ 454.76066109 8135.91893215 4457.29404758]
facteur 128-512 :  [ 53.99649431 360.27459291 226.79439887]
facteur 256-512 :  [ 7.56531856 17.70453874 17.09906285]
```

Comparaison avec PhiFEM



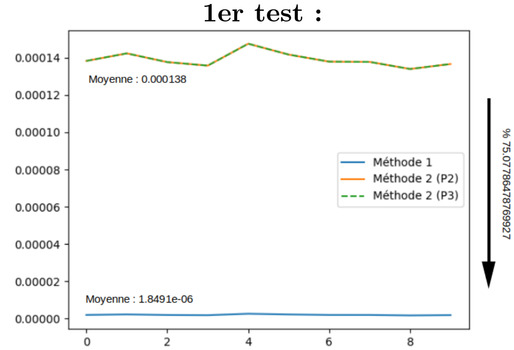

```
[2.37940622e-05 2.37940622e-05 2.37940622e-05]
[4.93059856e-05 1.90965356e-06 2.21294857e-06]
facteur :  [ 0.48257959 12.45988419 10.75219842]
```



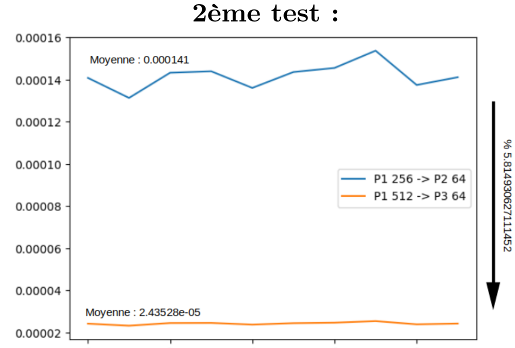

