Report

Mastère M2 MCHPS

TPs de Réduction de Modèles

Réalisé par: Aicha Maaoui

Mars 2023

ENS Paris-Saclay

Contents

List of Figures

Chapter 1

TP 1-Compression de données (A-Posteriori)

1.1 Introduction

On s'intéresse dans une première partie de ce TP à la compression d'une image donnée, en utilisant la redondance de l'information. En particulier, on utilise la *POD* par la décomposition en valeurs singulières (*SVD*), en évaluant l'erreur de l'approximation. Dans une deuxième partie, la *PGD* est implémentée pour approximer un champ spatio-temporel aléatoire. On ajoute l'orthogonalisation des modes spatiaux (*Gram-Schmidt*) et on évalue de nouveau l'erreur d'approximation.

1.2 Approximation POD par SVD de l'image

On considère l'image de *Maria Callas*, chargée dans *Matlab* à l'aide de *imread* et convertie en une matrice de pixels de dimensions $n_T \times n_M$. On souhaite approximer l'image, représentée par la matrice de pixels $\underline{\underline{M}}$, en utilisant un certain nombre de modes N, de telle sorte que l'erreur d'approximation soit minimale.

Principe de la décomposition de l'image en valeurs singulières

La décomposition d'une matrice $\underline{\underline{M}}$ (de dimensions $m \times n$) en valeurs singulières consiste

à la représenter sous la forme d'un produit de trois matrices comme suit:

$$\boxed{\underline{\underline{M}} = \underline{\underline{U}}\ \underline{\underline{S}}\ \underline{\underline{V}}^T} \tag{1.1}$$

où:

- $\underline{\underline{U}}$ est une matrice de vecteurs propres à gauche de dimension $m \times m$,
- $\underline{\underline{S}}$ est une matrice diagonale de valeurs singulières de dimension $m \times n$,
- $\underline{\underline{V}}$ est une matrice de vecteurs propres à droite de dimension $n \times n$.

Ce principe est également illustré dans la figure (1.1).

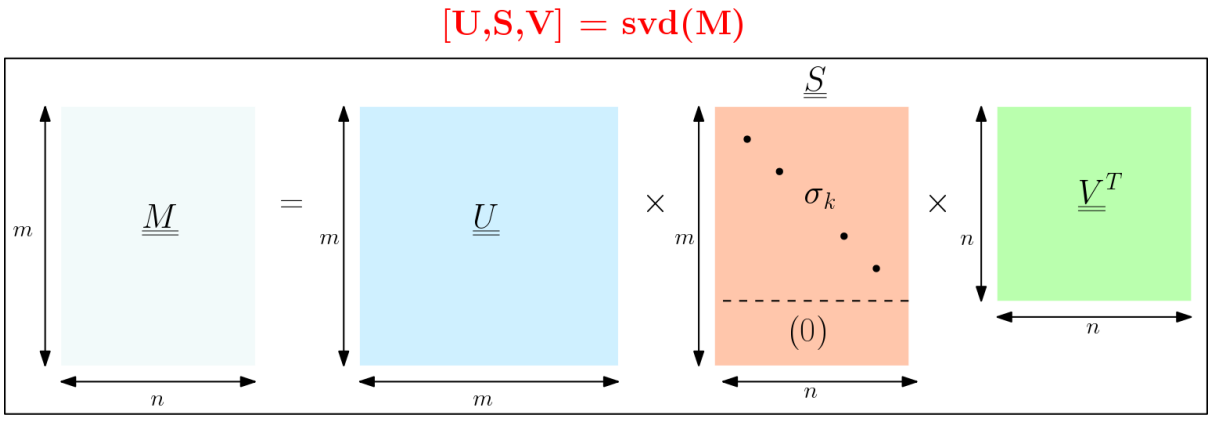


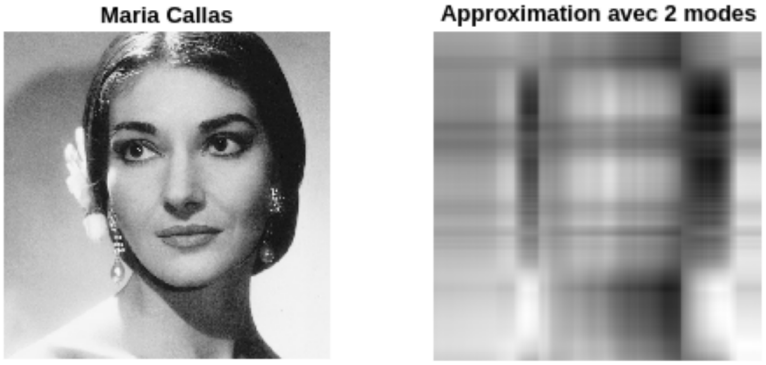
Figure 1.1: Décomposition de la matrice $\underline{\underline{M}}$ en valeurs singulières.

Application à la compression de l'image

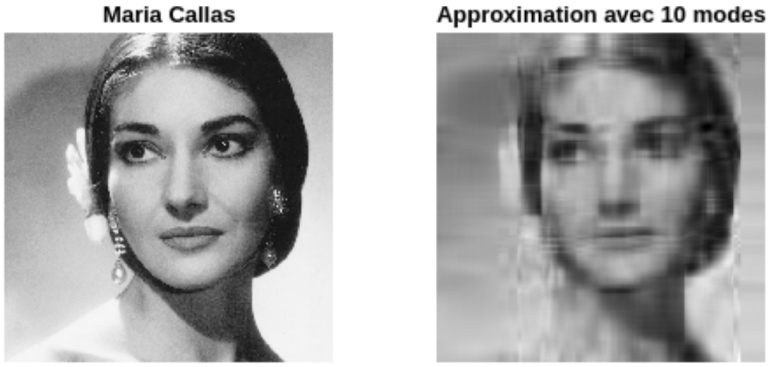
On choisit un nombre de modes N. En utilisant la décomposition de la matrice $\underline{\underline{M}}$ en valeurs singulières dans l'équation (1.1), la reconstruction de l'image avec N modes est donnée dans l'équation (1.2) tel que:

$$\underline{\underline{M}}_{POD} = \sum_{i=1}^{N} \underline{S}_i\ \underline{U}_i\ \underline{V}_i^T \tag{1.2}$$

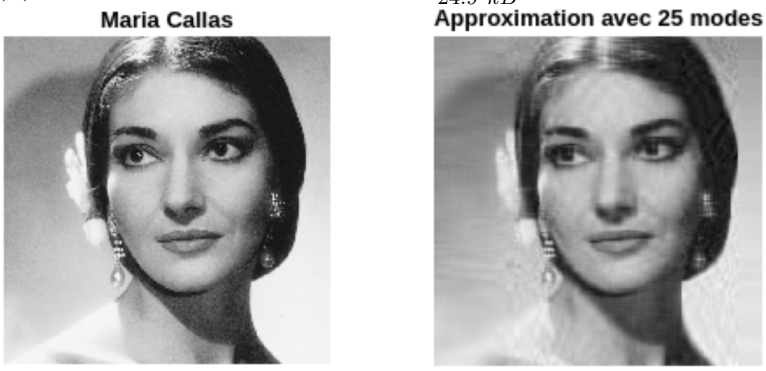
Afin de visualiser l'image compressée et de la comparer en fonction du nombre de modes, on fixe à chaque fois N à 2, 10, 25 et 50. On obtient les résultats des images reconstruites illustrées dans la figure (1.2). On note la taille de l'image originale $Taille_{init} = 87.6\ kB$ et on calcule pour chaque nombre de modes le gain en compression égal à $\frac{Taille_{init}}{Taille_{new}}$.



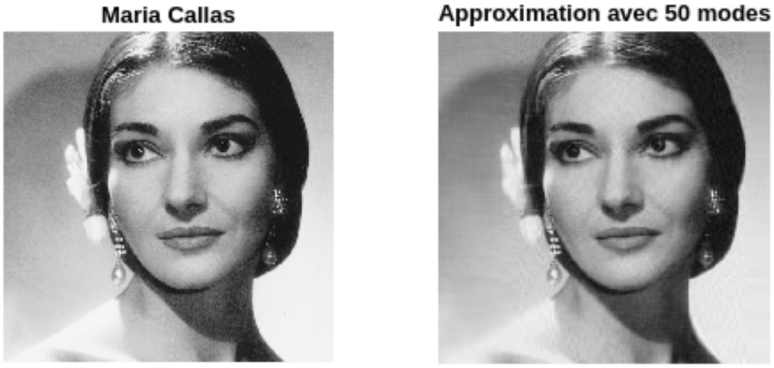
(a) $\mathbf{N = 2}$: grain de compression $= \frac{87.6\ kB}{17.5\ kB} \approx 5$



(b) $\mathbf{N = 10}$: grain compression $= \frac{87.6\ kB}{24.9\ kB} \approx 3.5$



(c) $\mathbf{N = 25}$: grain compression $= \frac{87.6\ kB}{30.5\ kB} \approx 2.9$



(d) $\mathbf{N = 50}$: grain compression $= \frac{87.6\ kB}{36.2\ kB} \approx 2.4$.

Figure 1.2: Reconstruction - compression de l'image donnée en fonction du nombre de modes N.

D'après la figure (1.2), on déduit qu'une reconstruction de l'image avec un nombre faible de modes ($N = 2$) donne une image floue. Une augmentation du nombre de modes permet d'améliorer la qualité de cette dernière, comme dans le cas de $N = 50$. Mais la question qui se pose à ce stade est: **Comment choisir le nombre de mode adéquat?**

Définition de l'erreur d'approximation

Pour répondre à la question précédente, un critère d'évaluation doit être mis en place, soit l'erreur d'approximation ou de reconstruction définie dans l'équation (1.3).

$$error = \frac{||\underline{\underline{M}} - \underline{\underline{M}}_{POD}||}{||\underline{\underline{M}}||} \tag{1.3}$$

L'évolution de cette erreur en fonction du nombre de modes N est tracée à l'aide de *Matlab*, comme montré dans la figure (1.3).

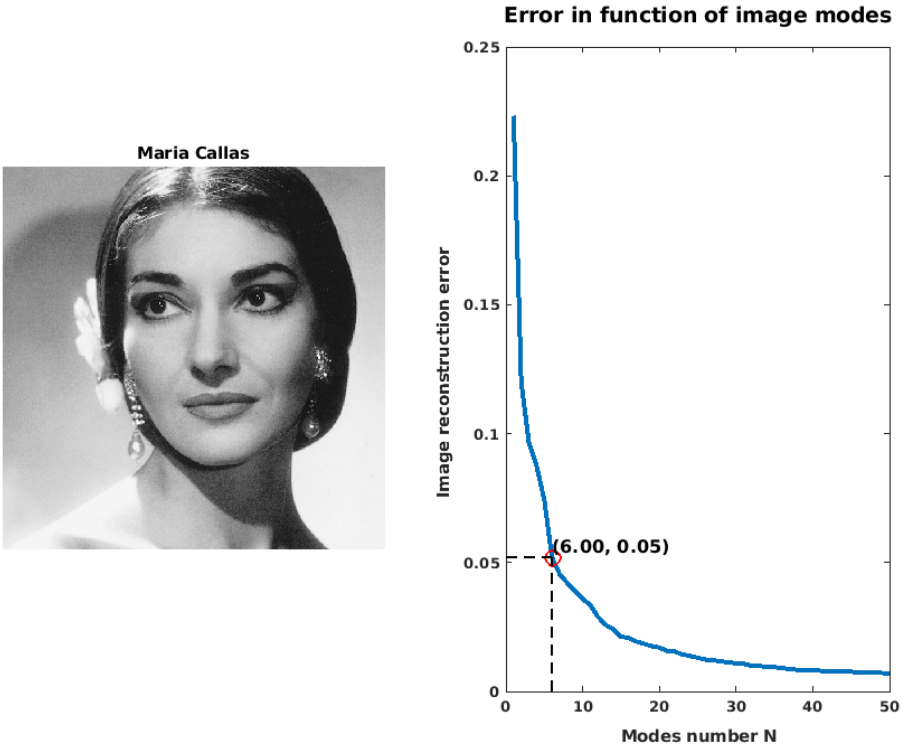


Figure 1.3: Évolution de l'erreur de reconstruction de l'image en fonction du nombre des modes.

