Projet numérique StatNum du département GMM
École Nationale des Ponts et Chaussées

BIGMéca

Réduction de modèle par apprentissage automatique

David Ryckelynck, Thomas Daniel, 2019

Photo Rolls-Royce d'une aube de turbine

Résumé : Une structure est soumise à un champ de température aléatoire et à un chargement mécanique aléatoire. La propagation d'incertitude vers la prévision de durée de vie fait appel à un grand nombre de simulations par éléments finis, plus de 1000. Or les premières simulations génèrent des données que l'on peut exploiter pour accélérer les simulations suivantes à l'aide d'un modèle d'ordre réduit construit par apprentissage automatique. Voici quelques questions abordées ici. Quels traitement statistique peut-on faire des données produites par simulation ? Combien faut-il de données d'entrainement pour obtenir un modèle réduit pertinent ? Comment évaluer la précision de modèles réduits ou simplifiés ? Quels sont les gains produits par ce type d'approche ? Quelles en sont les limites ?

1 Etude d'un problème thermomécanique

On s'intéresse à une structure assimilable à une aube de turbine, dans le sens où cette structure est :

— percée de trous,

— soumise à un chargement de flexion,
— soumise à champ de température aléatoire.

On limite notre étude au cas d'un matériau élastoplastique dont le module de Young, le coefficient de dilatation thermique et la limite d'élasticité sont fonction de la température. Le matériau d'étude est du type HSS (High Strength Structural Steels). Les coefficients thermo-élastoplastiques sont dans le fichier *elas* et *evp*.

Le modèle éléments finis est décrit par les fichiers *mesh.geof* pour le maillage (Figure **??**), et *simu.inp* pour la description du problème mécanique.

Les équations aux dérivées partielles étudiées sont :

$$\mathbf{div}\,\boldsymbol{\sigma} = 0$$

$$\boldsymbol{\sigma} = \frac{E(T)}{1+\nu}\,\boldsymbol{\varepsilon}^e + \frac{\nu\,E(T)}{(1+\nu)\,(1-2\nu)}\,\mathrm{Tr}(\boldsymbol{\varepsilon}^e)\,\mathbf{I}$$

$$\frac{\nabla\mathbf{u} + \nabla\mathbf{u}^T}{2} = \boldsymbol{\varepsilon}^e + \boldsymbol{\varepsilon}^p + \alpha(T)\,(T - T_o)\,\mathbf{I}$$

$$\dot{\boldsymbol{\varepsilon}}^p = \dot{p}\frac{3}{2}\,\frac{\boldsymbol{\sigma}^D}{\sqrt{\frac{3}{2}\boldsymbol{\sigma}^D : \boldsymbol{\sigma}^D}}, \quad \dot{p}^2 = \frac{2}{3}\,\dot{\boldsymbol{\varepsilon}}^p : \dot{\boldsymbol{\varepsilon}}^p$$

$$\boldsymbol{\sigma}^D = \boldsymbol{\sigma} - \frac{1}{3}\,\mathrm{Tr}(\boldsymbol{\sigma})\,\mathbf{I}$$

$$f = \sqrt{\frac{3}{2}\boldsymbol{\sigma}^D : \boldsymbol{\sigma}^D} - R_\infty\,(1 - \exp(-b\,p)) - \sigma_y(T)$$

$$f \leq 0, \quad f\,\dot{p} = 0, \quad \dot{p} \geq 0$$

$$\mathbf{u}(\mathbf{x}) = \mathbf{u}_D(\mathbf{x}) \quad \forall \mathbf{x} \in \partial_D\Omega$$

$$\boldsymbol{\sigma}.\mathbf{n}(\mathbf{x}) = 0 \quad \forall \mathbf{x} \in \partial\Omega\backslash\partial_D\Omega$$

où E est le module de Young du matériau, ν est le coefficient de Poisson, α le coefficient de dilatation, σ_y est la limite d'élasticité, R_∞ et b sont des coefficients d'écrouissage isotrope. $\mathbf{u}_D$ est une condition limite de Dirichlet donnée.

