Pdf resumé cours

Séance 1

On a commencé cet UE par un petit exercice de remise en forme, où il a été demandé de generer un code python pour résoudre une équation dans un domaine rectangulaire:

$$u_t + < V, \nabla u > -\nu\nabla(\nabla u) = -\lambda u + f \tag{1}$$

Tel que :

$$\begin{aligned} f(t,s) &= T_c \exp(-kd(s,s_c)^2) \\ d(s,s_c)^2 &= (s_1 - (s_c)_1)^2 + (s_2 - (s_c)_2)^2 \end{aligned} \tag{2}$$

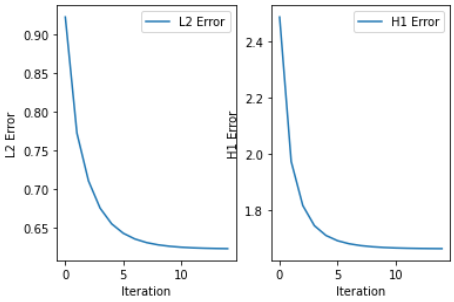
En utilisant Chatgpt cela se fait très rapidement en adaptant le code généré à notre problème. Puis on l'a fait sur du 1D avec des condition de Dirichlet et de Neumann.

Séance 2

On a ensuite, découvert "adrs.py" un code python qui résout :

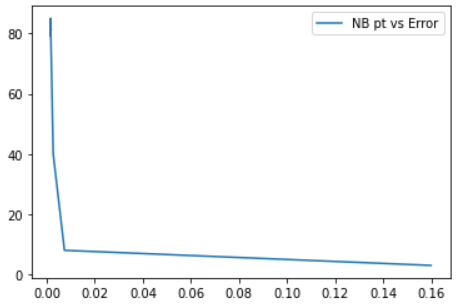
$$\begin{aligned} u_t + v u_{,s} - \nu u_{,ss} + \lambda u &= f(s) \\ u(s) &= \exp(-10(s - \frac{L}{2})^2) \end{aligned} \tag{3}$$

On utilise ce code qui sera de plus en plus précis en fonction du nombre d'itération raffinant le maillage. On affiche par ailleur l'erreur en norme L2 contre H1:

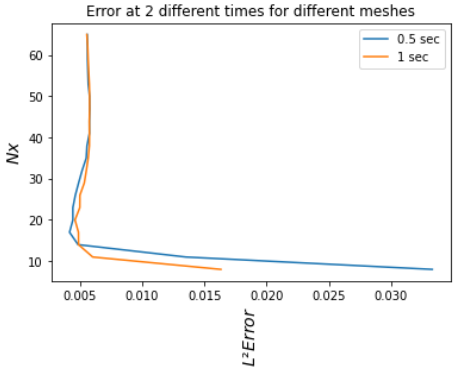


Séance 3

Cette fois-ci on a pris le code "adrsmeshadapt.py" où on a mis en place un contrôle local de métrique, en affichant l'erreur en fonction du nombre de points:

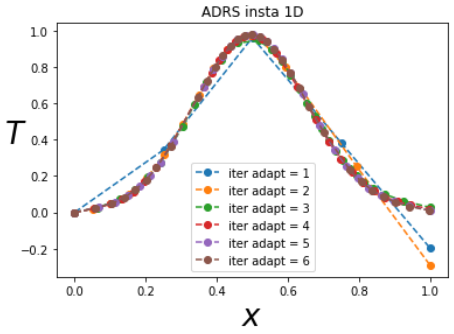


On a aussi appliqué Range-Kutta ordre 3 sur le code :



Séance 4

On a introduit dans cette séance un terme source qui dépend du temps, et donc le cas instationnaire dans le code "adrsmeshadaptinsta.py" :



Séance 5

Pendant cette séance, nous avons comparé deux techniques d'intégration, l'intégration de Riemann et l'intégration de Lebesgues que l'on peut voir dans le code "RiemannVLebesgue.py", et on voit qu'en adaptant le pas d'intégration, Lebesgues est plus précis.

Séance 6

Dans cette séance, on a appliqué le code adrs à de l'optimisation, donc minimiser une fonctionnelle :

$$\min J(u_h(x,s)) \tag{4}$$

Sous la contrainte :

$$F_h(u_h(x,s)) = f(x) \tag{5}$$

Tel que

$$f = \sum_i \alpha_i \sigma(s_i) \tag{6}$$

avec comme approximation :

$$\alpha_i \sigma(s_i) \sim \alpha_i \exp(-\beta(s - s_i)^2), \quad \beta >> 1 \tag{7}$$

Et

$$J(u_h(x,s)) = \frac{1}{2} \int_{\omega_h} (u_h(x,s) - u^*(s))^2 ds \tag{8}$$

Ainsi en minimisant J, on obtiendra le x optimal, comme dans le code "adrs-opti.py"