On sélectionne le nombre de modes minimisant l'erreur. Par exemple, pour obtenir une erreur de 0.05, le nombre de modes doit être égal à 6. Un nombre de modes $N = 50$ permet de réduire davantage cette erreur.

1.3 Approximation POD d'un champ spatio-temporel aléatoire

1.3.1 Définition et maillage du champ spatio-temporel aléatoire

On considère dans cette section une fonction aléatoire de 2 variables (x,t) définie par:

$$\begin{cases} U: & W \times I \mapsto \mathbb{R} \\ & (x,t) \mapsto U(x,t) \end{cases} \tag{1.4}$$

où x est la variable espace et t est la variable temps.

On souhaite approximer ce champs spatio-temporelle par un champ de dimension inférieure m représentant le nombre de modes, de telle sorte que l'erreur d'approximation soit minimale (problème d'optimisation). Soit en utilisant la séparation de variables:

$$\boxed{\underline{\underline{U}}(x,t) \approx \underline{\underline{U}}_m(x,t) = \sum_{i=1}^{m} \underline{\Lambda}_i(x)\ \underline{\Gamma}_i(t)^T} \tag{1.5}$$

où:

- $\underline{\underline{U}}$ est une matrice de dimension $N_x \times N_t$ dépendant de x et t
- $\underline{\Lambda}_i$ est un vecteur de dimension $N_x \times 1$ dépendant que de x
- $\underline{\Gamma}_i$ est un vecteur de dimension $N_t \times 1$ dépendant que de t

Discrétisation de la fonction *U*

Discrétisation temporelle

L'échelle de temps $[0,T]$ est discrétisé en un ensemble de points avec un pas de temps constant $\Delta t = \frac{T}{n_t - 1}$, où T représente le temps maximal de la simulation (en secondes) et n_t le nombre de points de discrétisation temporelle.

On définit deux types de maillage:

- Un maillage grossier ("coarse distribution"): L'intervalle de temps t_c est discrétisé de 0 à T avec un nombre de points égales $\frac{n_t}{10}$,
- Un maillage fin ("Fine distribution"): L'intervalle de temps t_f est discrétisé de 0 à T avec un nombre de points égales n_t.

Discrétisation spatiale

L'échelle de l'espace $[0, L]$ est discrétisé en un ensemble de points avec un pas d'espace constant $\Delta x = \frac{L}{n_x - 1}$, où L représente la longueur spatiale maximale dans la simulation et n_x le nombre de points de discrétisation spatiale.

On définit deux types de maillage:

- Un maillage grossier ("coarse distribution"): L'intervalle de l'espace x_c est discrétisé de 0 à L avec un nombre de points égales $\frac{n_x}{10}$,
- Un maillage fin ("Fine distribution"): L'intervalle de l'espace x_f est discrétisé de 0 à L avec un nombre de points égales n_x.

Création de maillage grossier et fin

La création du maillage grossier et fin se fait à l'aide de la fonction *meshgrid* de *Matlab*, tel que:

- Le maillage fin spatial $mesh\_x\_f$ est crée à partir du vecteur x_f, alors que le maillage fin temporel $mesh\_t\_f$ est crée à partir du vecteur t_f,
- Le maillage grossier spatial $mesh\_x\_g$ est crée à partir du vecteur x_c, alors que le maillage grossier temporel $mesh\_t\_g$ est crée à partir du vecteur t_c

Une distribution aléatoire du maillage grossier U_g de dimension $(\frac{n_t}{10} \times \frac{n_x}{10})$ est crée dans *Matlab*. Elle est interpolée dans la suite sur le maillage fin avec la fonction *interp2* afin d' obtenir une distribution plus lisse.

On trace ainsi les deux distributions du maillage grossier et fin avec la fonction *surf*. Un affichage lisse est mis en place avec la fonction *shading interp*. Ceci est illustré dans la figure (1.4). La distribution du maillage fin contient plus d'information, mais conduit à

la plus grande précision. De plus, si le champ est trop discontinu, le PGD aura du mal à l'approximer.

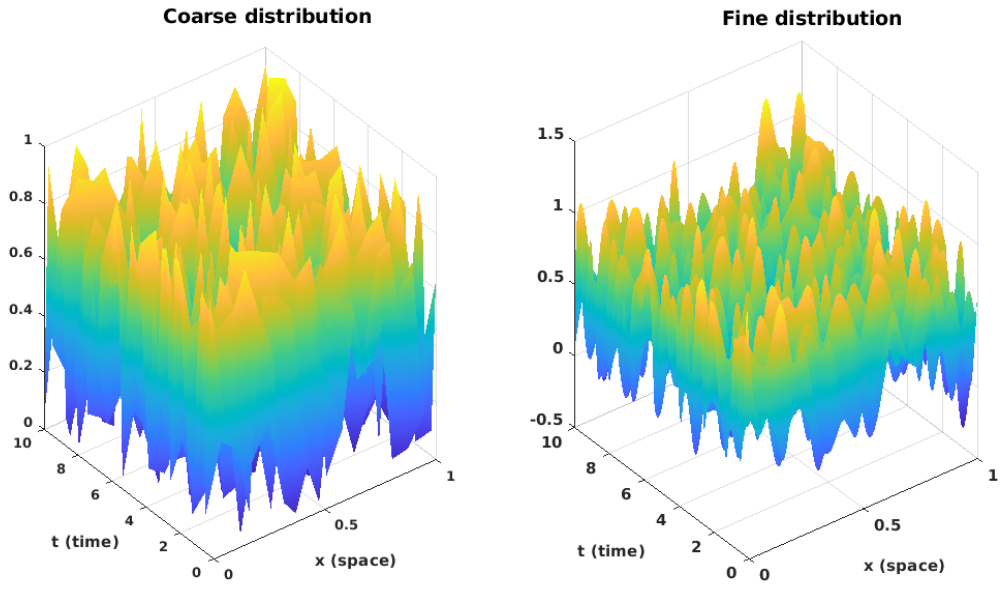


Figure 1.4: Distribution aléatoire du maillage grossier et fin.

1.3.2 Calcul des intégrales par la méthode des éléments finis

$\underline{\Lambda}$ et $\underline{\Gamma}$ sont des champs discrets. Par conséquent, on peut les interpoler à l'aide de fonctions de forme (même manière que dans la méthode des éléments finis) afin de calculer les intégrales. Dans ce TP, les fonctions de forme les plus simples "linéaires par morceaux" sont utilisées.

Calcul de l'intégral spatial

Pour calculer l'intégral spatial à l'aide des fonctions de formes linéaires par morceaux, on note:

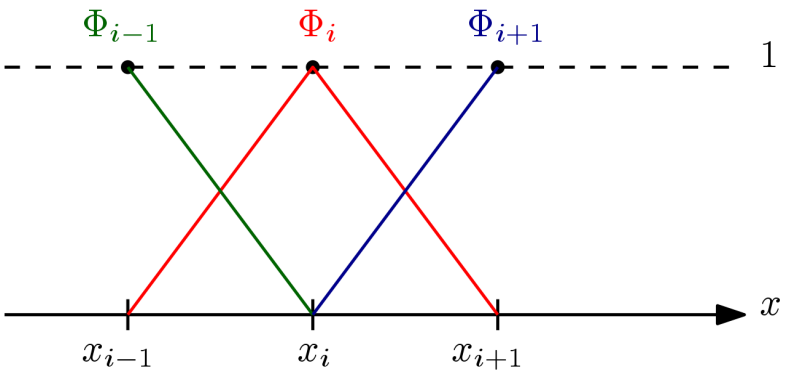
$$\underline{\Lambda}(x) = \sum_{i=1}^{N_x} \Phi_i(x)\ \Lambda_i \tag{1.6}$$

avec:

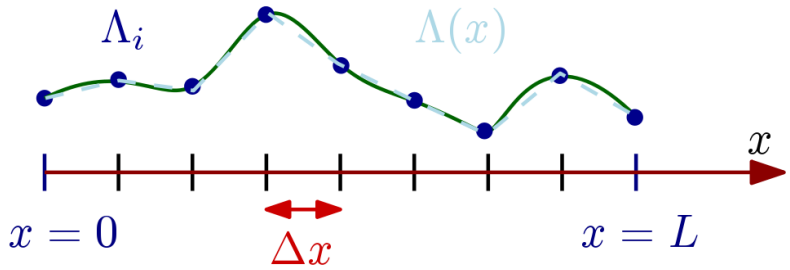
$$\begin{cases} \Lambda_i & = \Lambda(x_i) \\ x_i & = i \times \Delta x \end{cases} \tag{1.7}$$

Les figures (1.5a) et (1.5b) illustrent l' interpolation de $\underline{\Lambda}$ à l'aide de fonctions de forme linéaires par morceaux. On définit les fonctions de formes $\Phi_i(x_i)$ tel que:

$$\begin{cases} \Phi_i(x_i) & = 1 \\ \Phi_i(x) & = \frac{x - x_{i-1}}{\Delta x} \text{ , si } x \in [x_{i-1}, x_i] \\ \Phi_i(x) & = \frac{x_{i+1} - x}{\Delta x} \text{ , si } x \in [x_i, x_{i+1}] \\ \Phi_i(x) & = 0 \text{ , sinon} \end{cases} \tag{1.8}$$



(a) Fonctions de formes linéaires par morceaux.



(b) Champs spatial discrétisé et approximé.

Figure 1.5: Interpolation de $\underline{\Lambda}(x)$ à l'aide de fonctions de formes $\Phi_i(x_i)$.