2 Rappels théoriques

Le produit scalaire d'un vecteur $\mathbf{r} \in \mathbb{R}^{\mathcal{N}}$ sur un vecteur $\mathbf{v} \in \mathbb{R}^{\mathcal{N}}$ s'écrit : $\mathbf{v}^T\,\mathbf{r}$. Soit $\mathbf{V} \in \mathbb{R}^{\mathcal{N}\times N}$ une matrice orthogonale ($\mathbf{V}^T\,\mathbf{V} = \mathbf{I}$) qui contient N vecteurs d'un même espace vectoriel $\mathbb{R}^{\mathcal{N}}$. Alors les projections de $\mathbf{r}$ sur ces vecteurs sont obtenues par le produit : $\mathbf{V}^T\,\mathbf{r} \in \mathbb{R}^N$.

Par ailleurs, le lemme de Céa dit que l'erreur d'approximation de la solution d'une équation aux dérivées partielles elliptique est bornée par une constante de stabilité fois l'erreur de projection de cette solution dans l'espace d'approximation. Ce qui s'écrit :

$$\|\mathbf{u} - \mathbf{u}_a\|_{H^1(\Omega)} \leq c \inf_{\mathbf{v}_a \in \mathcal{V}_a} \|\mathbf{u} - \mathbf{v}_a\|_{H^1(\Omega)}$$

où $\mathbf{u} \in \mathcal{V}$ est la solution exacte de l'équation elliptique, $\mathbf{u}_a \in \mathcal{V}_a$ son approximation par projection de Galerkin de l'équation dans $\mathcal{V}_a$ et c est une constante de stabilité qui dépend du problème traité. Donc, meilleur est l'espace d'approximation, meilleure sera la prévision approchée. On remarque que l'évaluation de $\inf_{\mathbf{v}_a \in \mathcal{V}_a} \|\mathbf{u} - \mathbf{v}_a\|_{H^1(\Omega)}$ nécessite de connaitre $\mathbf{u}$. Dans le cadre d'une approche par apprentissage automatique, cela nécessite de disposer de données de test. Remarque : $\mathbf{u}_a$ qui est obtenue par des équations projetées sur $\mathcal{V}_a$, ne réalise pas le minimum de $\|\mathbf{u} - \mathbf{v}_a\|_{H^1(\Omega)}$, sauf cas particulier.

3 Modèle gaussien d'environnement aléatoire

3.1 Chargement mécanique aléatoire

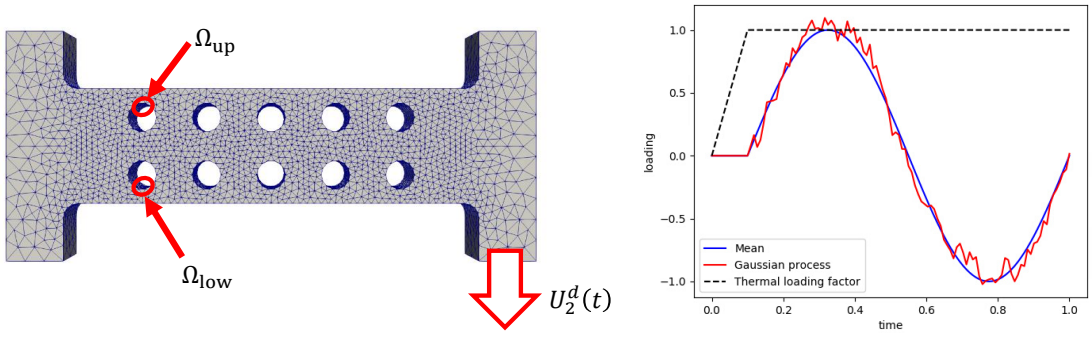


FIGURE 1 – Structure en flexion, zones d'intérêt Ω_{low} Ω_{up}, à droite chargement thermomécanique (T - -, U_2^d -)