```
[5.74036956e-06 5.74036956e-06]
[3.82656316e-06 1.16131386e-07]
facteur :  [ 1.50013715 49.42995808]
```

Stratégie P1 fin -> Pk grossier

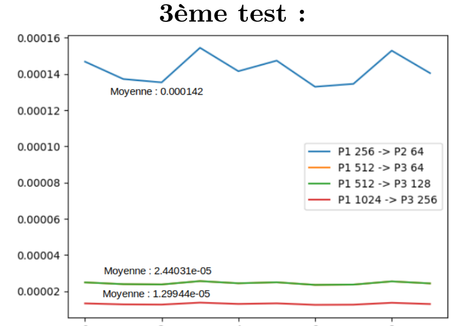
1er test :



2ème test :



3ème test :

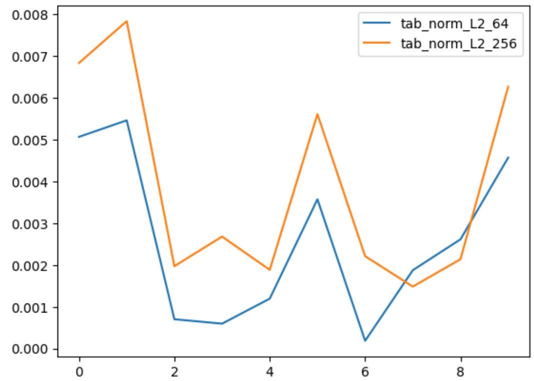


5.3 Tests sur le FNO

On veut maintenant comparer la première stratégie utilisée (avec le extrapolate à degré fixé égal à 3) et la nouvelle stratégie (P1 fin puis interpolation en Pk grossier). on considère ici que la solution que l'on va introduire dans la correction/certification est la sortie d'un FNO.

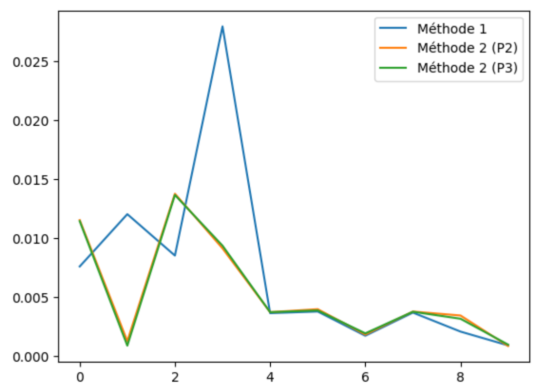
Appel avec nb_vert plus fin

On entraine le réseau FNO avec nb_vert=64. Avant de mettre en place la stratégie, on cherche à voir si la précision de ce réseau (entrainé avec nb_vert=64) est meilleure lorsque l'on prend nb_vert plus petit. On va alors comparer l'erreur pour nb_data=10 lorsqu'on l'on appelle le réseau avec nb_vert=64 et nb_vert=256. Voici les résultats obtenus :



Comparaison des 2 stratégies

On va maintenant comparer la première méthode utilisée (avec le extrapolate à degré fixé égal à 3) et la nouvelle méthode (P1 fin puis interpolation en Pk grossier). On prendra nb_vert=256. Voici les résultats obtenus (pour $k = 2$ et $k = 3$) :



Conclusion

Pour l'instant, on ne va s'intéresser qu'à la partie avec la solution analytique. Il semblerait que la première méthode avec le extrapolate de FEniCS soit meilleure que la deuxième stratégie proposée. C'est pourquoi, vendredi, on a commencé (Killian et moi) à s'intéresser au code source du extrapolate afin de comprendre pourquoi les résultats obtenus sont meilleurs. On considérera le FNO dans un second temps.

6 Semaine 6 : 13/03/2023 - 17/03/2023

Résumé

Après les tests de la semaine dernière sur la partie de correction/certification du modèle où l'on prend la solution analytique comme nouvelle level-set, il semblerait que la méthode avec les meilleurs résultats soit celle où l'on utilise la méthode extrapolate de FEniCS. C'est pourquoi, cette semaine on s'est intéressé en détail au code source de cette fonction FEniCS (Extrapolation). Étant donné que je n'étais pas présente mardi, mercredi après-midi et jeudi car j'étais malade, c'est tout ce qui a été fait cette semaine. De plus, une grosse partie de la méthode reste encore floue : la construction de la matrice A (pour la résolution du système linéaire dans compute_coefficients).

Pour illustrer les explications, nous considérerons un domaine rectangulaire maillés uniformément par des triangles. Dans la suite, nous considérerons également *cell0* comme étant une des cellules de maillage. On prendra comme exemple une extrapolation de $\mathbb{P}^1$ vers $\mathbb{P}^2$.

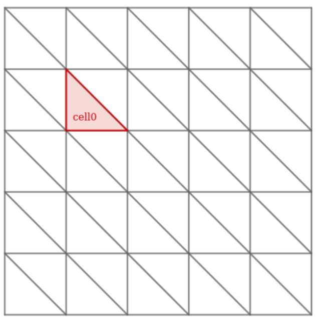


Figure 1 – Cell0

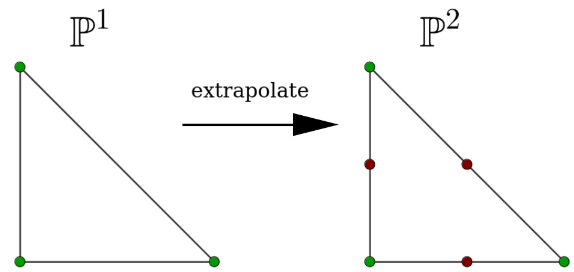


Figure 2 – Extrapolation de $\mathbb{P}^1$ vers $\mathbb{P}^2$

Corps de la fonction **extrapolate** qui a pour but d'extrapoler la fonction v en une fonction w :

```
void Extrapolation::extrapolate(Function& w, const Function& v)
```

On cherche à calculer la valeur en chacun des degrés de liberté associé à la fonction w. Pour cela, on va parcourir toutes les cellules du maillage dans le but de construire le tableau *coefficients* (de taille le nombre total de degrés de liberté associés à w). On appelle alors sur chacune des cellules la fonction **compute_coefficients** 6.1 qui va compléter le tableau *coefficients* aux indices associées à ses degrés de liberté. Comme les degrés de liberté d'une cellule peuvent être communs à ceux d'une autre, on finira par faire une moyenne des coefficients en chacun des degrés de liberté en utilisant la fonction **average_coefficients** 6.5, ce qui nous donne alors la fonction w.

6.1 compute_coefficients

Corps de la fonction :