Ainsi, l'intégral de produit de 2 fonctions $a(x)$ et $b(x)$ dépendantes de x est donné par:

$$\int_0^L a(x)\, b(x)\, dx = \sum_{i=0}^{N_x - 1} \int_{x_i}^{x_{i+1}} a(x)\, b(x)\, dx \tag{1.9}$$

où:

$$\begin{cases} a(x) & = \sum_{j=1}^{N_x} \Phi_j(x)\, a_j \\ b(x) & = \sum_{j=1}^{N_x} \Phi_j(x)\, b_j \end{cases} \tag{1.10}$$

En utilisant l'équation (1.8) sur l'intervalle $[x_i, x_{i+1}]$, l'intégral dans l'équation (1.9) devient:

$$\sum_{i=0}^{N_x-1} \int_{x_i}^{x_{i+1}} (a_i\ \Phi_i + a_{i+1}\ \Phi_{i+1}) \times (b_i\ \Phi_i + b_{i+1}\ \Phi_{i+1})\ dx \tag{1.11}$$

avec:

$$\int_{x_i}^{x_{i+1}} (a_i\Phi_i + a_{i+1}\Phi_{i+1})\ (b_i\Phi_i + b_{i+1}\Phi_{i+1})\ dx = \begin{pmatrix} a_i \\ a_{i+1} \end{pmatrix}^T \int_{x_i}^{x_{i+1}} \begin{pmatrix} (\Phi_i)^2 & \Phi_i\ \Phi_{i+1} \\ \Phi_i\ \Phi_{i+1} & (\Phi_{i+1})^2 \end{pmatrix} dx \begin{pmatrix} b_i \\ b_{i+1} \end{pmatrix} \tag{1.12}$$

La matrice élémentaire $\underline{\underline{I_{x,ele}}}$ est donnée dans l'équation ci-dessous:

$$\underline{\underline{I_{x,ele}}} = \int_{x_i}^{x_{i+1}} \begin{pmatrix} (\Phi_i)^2 & \Phi_i\ \Phi_{i+1} \\ \Phi_i\ \Phi_{i+1} & (\Phi_{i+1})^2 \end{pmatrix} dx = \frac{\Delta x}{6} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \tag{1.13}$$

L'assemblage de la matrice élémentaire $\underline{\underline{I_{x,ele}}}$ induit la matrice globale $\underline{\underline{I_x}}$ donnée dans la figure (1.6). L'approximation de l'intégrale est donc donnée dans l'équation (1.14), également illustrée dans la figure (1.6).

$$\boxed{\int_0^L a(x)\ b(x)\ dx = \underline{a}^T\ \underline{\underline{I_x}}\ \underline{b}} \tag{1.14}$$

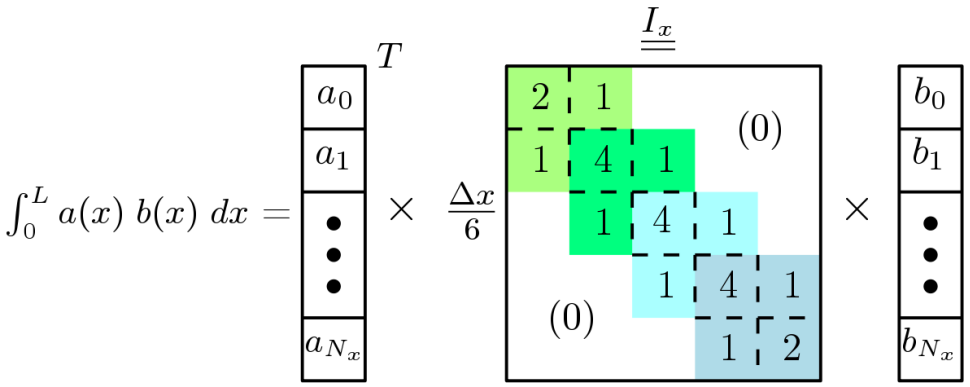


Figure 1.6: Approximation de l'intégral spatial par la méthode des éléments finis.

Calcul de l'intégral temporel

Pour calculer l'intégral temporel à l'aide des fonctions de formes linéaires par morceaux, on procède de la même manière que précédemment. Soit:

$$\underline{\Gamma}(t) = \sum_{i=1}^{N_t} \Phi_i(t)\, \Gamma_i \tag{1.15}$$

avec:

$$\begin{cases} \Gamma_i & = \Gamma(t_i) \\ t_i & = i \times \Delta t \end{cases} \tag{1.16}$$

La figures (1.7) illustre l' interpolation de $\underline{\Gamma}$ à l'aide de fonctions de forme linéaires par morceaux. On définit les fonctions de formes $\Phi_i(t_i)$ tel que:

$$\begin{cases} \Phi_i(t_i) & = 1 \\ \Phi_i(t) & = \frac{t - t_{i-1}}{\Delta t} \text{ , si } t \in [t_{i-1}, t_i] \\ \Phi_i(t) & = \frac{t_{i+1} - t}{\Delta t} \text{ , si } t \in [t_i, t_{i+1}] \\ \Phi_i(t) & = 0 \text{ , sinon} \end{cases} \tag{1.17}$$

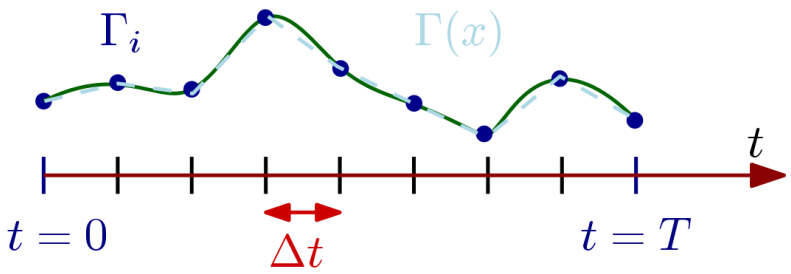


Figure 1.7: Interpolation de $\underline{\Lambda}(t)$ à l'aide de fonctions de formes $\Phi_i(t_i)$.

Ainsi, l'intégral de produit de 2 fonctions $a(t)$ et $b(t)$ dépendantes de t est donné par:

$$\int_0^T a(t)\, b(t)\, dt = \sum_{i=0}^{N_t - 1} \int_{t_i}^{t_{i+1}} a(t)\, b(t)\, dt \tag{1.18}$$

où:

$$\begin{cases} a(t) & = \sum_{j=1}^{N_t} \Phi_j(t)\ a_j \\ b(t) & = \sum_{j=1}^{N_t} \Phi_j(t)\ b_j \end{cases} \tag{1.19}$$

En utilisant l'équation (1.17) sur l'intervalle $[t_i, t_{i+1}]$, l'intégral dans l'équation (1.18) devient:

$$\sum_{i=0}^{N_t-1} \int_{t_i}^{t_{i+1}} (a_i\ \Phi_i + a_{i+1}\ \Phi_{i+1}) \times (b_i\ \Phi_i + b_{i+1}\ \Phi_{i+1})\ dt \tag{1.20}$$

avec:

$$\int_{t_i}^{t_{i+1}} (a_i\Phi_i + a_{i+1}\Phi_{i+1})\ (b_i\Phi_i + b_{i+1}\Phi_{i+1})\ dt = \begin{pmatrix} a_i \\ a_{i+1} \end{pmatrix}^T \int_{t_i}^{t_{i+1}} \begin{pmatrix} (\Phi_i)^2 & \Phi_i\ \Phi_{i+1} \\ \Phi_i\ \Phi_{i+1} & (\Phi_{i+1})^2 \end{pmatrix} dt \begin{pmatrix} b_i \\ b_{i+1} \end{pmatrix} \tag{1.21}$$

La matrice élémentaire $\underline{\underline{I_{t,ele}}}$ devient:

$$\underline{\underline{I_{t,ele}}} = \int_{t_i}^{t_{i+1}} \begin{pmatrix} (\Phi_i)^2 & \Phi_i\ \Phi_{i+1} \\ \Phi_i\ \Phi_{i+1} & (\Phi_{i+1})^2 \end{pmatrix} dt = \frac{\Delta t}{6} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \tag{1.22}$$

L'assemblage de la matrice élémentaire $\underline{\underline{I_{t,ele}}}$ induit la matrice globale $\underline{\underline{I_t}}$ donnée dans la figure (1.5). L'approximation de l'intégrale est donc donnée dans l'équation (2.13), également illustrée dans la figure (1.10).

$$\boxed{\int_0^T a(t)\ b(t)\ dt = \underline{a}^T\ \underline{\underline{I_t}}\ \underline{b}} \tag{1.23}$$

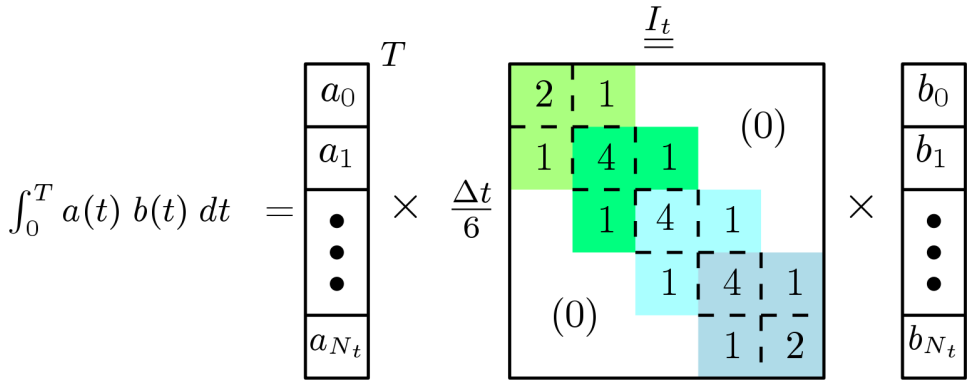


Figure 1.8: Approximation de l'intégral temporel par la méthode des éléments finis.

1.3.3 Approximation PGD: Algorithme glouton et point fixe

Le couple $(\underline{\Gamma}_i, \underline{\Lambda}_i)$ est tel que l'erreur d'approximation dans l'équation (1.5) est minimale. On définit cette erreur par l'équation ci-dessous:

$$(\underline{\Gamma}_i, \underline{\Lambda}_i) = \arg \min_{I \times W} (||\underline{\underline{U}} - \sum_{k=1}^{i-1} \underline{\Lambda}_k(x)\, \underline{\Gamma}_k^T(t) - \underline{\Lambda}_i\, \underline{\Gamma}_i^T||^2) \tag{1.24}$$

avec : $\sum_{k=1}^{i-1} \underline{\Lambda}_k(x)\, \underline{\Gamma}_k^T(t) = \underline{\underline{U}}_{PGD}^{i-1}$

Dans la suite, on se propose de résoudre le problème d'optimisation (minimisation dans notre cas) avec l'algorithme de point fixe.

Algorithme de point fixe

L'algorithme de point fixe est implémenté dans *Matlab*. Le pseudo-code pour l'approximation PGD (algorithme de point fixe) est donné ci-dessous.