On considère pour commencer un chargement mécanique $U_2^d(t)$ aléatoire représenté par un modèle gaussien (Figure 1) :

$$U_2^d(t) = a(t) + X(t), \quad a(t) = \begin{cases} 0 & t < 0,1 \\ u_o \sin(\frac{2\pi(t-0,1)}{0,9}) \end{cases}$$

où $a(t)$ est le chargement nominal et $X(t)$ est un processus gaussien à moyenne nulle et à fonction de covariance $K_X(t,s)$ entre deux instants. $K_X(t,s)$ est une fonction de covariance gamma-exponentielle.

Récupération des données : copier *partages/BigMeca/Cours_1* dans *perso*
Aller : dans *Cours_1/TP1*
Réaliser un calcul nominal sans perturbation aléatoire et sauvegarder la valeur du dommage sur les zones Ω_{low} et Ω_{up}.
Zrun simu_mean
Zrun -pp simu_mean
Visualiser *la prévision des dommages avec Zmaster gappy.ut. Est-ce normal qu'il y ait un dommage important à gauche de la structure ?*

La fonction de covariance gamma-exponentielle (gaussienne pour $\gamma = 2$) a pour expression :

$$K_X(t,s) = k_o \, \exp\left(-(\frac{|t-s|}{t_c})^\gamma\right), \quad 0 < \gamma \leq 2, \; k_o > 0, \; t_c > 0$$

En appliquant le théorème de Karhunen-Loève, on calcule N modes $e_k(t)$ pour représenter $X(t)$ avec N composantes $(Z_k)_{k=1}^N$:

$$X(t) = \sum_{k=1}^{N} Z_k \, e_k(t), \quad \int_{s=0}^{t_f} e_i(s) \, e_j(s) \, ds = \delta_{ij}$$

$$\int_{s=0}^{t_f} K_X(t,s) \, e_k(s) \, ds = \lambda_k \, e_k(t)$$

où les Z_k sont des variables gaussiennes.
En introduisant un découpage de $[0, t_f]$ en n pas de temps, on obtient la matrice de covariance $\mathbf{\Gamma}$ de taille $n \times n$:

$$\Gamma_{ij} = K_X(t_i, t_j)$$

Les valeurs propres de $\mathbf{\Gamma}$ sont rangées par ordre décroissant. Le $k$ième vecteur propre de $\mathbf{\Gamma}$ nous donne une approximation du vecteur $e_k(t_i)$ pour $t_i \in \{t_0, \ldots, t_n\}$, si le vecteur propre est de norme unitaire. On préfère prendre la norme du vecteur propre k égale à $\sqrt{\lambda_k}$. Soit $\mathbf{A}$ la matrice des N premiers vecteurs propres de $\mathbf{\Gamma}$ avec :

$$\mathbf{A}^T \mathbf{A} = \mathrm{diag}([\lambda_1, \ldots \lambda_N]), \quad \mathbf{A} \, \mathbf{A}^T = \mathbf{\Gamma}$$

En prenant un vecteur gaussien centré $\mathbf{y} \in \mathbb{R}^N$ de matrice de covariance égale à l'identité, on construit le vecteur gaussien $\mathbf{X}$ tel que :