```
void Extrapolation::compute_coefficients(
std::vector<std::vector<double>>& coefficients,
const Function& v,
const FunctionSpace& V,
const FunctionSpace& W,
const Cell& cell0,
const std::vector<double>& coordinate_dofs0,
const ufc::cell& c0,
const Eigen::Ref<const Eigen::Matrix<dolfin::la_index, Eigen::Dynamic, 1>> dofs,
std::size_t& offset)
```

Cette fonction a pour but de compléter le tableau *coefficients* aux indices associées aux degrés de liberté d'une cellule donnée *cell0*. Autrement dit, on cherche à déterminer les valeurs aux degrés de liberté de la cellule *cell0* en utilisant l'information que nous apporte les cellule voisines à celle-ci.

On commence par construire les tableaux *cell2dof2row* et *unique_ dofs* en utilisant la fonction **build_unique_dofs** 6.2. L'ensemble *unique_ dofs* contient tous les degrés de liberté de notre espace de départ V (associé à la fonction v)

des cellules voisines à *cell0*. Le dictionnaire *cell2dof2row* permet d'associer à chaque degré de liberté (unique) d'une cellule donnée un numéro de ligne unique.

Ensuite, on définit N le nombre de degré de liberté associé à un élément de W (dans notre cas $N = 6$) et M le nombre de degré de liberté (unique) des cellules voisines à la cellule courante *cell0* (dans notre cas les nœuds des cellules voisines et donc $M = 12$). Attention : il faut que $M \geq N$ pour avoir suffisamment de degré de libertés pour pouvoir construire l'extrapolation.

On peut maintenant créer la matrice A (de taille $M \times N$) et le vecteur b (de taille M). En parcourant les cellules voisines de la cellule courante *cell0*, on va compléter la matrice A et le vecteur b en utilisant la fonction **add_cell_equations** 6.4. A noter que la cellule courante *cell0* est inclue dans ses cellules voisines.

On pourra ensuite résoudre le système linéaire $Ax = b$ qui nous donnera la valeur en chacun des degrés de liberté de la cellule courante *cell0*. Ces valeurs sont alors ajoutées au tableau global *coefficients* qui nous fournit après avoir utilisé la fonction **average_coefficients** 6.5 les valeurs en chaque degré de liberté de w.

6.2 build_unique_dofs

Corps de la fonction :

```
void Extrapolation::build_unique_dofs(
std::set<std::size_t>& unique_dofs,
std::map<std::size_t, std::map<std::size_t, std::size_t>>& cell2dof2row,
const Cell& cell0,
const FunctionSpace& V)
```

Cette fonction a pour but de compléter les tableaux *cell2dof2row* et *unique_dofs* donnés en entrée. A noter que au total, on a le même nombre de degré de liberté dans *cell2dof2row* et *unique_dofs*.

On commence par remplir un ensemble contenant les cellules voisines à *cell0*. Pour être plus précis, les cellules voisines à *cell0* sont les cellules ayant un nœud commun avec la cellule courante *cell0* (Figure 3).

En parcourant ensuite chacune de ces cellules, on va pouvoir compléter les tableaux *cell2dof2row* et *unique_dofs* donnés en entrée en appelant la fonction **compute_unique_dofs** 6.3.

Dans le cas de notre exemple, on va numéroter tous les nœuds des cellules voisines à *cell0* et on supposera que le parcours des cellules est effectuées dans un ordre précis (Figure 4).

Alors le set *unique_dofs* contiendra tous les noeuds des cellules voisines à *cell0* :

$$unique\_dofs = \{n1, n2, n3, \ldots, n12\}$$

Et le dictionnaire *cell2dof2row* associé à *cell0* est construit de la manière suivante :

$$\begin{aligned} cell2dof2row = \{ \quad & "cell0" : \{"n1" : 0, "n2" : 1, "n3" : 2\}, \\ & "cell1" : \{"n4" : 3\}, \\ & "cell2" : \{"n5" : 4\}, \\ & \ldots \quad \} \end{aligned}$$

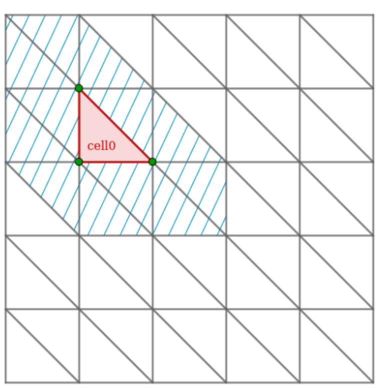


Figure 3 – Cellules voisines à Cell0

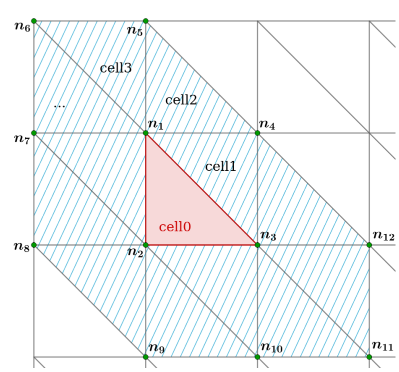


Figure 4 – Parcours des cellules voisines

6.3 compute_unique_dofs

Corps de la fonction :