Algorithm 1 Approximation PGD (Point fixe)

1: On calcule : $\underline{\underline{u}}^* \leftarrow \underline{\underline{u}}_f - \underline{\underline{u}}_{PGD}^{i-1}$

2: On initialise : $\underline{\Gamma}_i^{new} \leftarrow linspace(0, T, N_t)^T$

3: On calcule : $\underline{\Lambda}_i^{new} \leftarrow \frac{\int_I \underline{\underline{u}}^*(x,t)\, \underline{\Gamma}_i^{new}\, dt}{\int_I (\underline{\Gamma}_i^{new})^2\, dt}$

4: On normalise (unicité) : $\underline{\Lambda}_i^{new} \leftarrow \frac{\underline{\Lambda}_i^{new}}{\int_W (\underline{\Lambda}_i^{new})^2\, dx}$

5: $\underline{\Gamma}_i^{old} \leftarrow \underline{\Gamma}_i^{new}$

6: On calcule: $\underline{\Gamma}_i^{new} \leftarrow \frac{\int_W \underline{\underline{u}}^*(x,t)\, \underline{\Lambda}_i^{new}\, dx}{\int_W (\underline{\Lambda}_i^{new})^2\, dx}$

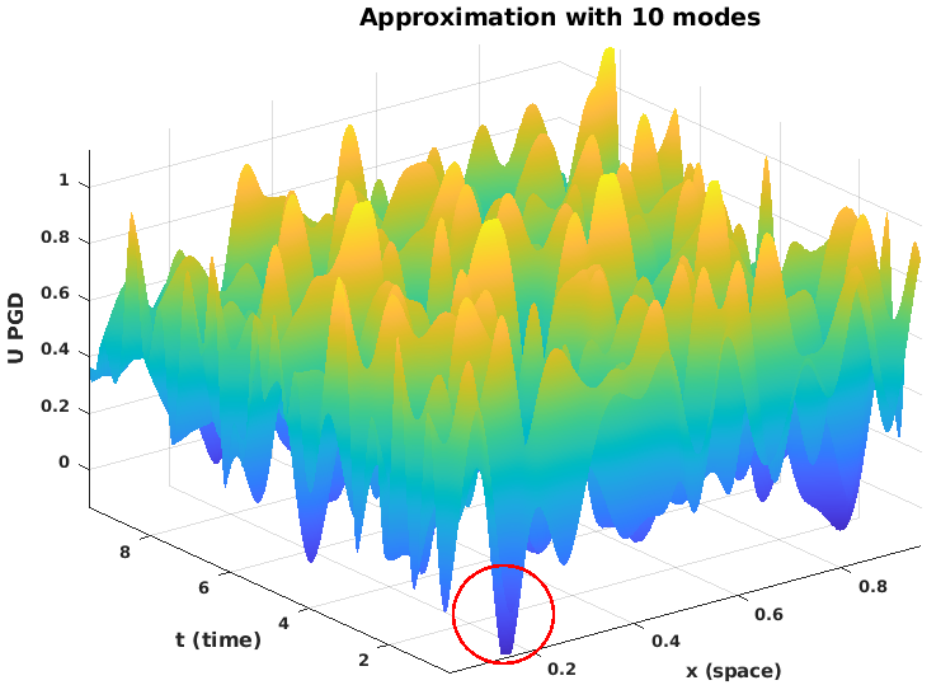
7: On calcule le critère de stagnation : $s \leftarrow \frac{\int_I (\underline{\Gamma}_i^{new} - \underline{\Gamma}_i^{old})^2\, dt}{\int_I (\underline{\Gamma}_i^{old})^2\, dt}$

8: si $(s < threshold\ \epsilon = 10^{-3})$:

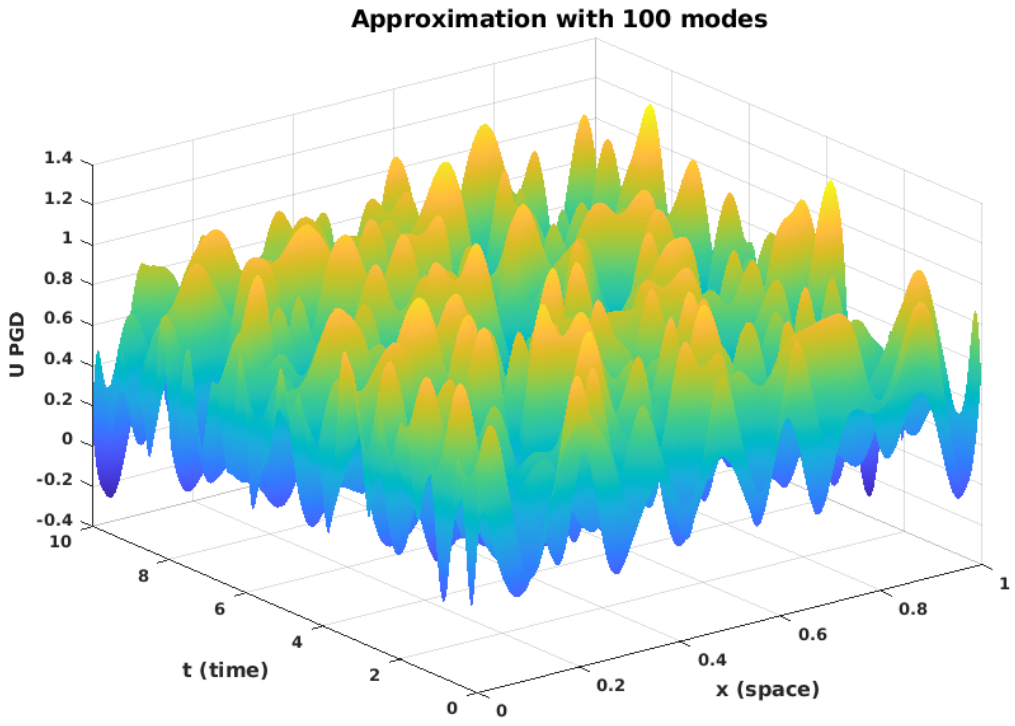
- On enregistre les modes: $\underline{\Gamma}_i \leftarrow \underline{\Gamma}_i^{new}$ et $\underline{\Lambda}_i \leftarrow \underline{\Lambda}_i^{new}$
- $\underline{\underline{u}}_{PGD} = \underline{\underline{u}}_{PGD} + \underline{\Gamma}_i\, \underline{\Lambda}_i^T$
- On mis à jour le champ à approximer : $\underline{\underline{u}}^* \leftarrow \underline{\underline{u}}^* - \underline{\Gamma}_i\, \underline{\Lambda}_i^T$
- On cherche le mode $(i+1)$.

9: sinon :

- On retourne à l'étape 3 jusqu'à convergence de l'algorithme de point fixe.



(a) Approximation du champ aléatoire avec $m = 10$ modes.



(b) Approximation du champ aléatoire avec $m = 100$ modes.

Figure 1.9: Reconstruction de la distribution de champ aléatoire en fonction du nombre de modes m.

La reconstruction de la distribution aléatoire du maillage fin illustrée dans la figure (1.4) est faite avec un nombre de modes m égal à 10 et 100, respectivement. En utilisant l'algorithme (1), on obtient les distributions illustrées dans les figures (1.9a) et (1.9b).

On peut remarquer que l'approximation avec un nombre de modes $m = 100$ est plus précise que celle avec $m = 10$. Par exemple, la partie du champ entourée en rouge n'existe pas dans la distribution initiale de la figure (1.4).

Évaluation de l'erreur d'approximation

On se propose dans cette partie de tracer l'erreur d'approximation en fonction du nombre de modes m allant de 1 à 100. Cette erreur est donnée dans l'équation (2.15).

$$error = \frac{||\underline{\underline{U}}^{ref} - \underline{\underline{U}}^{PGD}||_{W\times I}}{||\underline{\underline{U}}^{ref}||_{W\times I}} = \frac{\int_W \int_I (\underline{\underline{U}}^{ref} - \underline{\underline{U}}^{PGD})^2 \, dt \, dx}{\int_W \int_I (\underline{\underline{U}}^{ref})^2 \, dt \, dx} \tag{1.25}$$

Pour calculer cette erreur, on procède comme suit:

- On intègre sur I. Par conséquent, on obtient un vecteur de l'ordre N_x
- On intègre sur W. Par conséquent, on obtient un scalaire représentant l'erreur pour chaque mode.

L'évolution de l'erreur d'approximation en fonction du nombre de modes m est donné dans la figure ci-dessous. Elle est décroissante en fonction du nombre de modes.

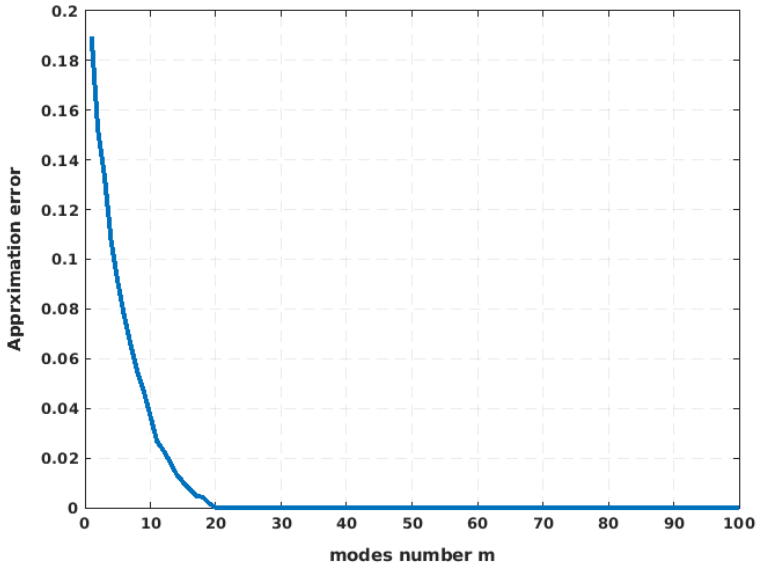


Figure 1.10: Erreur d'approximation du champ aléatoire en fonction du nombre de modes.

Orthogonalisation des modes avec Gram-Schmidt

L'orthogonalisation des modes est nécessaire pour éviter la redondance de l'information et minimiser l'erreur d'approximation. Ceci est possible à l'aide de l'algorithme de *Gram-Schmidt* donné dans l'algorithme (2), également implémenté dans *Matlab*.

Algorithm 2 Orthogonalisation des modes spatiales avec *Gram-Schmidt*

1: On supprime de l'information redondante : $\underline{\Lambda}_{m+1} \leftarrow \underline{\Lambda}_{m+1}^{new} - \sum_{i=1}^{m} (\underline{\Lambda}_{m+1}^{new}, \underline{\Lambda}_i)\ \underline{\Lambda}_i$

2: On ajoute de l'information : $\underline{\Gamma}_i \leftarrow \underline{\Gamma}_i + \underline{\Gamma}_{m+1}\ (\underline{\Lambda}_{m+1}^{new}, \underline{\Lambda}_i),\ \forall i \in [1, m]$

3: On normalise : $\underline{\Lambda}_i^{new} \leftarrow \frac{\underline{\Lambda}_{m+1}^{new}}{\sqrt{\int_W (\underline{\Lambda}_{m+1})^2\ dx}}$

4: On normalise : $\underline{\Gamma}_i^{new} \leftarrow \underline{\Gamma}_{m+1}^{new} \times \sqrt{\int_W (\underline{\Lambda}_{m+1})^2\ dx}$

L'évolution de l'erreur d'approximation en fonction du nombre de modes m après orthogonalisation avec l'algorithme de *Gram-Schmidt* est donnée dans la figure ci-dessous. On remarque également une décroissance en fonction du nombre de modes m.