$$\mathbf{X} = \mathbf{A} \, \mathbf{y}, \quad X(t_i) = X_i$$

Programmer le calcul de $K_X(t1, t2)$ dans *01-gaussian_process.py*, *myCovFct* (choisir γ, t_c, k_o, avec $param = [k_o, t_c, \gamma]$).
Copier la fonction dans script simple pour tracer la fonction $f(t) = K_X(t, s)$ pour $s = 0.4$ (utiliser matplotlib.pyplot as plt, et s'inspirer de PlotCurves dans *01-gaussian_process.py*)
Programmer le calcul de Γ_{ij} dans *01-gaussian_process.py*, dans *GetCovMatrix* (*self* : permet de pointer sur l'objet GaussianProcess).
Calculer les vecteurs propres de $\mathbf{\Gamma}$, avec *GetModes*. Visualiser un mode interprété comme une fonction du temps, c'est à dire une colonne de $\mathbf{A}$ (avec 100 pas de temps de 0. à 1.).
Engendrer 20 vecteurs $\mathbf{y} \in \mathbb{R}^5$ (voir les commentaires dans *GetRealizations*).
Lancer python 01-gaussian_process.py, pour générer 20 cas de chargement.
Lancer les simulations mécaniques, avec *python 02-run_simus.py*.
Récupérer les dommages des 20 cas de chargement, avec *03-postprocessing.py*.
Commenter l'estimation de la densité de probabilité pour le dommage pour chaque zone Ω_{low} et Ω_{up}.
Estimer la moyenne et l'écart type pour les deux dommages (utiliser l'aide de numpy sur les vecteurs damageLowerZone, damageUpperZone et ratio).
Analyser le résultat de la propagation d'incertitude. La moyenne correspond-t-elle au dommage du cas nominal ? Que dire de la forme des densités de probabilité ?

3.2 Chargement thermique aléatoire

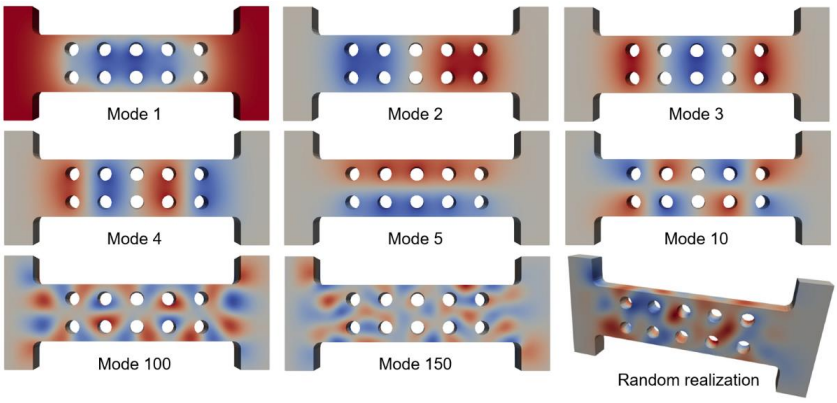
On peut également générer un champ gaussien pour modéliser la variation spatiale des champs de température à la surface de Ω, notée $\partial\Omega$. Dans ce cas, la fonction de covariance est :

$$K_T(\mathbf{x}, \mathbf{z}) = k'_o \exp\left(-(\frac{d_G(\mathbf{x}, \mathbf{z})}{\ell_c})^{\gamma'}\right), \quad 0 < \gamma' \leq 2,\ k'_o > 0,\ \ell_c > 0$$

où $d_G(\mathbf{x}, \mathbf{z})$ est la distance géodésique entre deux points de la surface $\partial\Omega$.

En pratique, nous évaluons les distances géodésiques sur les nœuds du maillage surfacique de Ω. Ainsi la matrice de covariance a une taille égale au nombre de nœuds de ce maillage surfacique et Γ_{ij} est le terme de covariance calculé entre le nœud i et le nœud j. Les modes propres de $\mathbf{\Gamma}$, c'est à dire les colonnes de $\mathbf{A}$, sont des champs surfaciques de température représentés sur la figure ci-dessous.