```
std::map<std::size_t, std::size_t> Extrapolation::compute_unique_dofs(
const Cell& cell,
const FunctionSpace& V,
std::size_t& row,
std::set<std::size_t>& unique_dofs)
```

Cette fonction a pour but de traiter chacune des cellules voisines afin de compléter les tableaux *unique_dofs* et *cell2dof2row* créés dans **compute_coefficients** 6.1.

Pour une cellule *cell* (voisine à *cell0*), on va parcourir chacun de ses degrés de liberté associé à l'espace V. Autrement dit, on parcourt tous les degrés de liberté dont la valeur est connue (dans notre cas les degrés de liberté $\mathbb{P}^1$ et donc les noeuds de la cellule). Si ce degré de liberté fait partie de *unique_dofs*, on ne fait rien. Sinon, on l'ajoute à *unique_dofs*. On va également créer un tableau *dof2row* qui a pour but d'associer un degré de liberté à un numéro de ligne unique. Ce dictionnaire *dof2row* est retourné par la fonction et permet de remplir le dictionnaire plus général *cell2dof2row* créé dans **compute_coefficients** 6.1.

6.4 add_cell_equations

Corps de la fonction :

```
void Extrapolation::add_cell_equations(
Eigen::MatrixXd& A,
Eigen::VectorXd& b,
const Cell& cell0,
const Cell& cell1,
const std::vector<double>& coordinate_dofs0,
const std::vector<double>& coordinate_dofs1,
const ufc::cell& c0,
const ufc::cell& c1,
const FunctionSpace& V,
const FunctionSpace& W,
const Function& v,
std::map<std::size_t, std::size_t>& dof2row)
```

Cette fonction a pour but de remplir une partie de la matrice A et du vecteur b à partir de la cellule courante *cell0* et d'une de ses cellules voisines *cell1*.

On commence par créer les fonctions de base Φ_j associées aux degrés de liberté de la cellule courante *cell0*.
On va ensuite parcourir les degrés de liberté associés à la cellule voisine *cell1* dans le dictionnaire *cell2dof2row* (c'est le tableau *dof2row* donné en argument). On évalue alors Φ_j en chacun des degrés de liberté de la cellule voisine *cell1*. On peut alors compléter $A(row, j)$ (où row nous ai donné par le dictionnaire *dof2row*). On complète également $b(row)$ par la valeur aux degrés de liberté associés à l'espace V (les nœuds dans notre cas).

6.5 average_coefficients

Corps de la fonction :

```
void Extrapolation::average_coefficients(
Function& w,
std::vector<std::vector<double>>& coefficients)
```

Cette fonction a pour but de faire la moyenne pour chaque degré de liberté des coefficients calculés. Elle associe ensuite ces valeurs au vecteur w.

Conclusion

La semaine prochaine, il faudrait continuer à essayer de comprendre la partie de construction de la matrice A.