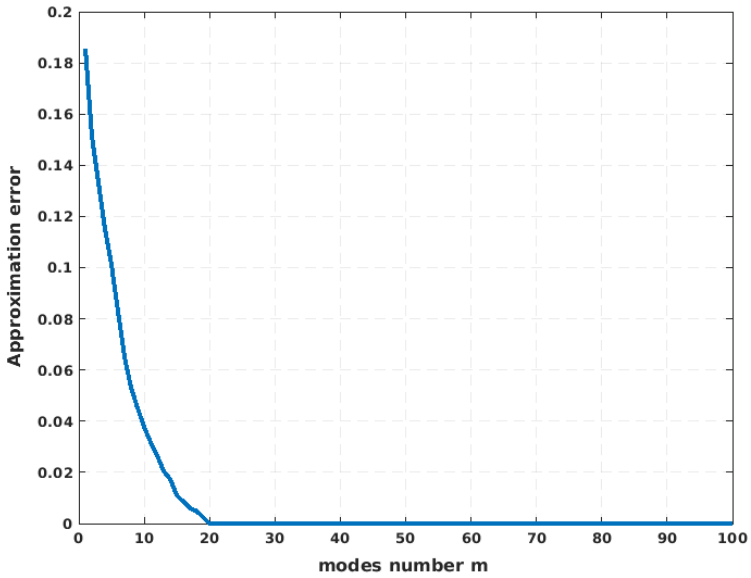


Figure 1.11: Erreur d'approximation du champ aléatoire en fonction du nombre de modes, en considérant l'orthogonalisation des modes.

On constate d'après la figure (1.11) que l'erreur est la même que celle obtenue dans la figure (1.10), où la phase d'orthogonalisation n'est pas ajoutée. Une comparaison des erreurs pour m variant de 1 à 100 est illustrée dans la figure (1.12).

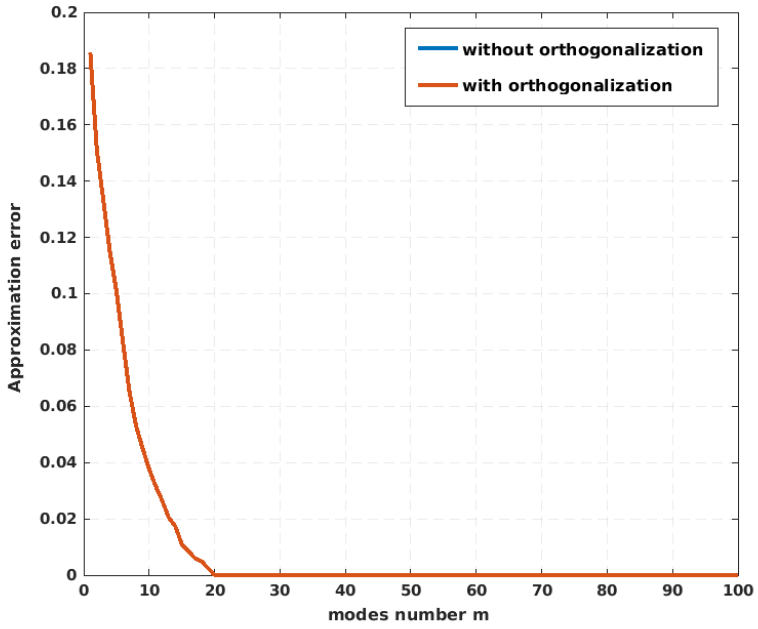


Figure 1.12: Comparaison de l'erreur d'approximation du champ aléatoire en fonction du nombre de modes sans et avec l'orthogonalisation des modes avec *Gram-Schmidt*.

Généralement, l'ajout de la phase d'orthogonalisation des modes permet de contrôler l'erreur d'approximation. Cependant, l'orthogonalisation des modes n'est pas utile ici. La reconstruction du champs aléatoire avec un nombre de modes $m = 100$ et en considérant l'orthogonalité avec *Gram-Schmidt* est illustré dans la figure (1.13). On peut constater que le PGD pour $m = 100$ approxime très bien le champs spatio-temporel.

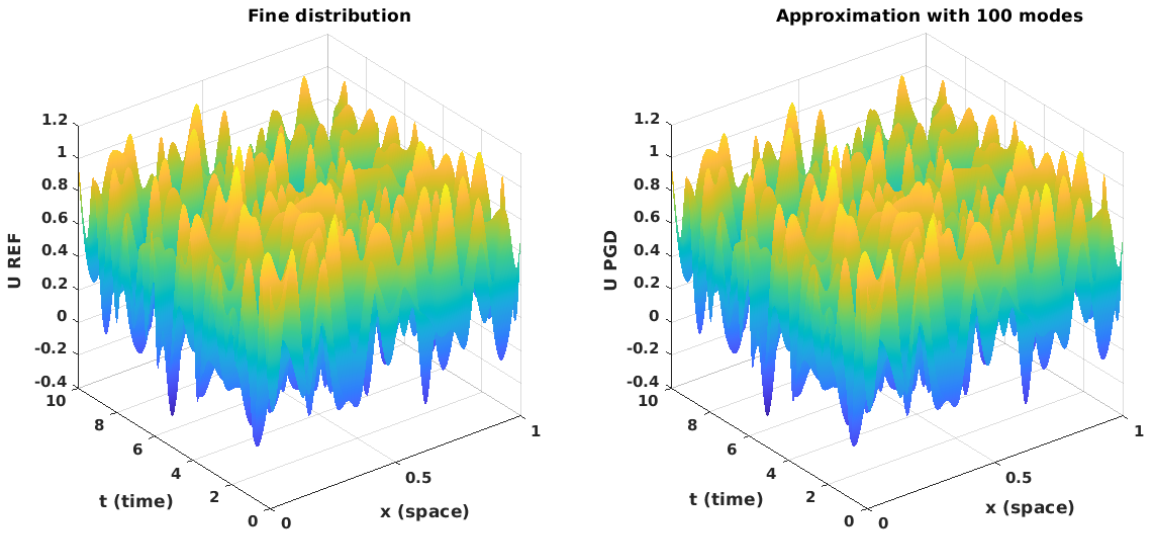


Figure 1.13: Reconstruction du champ aléatoire avec $m = 100$ modes.

1.3.4 Application à l'image de *Maria Callas*

On s'intéresse dans cette partie à comparer les deux algorithmes *POD* avec *SVD* et *PGD* (algorithme glouton avec point fixe) dans le cas ou notre donnée d'entrée est l'image de *Maria Callas*. Cette dernière est stockée dans une matrice de pixels M de taille $n_T \times n_M$. Les étapes à suivre pour comprimer l'image avec *SVD* et *PGD* sont données ci-dessous:

- Chargement de l'image à partir du fichier,
- Obtention des dimensions de l'image n_T et n_M,
- Conversion de l'image en une matrice de pixels M de dimensions $n_T \times n_M$,
- Calcul des matrices élémentaires Ix et It pour l'approximation de l'intégration,
- Appel de la fonction *GreedyOrthogonalization* pour la reconstruction de l'image avec *PGD* orthogonalisé avec *Gram-Schmidt*,
- Calcul de l'erreur d'approximation de l'algorithme *PGD* en fonction du nombre de modes m,
- Reconstruction de l'image avec *SVD* de *Matlab*, comme dans la première partie,
- Calcul de l'erreur d'approximation de l'algorithme *SVD* en fonction du nombre de modes m.

En fixant le nombre de modes m à 20, on obtient les images recnstruites avec *SVD* et *PGD* orthogonalisé illustrées dans la figure (1.14).

Figure 1.14: Reconstruction de l'image avec les algorithmes *SVD* et *PGD* orthogonalisé.

La taille de l'image originale est évalué à $87.6KB$. Après compression, les nouvelles tailles obtenues de l'image sont:

- **avec l'algorithme *SVD*** : 28.9 $kB \Rightarrow$ gain de compression : 3.03
- **avec l'algorithme *PGD* orthogonalisé** : 28.7 $kB \Rightarrow$ gain de compression : 3.05

On varie le nombre de modes m de 1 à 20, la comparison des erreurs d'approximation issues des deux algorithmes SVD et PGD orthogonalisé est illustrée dans la figure (1.16).

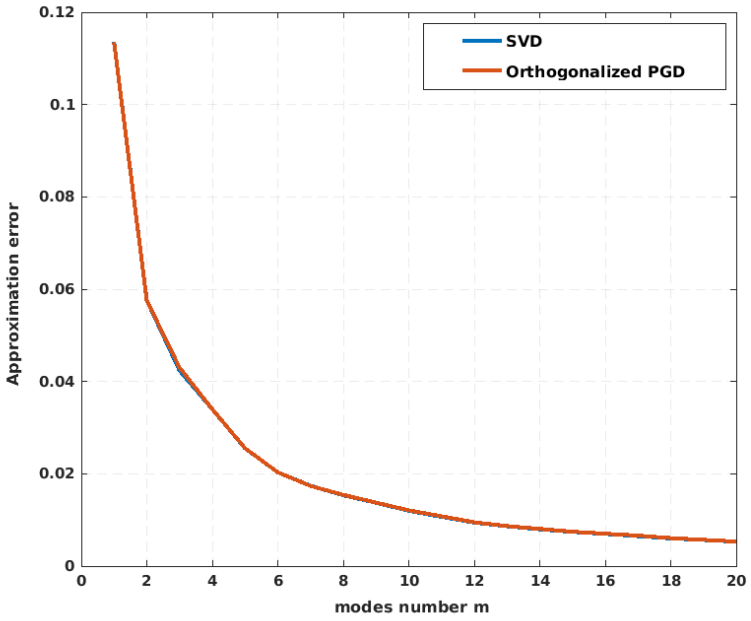


Figure 1.15: Comparaison des erreurs d'approximation de l'image pour les algorithmes SVD et PGD orthogonalisé pour un nombre de modes m allant de 1 à 20.

Un zoom sur une partie de la figure (1.16) nous permet de visualiser que l'erreur d'approximation en fonction du nombre de modes m est un peu plus faible en utilisant l'algorithme SVD. Néanmoins, les courbes montrant l'évolution des erreurs d'approximation en fonction de m issues des deux algorithmes sont à-peu-près superposées.

Les deux algorithmes SVD et PGD sont donc efficaces pour la compression de l'image. On a pu obtenir un gain de compression égal à 3 avec un nombre de modes $m = 20$. Le choix de ce nombre de modes permet de conserver les information importantes (figure(1.14)), tout en minimisant l'erreur d'approximation (de l'ordre de 0.0053).

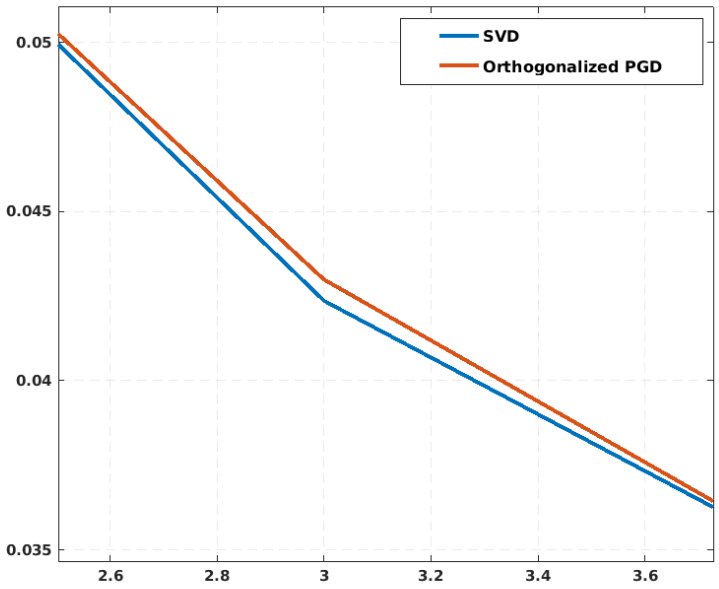


Figure 1.16: Zoom sur une partie de la comparaison des erreurs d'approximation de l'image pour les algorithmes *SVD* et *PGD* orthogonalisé en fonction du nombre de modes.