Ensuite chaque mode surfacique est étendu dans le volume Ω via les équations de la chaleur en régime stationnaire et le modèle éléments finis 3D. Les modes volumiques



Modes de température obtenus par champ gaussien.

sont appelés $(\delta T_i(\mathbf{x}))_{i=1,\dots N}$. La forme du champ de température stochastique est alors la suivante pour $\mathbf{x} \in \Omega$ et pour des variables aléatoires $(\alpha_i)_{i=1}^N$ gaussiennes centrées réduites $\mathcal{N}(0,1)$:

$$T(\mathbf{x}, \boldsymbol{\alpha}) = \tau(\, T_{ref}(\mathbf{x}) + \sum_{i=1}^{N} \alpha_i \, \delta T_i(\mathbf{x}) \,)$$

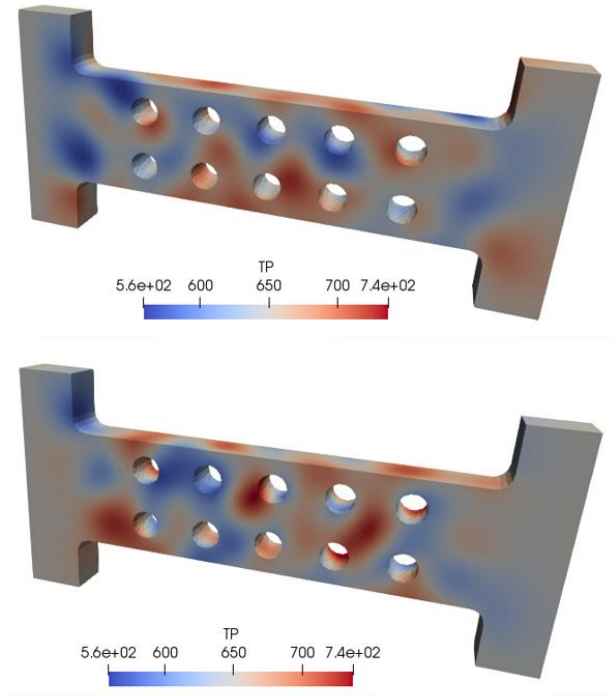
où la fonction τ réalise le minimum entre sa variable d'entrée et la température de fusion du matériau (On ne peut pas dépasser la température de fusion).

On vérifie en pratique, que la fonction τ n'affecte pas la propriété suivante : en tout point $\mathbf{x}$ de $\partial\Omega$ on a :

$$\mathbb{E}(T(\mathbf{x},)) = T_{ref}(\mathbf{x}), \quad \forall \mathbf{x} \in \partial\Omega$$

4 Régression linéaire pour la prévision de dommages par métamodèle

Métamodèle : Un métamodèle est un modèle qui établit une relation mathématique entre des entrées (appelées aussi variables explicatives) et des sorties (variables expliquées) en se substituant à un modèle plus complexe à évaluer. Les modèles de simulation basés sur la physique intègrent des concepts généraux, ou des données génériques, qui dépassent le cadre de l'application particulière qui lie les entrées et les sorties mentionnées ci-dessus. Un métamodèle est construit pour prévoir de façon rapide et éventuellement approximative cette relation particulière entre entrées et sorties, dans un certain domaine de variation des données d'entrée. Il est construit à partir de données d'entrées rangées dans une matrice $\mathbf{X}$ (observations $\times$ variables explicatives) et de données de sortie $\mathbf{Y}$ (observations $\times$ variables expliquées) associées aux variables explicatives par le modèle



Exemple de 2 réalisations aléatoires du champ de température volumique

d'origine que l'on souhaite remplacer par le métamodèle. L'objectif est de réaliser de nouvelles prévisions $\widehat{y}(x^\star)$ pour des variables d'entrée $x^\star$ qui ne sont pas contenues dans $\mathbf{X}$ en exploitant le métamodèle pour le calcul des données de sortie $\widehat{y}$. Si le métamodèle est à la fois précis et rapide (notion à préciser par l'utilisateur de celui-ci à l'aide des outils statistiques disponibles), il peut être exploité pour faire de la propagation d'incertitudes, afin obtenir une prévision d'une densité de probabilité sur $\widehat{y}$.