1.4 Conclusion

Dans ce premier TP, on a pu reconstruire une image donnée et un champ dépendant de deux variables avec un nombre de modes afin de les comprimer.

Deux algorithmes sont utilisés : *SVD* et *PGD*. Le dernier algorithmes est implémenté dans deux fonctions différentes dans *Matlab* représentants ses deux versions: (i) simple sans orthogonalisation et (ii) orthogonalisé avec *Gram-Schmidt*.

Le choix de nombre de modes est effectué de telle sorte que l'erreur d'approximation de l'image/champ soit minimisée et le gain de compression est bon.

Chapter 2

TP 2-Résolution d'une EDP par réduction de modèles

2.1 Introduction

On s'intéresse dans ce TP de résoudre et réduire une équation différentielle, populaire dans la mécanique de structure. Pour ce faire, on commence par définir l'espace fonctionnel et écrire la forme variationnelle de l'équation. Après discrétisation, l'équation est résolue avec l'algorithme de force brute. Le déplacement est ensuite réduit avec le *PGD*.

2.2 Problématique

2.2.1 EDP

On pose le système d'équation à résoudre donné dans l'équation (2.1).

$$\begin{cases} -\Delta u \;=\; f & (I \times \Omega) \\ u \;=\; u_d & (I \times \partial\Omega) \end{cases} \tag{2.1}$$

On propose dans la suite de la résoudre en utilisant l' algorithme de force brute.

2.2.2 Forme variationnelle

On définit l'espace fonctionnel tel que:

$$\begin{cases} U = \{v |\ v \text{ régulier}, v = u_d,\ I \times \partial\Omega\} \\ U0 = \{v |\ v \text{ régulier}, v = 0,\ I \times \partial\Omega\} \end{cases} \tag{2.2}$$

La forme variationnelle issue de l'équation (2.1) consiste à trouver $u \in U$ vérifiant:

$$a(u, v) = l(v)\ \ ,\ \ \forall v \in U0 \tag{2.3}$$

Pour se faire, on intègre l'équation (2.1) sur la variable espace Ω:

$$\int_\Omega -\Delta u\ v\ d\Omega = \int_\Omega f\ v\ d\Omega \tag{2.4}$$

Le calcul de l'intégral ($\int_\Omega \Delta u\ v\ d\Omega$) est possible avec l'intégration par partie, en posant:

$$\begin{cases} (func_1)' = \frac{\partial^2 u}{\partial \Omega^2} \ \Rightarrow\ func_1 = \frac{\partial u}{\partial \Omega} \\ func_2 = v \ \Rightarrow\ (func_2)' = \frac{\partial v}{\partial \Omega} \end{cases} \tag{2.5}$$

On obtient:

$$\int_\Omega \Delta u\ v\ d\Omega = [func_1 \times func_2]_\Omega - \int_\Omega func_1 \times (func_2)'\ d\Omega \tag{2.6}$$

Ce qui conduit à:

$$a(u, v) = \int_\Omega -\Delta u\ v\ d\Omega = \int_\Omega -\underline{\nabla} u\ \underline{\nabla} v\ d\Omega \tag{2.7}$$

On en déduit la forme variationnelle en utilisant les équations (2.4) et (2.7):

$$\boxed{\int_\Omega -\underline{\nabla} u\ \underline{\nabla} v\ d\Omega = \int_\Omega f\ v\ d\Omega\ \ ,\ \ \forall v \in U0} \tag{2.8}$$

2.3 Résolution avec l'algorithme de force brute

2.3.1 Discrétisation spatiale et temporelle

On s'intéresse dans cette partie à la discrétisation spatiale et temporelle pour pouvoir utiliser l'algorithmes de force brute. On définit les intervalles d'étude suivantes:

$$\begin{cases} \Omega = [0, L] \\ I = [0, T] \end{cases} \tag{2.9}$$

Discrétisation spatiale

Afin de faire la discrétisation spatiale, on considères les paramètres suivante:

Paramètre	**Signification**	**Valeur**
L	Longueur maximale	10
n_x	nombre de points selon x	1000
l_x	intervalle spatiale	$0 : \frac{L}{n_x - 1} : L$

Discrétisation temporelle

Afin de faire la discrétisation temporelle, on considères les paramètres suivante:

Paramètre	**Signification**	**Valeur**
T	temps maximal	1
n_t	nombre de points selon t	100
l_t	intervalle temporelle	$0 : \frac{T}{n_t - 1} : L$

2.3.2 Conditions limites et force

On définit dans la première simulation le déplacement prescrit à $x = 0$ et $x = L$ tel que:

$$\begin{cases} u_{d0} = 8\ sin(\frac{2\,\Pi\,l_t}{T}) \ \ , \ \ x = 0 \\ u_{dL} = -5\ sin(\frac{4\,\Pi\,l_t}{T}) \ \ , \ \ x = L \end{cases} \tag{2.10}$$

On considère aussi une force nulle $f(t,x) = 0$ représentée par une matrice nulle $\underline{\underline{f}}$ de taille $n_x \times n_t$.

2.3.3 Calcul de l'intégral par la méthode des éléments finis

On considère la forme variationnelle donnée dans l'équation (2.8). On s'intéresse dans cette partie au calcul de l'intégral $\int_0^L \frac{\partial u}{\partial x} \frac{\partial v}{\partial x}\, dx$ par la méthode des éléments finis. Pour ce faire, on utilise les fonctions de formes linéaires par morceaux. L'intégral s'écrit comme la contributions des éléments de l'intervalle spatiale.

$$\int_0^L \frac{\partial u}{\partial x} \frac{\partial v}{\partial x}\, dx = \sum_{i=0}^{n_x-1} \int_{x_i}^{x_j} \frac{\partial u}{\partial x} \frac{\partial v}{\partial x}\, dx \tag{2.11}$$

La fonction $u(x)$ peut être discrétisée et approximée par une fonction linéaire par morceaux comme illustré dans la figure (2.1). On considère l'intervalle $[x_i, x_j]$ de longueur $\Delta x = \frac{L}{n_x - 1}$ correspondant à un élément.

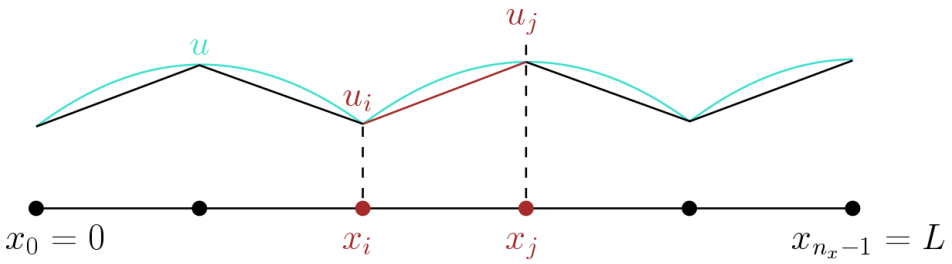


Figure 2.1: Interpolation de $u(x)$ à l'aide de fonctions de formes $\varphi_i(x_i)$.

Ainsi, $u(x)$ élémentaire s'écrit dans l'intervalle $[x_i, x_j]$ comme:

$$u(x) = u_i\, \varphi_i(x) + u_j\, \varphi_j(x) \tag{2.12}$$

On remplace l'équation (2.12) dans (2.11). Ce qui conduit à l'équation (2.13).

$$\int_0^L \frac{\partial u}{\partial x} \frac{\partial v}{\partial x}\, dx = \sum_{i,j} \int_{x_i}^{x_j} \frac{\partial}{\partial x} \left(u_i\, \varphi_i(x) + u_j\, \varphi_j(x)\right) \frac{\partial}{\partial x} \left(v_i\, \varphi_i(x) + v_j\, \varphi_j(x)\right) dx \tag{2.13}$$

L'écritue matricielle est tel que:

$$\sum_{i,j} \begin{bmatrix} u_i \\ u_j \end{bmatrix}^T \int_{x_i}^{x_j} \begin{bmatrix} \varphi_i'(x)^2 & \varphi_i'(x)\ \varphi_j'(x) \\ \varphi_i'(x)\ \varphi_j'(x) & \varphi_j'(x)^2 \end{bmatrix} dx \begin{bmatrix} v_i \\ v_j \end{bmatrix} \tag{2.14}$$

avec:

- $\varphi_i(x) = 1 - \frac{x}{\Delta x}$
- $\varphi_j(x) = \frac{x}{\Delta x}$

L'équation (2.14) devient:

$$\sum_{i,j} \begin{bmatrix} u_i \\ u_j \end{bmatrix}^T \int_0^{\Delta x} \frac{1}{\Delta x^2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} dx \begin{bmatrix} v_i \\ v_j \end{bmatrix} \tag{2.15}$$

Après assemblage, on obtient:

$$\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n_x-1} \\ u_{n_x} \end{pmatrix}^T \frac{1}{\Delta x} \begin{pmatrix} 1 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ & & & \ddots & & \\ 0 & \cdots & 0 & -1 & 2 & -1 \\ 0 & \cdots & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n_x-1} \\ v_{n_x} \end{pmatrix} \tag{2.16}$$

On conclut à partir de l'équation (2.16) que $a(u, v)$ s'écrit sous forme matricielle discrétisée comme suit:

$$\boxed{a(u,v) = \int_\Omega -\underline{\nabla} u\ \underline{\nabla} v\ d\Omega = \int_0^L \frac{\partial u}{\partial x}\ \frac{\partial v}{\partial x}\ dx = \underline{U}^T\ \underline{\underline{K}}\ \underline{V}} \tag{2.17}$$

avec:

- $\underline{\underline{K}}$: la matrice de rigidité assemblée de taille $n_x \times n_x$
- $\underline{U}$: vecteur colonne d'ordre n_x
- $\underline{V}$: vecteur colonne d'ordre n_x

De plus, le terme de droite dans l'équation (2.4) s'écrit:

$$l(v) = \int_0^L f(t,x)\, v(x)\, dx = \int_0^L \underline{\underline{f}}\, \underline{v}\, dx \tag{2.18}$$

Finalement, on obtient:

$$\boxed{l(v) = \underline{V}^T\, \underline{\underline{F}}} \tag{2.19}$$

avec: $\underline{\underline{F}} = \underline{\underline{I_x}}\, \underline{\underline{f}}$ une matrice de dimension $n_x \times n_t$, où $\underline{\underline{I_x}}$ est la matrice d'intégration calculée dans le premier chapitre (équation (1.14)).

Le problème devient ainsi de trouver $\underline{u}(t) \in \mathrm{U}$ tel que:

$$\underline{V}^T\, \underline{\underline{K}}\, \underline{U} = \underline{V}^T\, \underline{\underline{F}} \tag{2.20}$$

La résolution de l'équation (2.20) est possible à l'aide de l'algorithme de force brute.

2.3.4 Algorithme de force brute

Afin d'utiliser l'algorithme de force brute, on organise le système dans l'équation (2.20) de telle sorte que les unconnus et les conditions limites soient séparés, comme illustré dans la figure (2.2). Ceci nous permet de calculer les indices de degrés de libertés correspondant.

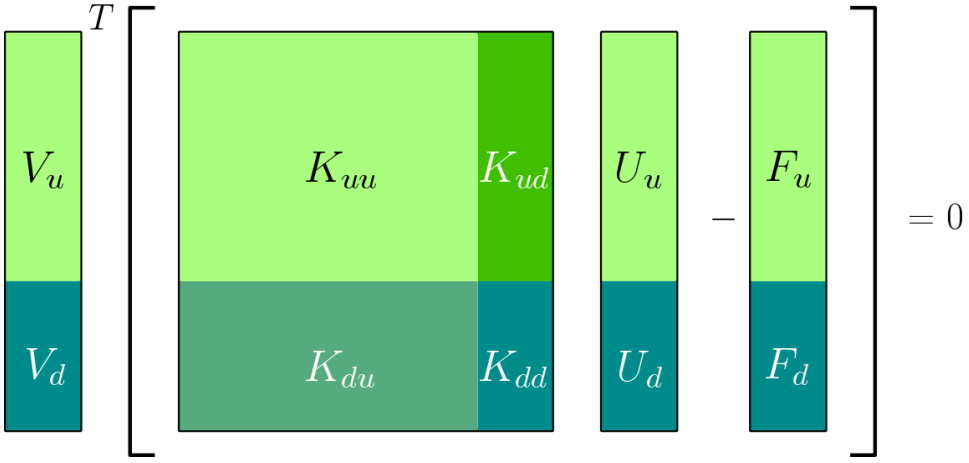


Figure 2.2: Système à résoudre avec l'algorithme de force brute.

D'après la figure (2.2), la résolution du déplacement inconnue est donnée par: $U_u = K_{uu}^{-1}(F_u - K_{ud}\, U_d)$

Résultats de la simulation avec l'algorithme de force brute

On s'intéresse dans cette partie à tracer le déplacement obtenu en fonction du temps et de l'espace. On impose le déplacement prescrit à $x = 0$ et $x = L$ donné dans l'équation (2.10). Cependant, on varie la force $\underline{\underline{f}}$ dans les deux simulations exécutées.

Dans la première simulation, la matrice de force $\underline{\underline{f}}$ est nulle, alors qu'elle est égale à $5 \times sin(\frac{3\,\pi\,l_x}{L})^T \times sin(\frac{2\,\pi\,l_t}{T})$ dans la deuxième simulation. Les résultats de déplacement obtenus sont illustrés dans les figure (2.3) et (2.4).

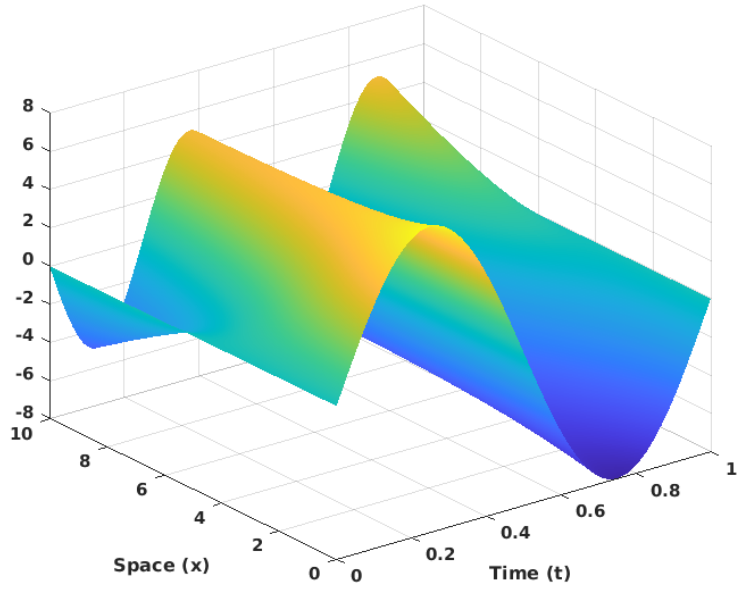


Figure 2.3: Variation du déplacement en fonction du temps et de l'espace pour $\underline{\underline{f}}$ nulle.

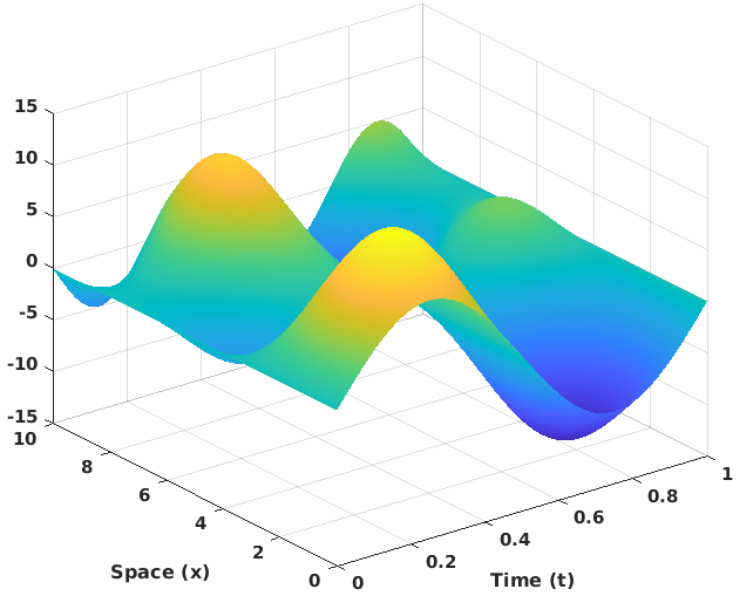


Figure 2.4: Variation du déplacement en fonction du temps et de l'espace pour $\underline{\underline{f}} = 5 \times sin(\frac{3\,\pi\,l_x}{L})^T \times sin(\frac{2\,\pi\,l_t}{T})$

2.4 Réduction de modèles avec l'algorithme PGD

On a pu écrire le problème donné dans l'équation (2.1) sous forme matricielle discrétisée comme indiqué dans l'équation (2.20). Ceci est équivalent à:

$$\underline{V}^T \, (\underline{\underline{K}} \, \underline{U}(t) - \underline{\underline{F}}(t)) = 0 \ , \ \ \forall V \in \mathrm{U}0 \tag{2.21}$$

On peut réécrire $\underline{U}(t)$ en utilisant la séparation de variables tel que:

$$\underline{U}(t) = \sum_i \lambda_i(t) \, \underline{\Lambda}_i + \underline{U}_{CL}(t) = \underline{W}(t) + \underline{U}_{CL}(t) \tag{2.22}$$

où : $\underline{W}(t) \in \mathrm{U}0$ et $\underline{U}_{CL}(t) \in \mathrm{U}$ représentant les conditions limites.

En remplaçant l'équation (2.22) dans (2.23), l'équation devient:

$$\boxed{\underline{V}^T \, (\underline{\underline{K}} \, \underline{W}(t) - \underline{G}(t)) = 0 \ , \ \ \forall V \in \mathrm{U}0} \tag{2.23}$$

avec:

$$\boxed{\underline{G}(t) = \underline{\underline{F}}(t) - \underline{\underline{K}} \, \underline{U}_{CL}(t)} \tag{2.24}$$

On pose le déplacement prescrit $U_{CL}(t, x)$ tel que:

$$U_{CL}(t, x) = (1 - \frac{x}{L}) \, u_d^0(t) + \frac{x}{L} \, u_d^L(t) \tag{2.25}$$

Sous forme discrétisée, on aura:

$$\begin{cases} \underline{\underline{U}}_{CL} = (1 - \frac{l_x}{L})^T \times \underline{u}_d^0 + (\frac{l_x}{L})^T \times \underline{u}_d^L \\ \underline{\underline{G}} = \underline{\underline{F}} - \underline{\underline{K}} \times \underline{\underline{U}}_{CL} \end{cases} \tag{2.26}$$

2.4.1 Séparation de variables et algorithme glouton

Le système à résoudre s'écrit sous la forme:

$$\int_0^T \underline{V}^T \, (\underline{\underline{K}} \, \underline{W}(t) - \underline{G}(t)) \, dt = 0 \;\; , \;\; \forall V \in U0 \tag{2.27}$$

avec:

$$\begin{cases} \underline{W}(t) \approx \lambda(t) \, \underline{\Lambda} \\ \underline{V} = \lambda^* \, \underline{\Lambda} + \lambda \, \underline{\Lambda}^* \end{cases} \tag{2.28}$$

On remplace l'équation (2.28) dans (2.27), ce qui conduit à **l'équation de *Galerkin.***

$$\boxed{\int_I (\lambda^* \, \underline{\Lambda} + \lambda \, \underline{\Lambda}^*)^T \, (\underline{\underline{K}} \, \lambda \, \underline{\Lambda} - \underline{G}) \, dt = 0 \;\; , \;\; \forall V \in U0} \tag{2.29}$$

Le système (2.30) s'en découle:

$$\begin{cases} \int_0^T \lambda^* \, \underline{\Lambda}^T \, (\underline{\underline{K}} \, \lambda \, \underline{\Lambda} - \underline{G}) \, dt = 0 \\ \int_0^T \lambda \, \underline{\Lambda}^{*T} \, (\underline{\underline{K}} \, \lambda \, \underline{\Lambda} - \underline{G}) \, dt = 0 \end{cases} \tag{2.30}$$

équivalent à, $\forall \underline{\Lambda}^* \in U0$:

$$\begin{cases} \underline{\Lambda}^T \, \underline{\underline{K}} \, \underline{\Lambda} \, \underline{\lambda}(t) = \underline{\Lambda}^T \, \underline{G}(t) \\ \underline{\Lambda}^{*T} \, (\int_0^T \lambda(t)^2 \, dt \, \underline{\underline{K}} \, \underline{\Lambda} - \int_0^T \lambda(t) \, \underline{G}(t) \, dt) = 0 \end{cases} \tag{2.31}$$

Soit finalement:

$$\begin{cases} \lambda(t) = \frac{l(t)}{k} \\ \underline{\Lambda}^{*T} \, (\underline{\underline{H}} \, \underline{\Lambda} - \underline{J}) = 0 \end{cases} \tag{2.32}$$

avec:

$$\begin{cases} l(t) = \underline{\Lambda}^T \, \underline{G}(t) \\ k = \underline{\Lambda}^T \, \underline{\underline{K}} \, \underline{\Lambda} \\ \underline{\underline{H}} = \int_0^T \lambda(t)^2 \, dt \, \underline{\underline{K}} = \underline{\lambda} \, \underline{\underline{I_t}} \, \underline{\lambda}^T \, \underline{\underline{K}} \\ \underline{J} = \int_0^T \lambda(t) \, \underline{G}(t) \, dt = \underline{\lambda} \, \underline{\underline{I_t}} \, \underline{G}^T \end{cases} \tag{2.33}$$

Le couple $(\lambda(t), \Lambda)$ est à résoudre avec la méthode de point fixe, comme dans le chapitre précédant.

2.4.2 Implémentation de PGD et PGD-orthogonalisé

Les deux versions de l'algorithme *PGD* sont implémentées sous *Matlab*: (i) Le *PGD* sans orthogonalisation, (ii) Le *PGD* avec orthogonalisation (*Gram-Schmidt*). Quelques résultats de simulations en variant la force $\underline{\underline{f}}$ sont illustrées ci-dessous. La première simulation (figure (2.5)) est comparée au déplacement de référence dans la figure (2.3), pour $\underline{\underline{f}}$ nulle. La deuxième simulation (figure (2.6)) est aussi comparée au déplacement de référence dans la figure (2.4), pour $\underline{\underline{f}} = 5 \times sin(\frac{3\,\pi\,l_x}{L})^T \times sin(\frac{2\,\pi\,l_t}{T})$ avec $n_{modes} = 20$.

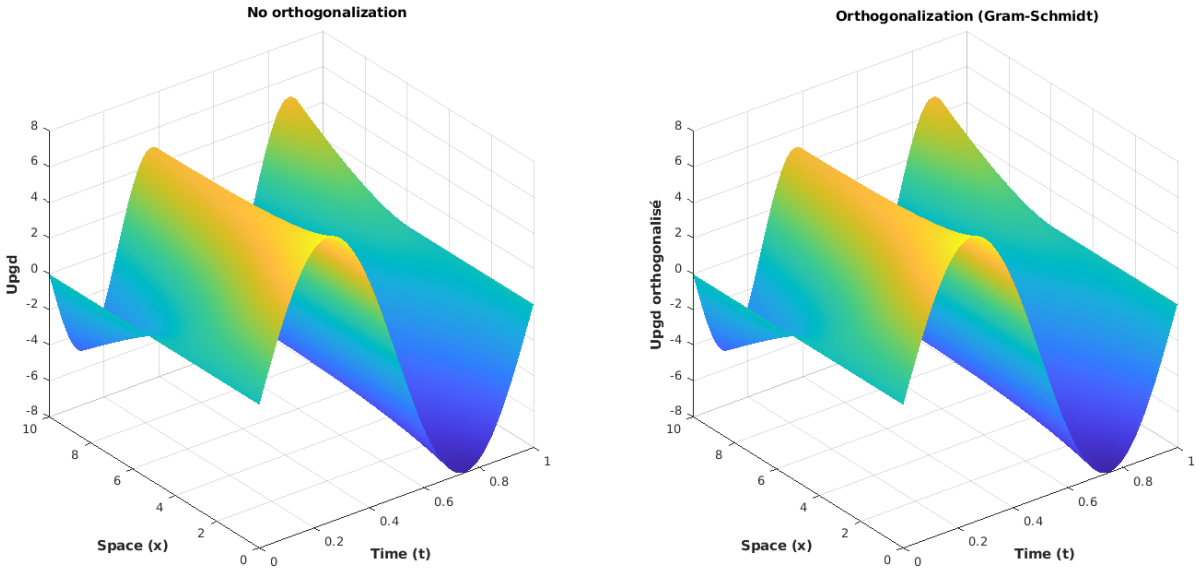


Figure 2.5: Comparaison de l'approximation de déplacement en fonction de t et x pour $\underline{\underline{f}}$ nulle. **Legend**: (gauche): PGD sans orthogonalisation, (droite): PGD orthogonalisé.

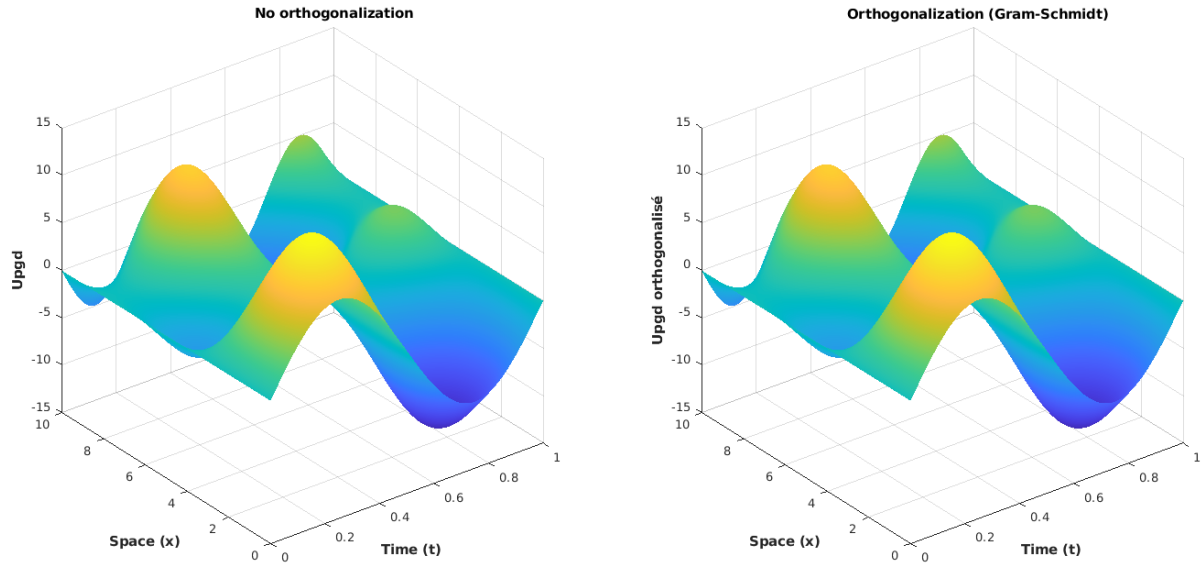


Figure 2.6: Comparaison de l'approximation de déplacement en fonction du temps et de l'espace pour $\underline{\underline{f}} = 5 \times sin(\frac{3\,\pi\,l_x}{L})^T \times sin(\frac{2\,\pi\,l_t}{T})$. **Legend**: (gauche): PGD sans orthogonalisation, (droite): PGD orthogonalisé (nombre de modes = 20).

Une dernière simulation est ensuite exécutée dans le cas où $\underline{\underline{f}} = 3 \times cos(\frac{3\,(\pi/3)\,l_x}{L})^T \times cos(\frac{2\,(\pi/3)\,l_t}{T})$. Les résultats du déplacement de référence et celui approximé avec l'algorithme de PGD et PGD-orthogonalisé sont montrés ci-dessous pour $n_{modes} = 20$.

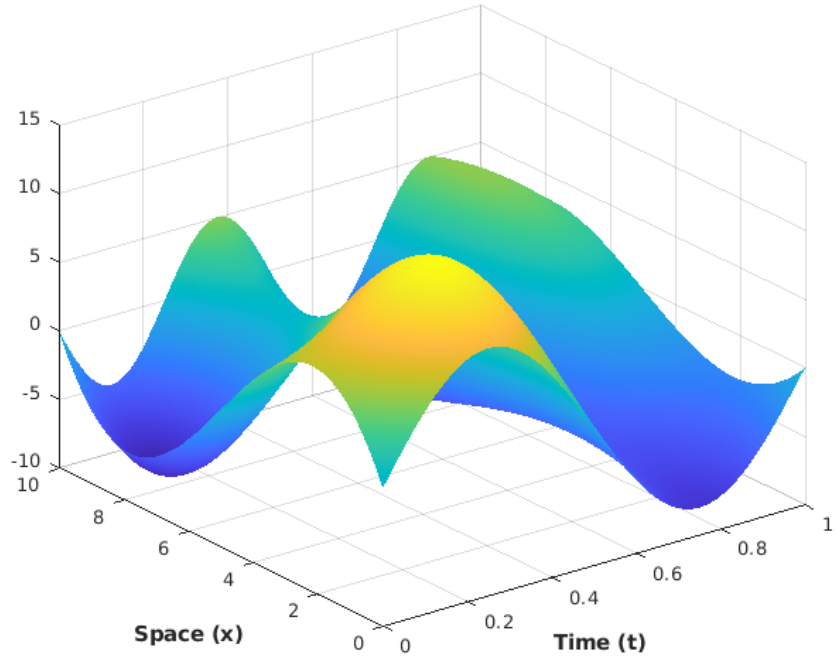


Figure 2.7: Variation de déplacement en fonction du temps et de l'espace pour $\underline{\underline{f}} = 3 \times cos(\frac{3\,(\pi/3)\,l_x}{L})^T \times cos(\frac{2\,(\pi/3)\,l_t}{T})$ (nombre de modes = 20).

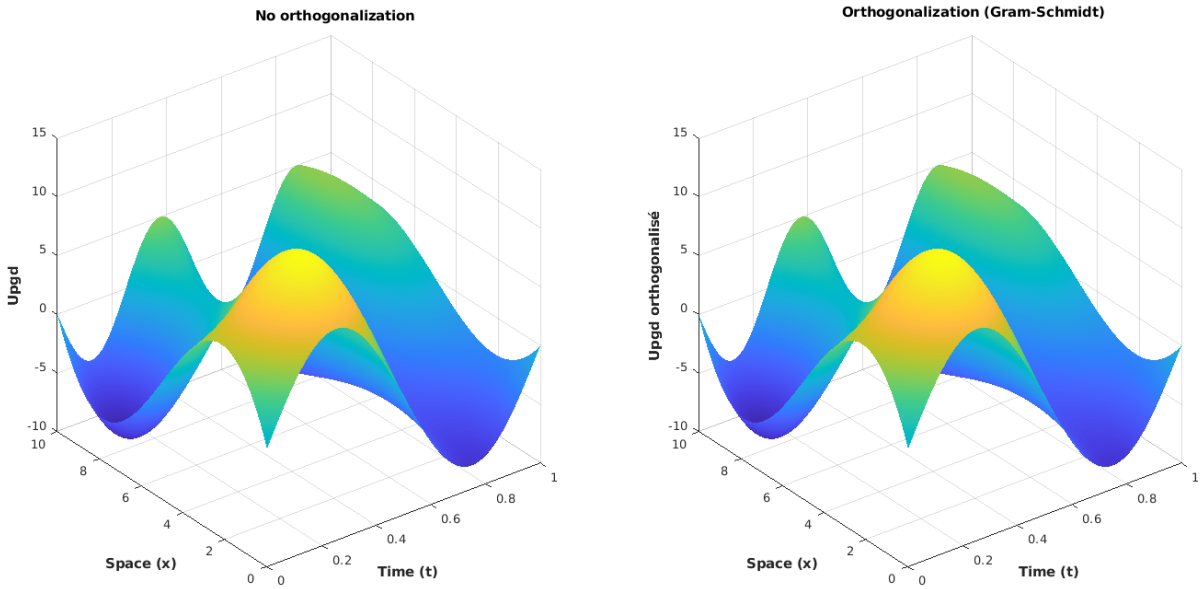


Figure 2.8: Variation de déplacement en fonction du temps et de l'espace pour $\underline{\underline{f}} = 3 \times cos(\frac{3\,(\pi/3)\,l_x}{L})^T \times cos(\frac{2\,(\pi/3)\,l_t}{T})$ (nombre de modes = 20).

2.5 Conclusion

Dans ce dernier TP, on a pu résoudre une équation différentielle pour trouver le déplacement dépendant du temps et de l'espace avec l'algorithme de force brute. Ensuite, la réduction de ce champ de déplacement est entamée en utilisant l'algorithme de *PGD* en imposant le nombre de modes. Pour pouvoir minimiser l'erreur d'approximation, une deuxième version améliorée de *PGD* consistant à l'ajout de l'orthogonalisation avec *Gram-Schmidt*.