Dans l'exemple qui suit nous disposons de 500 résultats de simulations de dommage pour différents champs de températures. Tous les autres paramètres de simulation des dommages sont supposés déterministes et conservent une valeur constante dans cette étude.

Les champs de température sont exprimés à la l'aide des modes présentés dans la section précédente :

$$T(\mathbf{x}, \boldsymbol{\alpha}) = \tau(\, T_{ref}(\mathbf{x}) + \sum_{i=1}^{N} \alpha_i \, \delta T_i(\mathbf{x}) \,)$$

où $\boldsymbol{\alpha}$ est le vecteur des variables explicatives du métamodèle auxquelles on souhaite associer une prévision du dommage $D(\mathbf{x}_o, t_f)$ dans la région Ω_{up}.

4.1 Choix d'un ensemble d'entrainement et d'un ensemble de test

A l'aide d'une permutation aléatoire de la liste d'indice de 0 à 499, construire une liste de données d'apprentissage à partir des données disponibles. Prendre 80% des données pour l'apprentissage.

Récupérer les données dans *partages/BigMeca/Cours_2/* Utiliser la commande *numpy.random.permutation* après avoir consulté l'aide de numpy.

4.2 Modèle stochastique des températures à deux variables explicatives

La prévision du dommage par une régression linéaire a la forme suivante

$$D(\mathbf{x}_o, t_f) = \beta_o + \sum_{i=1}^{N} \beta_i \alpha_i + \varepsilon$$

avec $p = N+1$ coefficients de régression linéaire à identifier en exploitant n prévisions du dommage par le modèle éléments finis correspondant à n valeurs aléatoires des variables explicatives notées $\boldsymbol{\alpha}^{(j)} \in \mathbb{R}^N$, $j = 1, \dots n$. Cette équation prend la forme matricielle suivante pour l'ensemble des observations :

$$\mathbf{y} = \mathbf{X}\,\boldsymbol{\beta} + \varepsilon$$

avec $y = D(\mathbf{x}_o, t_f)$ et :

$$\mathbf{X} = \begin{bmatrix} 1 & \alpha_1^{(1)} & \dots & \alpha_N^{(1)} \\ \dots & \alpha_1^{(j)} & \dots & \alpha_N^{(j)} \\ 1 & \alpha_1^{(n)} & \dots & \alpha_N^{(n)} \end{bmatrix}$$

où $\mathbf{y}$ est le vecteur des n valeurs calculées par le modèle éléments finis : $y_j = D^{(j)}$. La valeur optimale des coefficients de la régression linéaire s'obtient par moindres carrés :

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

si le rang de $\mathbf{X}$ est bien égal à p.

On en déduit le métamodèle suivant :

$$\widehat{y} = \widehat{\boldsymbol{\beta}}^T \begin{bmatrix} 1 \\ \boldsymbol{\alpha} \end{bmatrix} \quad \forall \boldsymbol{\alpha}$$

Dans la suite on suppose que ε suit une distribution gaussienne.

Dans cette section on considère le cas simple $p = 3$, c'est à dire $N = 2$ modes de température pour le modèle stochastique.

[Question facultative] : Programmer le calcul des indices de conditionnement de la matrice de corrélations (cf. page 176 du polycopié du cours STNUM). Va-t-on pouvoir réaliser une régression linéaire avec les données sélectionnées précédemment ?

Dans *02-tp2.py* : fixer *dim=2* (nombre de modes de température) et exécuter le script pour effectuer une régression linéaire par la méthode des moindres carrés ordinaire.
Commenter les p-valeurs des tests de significativité.
Commenter le coefficient de détermination R^2, ainsi que les signes des coefficients β_i obtenus.
Quel est l'indicateur d'erreur le plus pertinent (RMSE ou MAPE, cf. page 180 du polycopié du cours STNUM) ?
Relancer le script dans le cas $n = p$ en fixant le paramètre de la fonction *train_test_split* de scikit-learn à 0.994, puis dans le cas $n = 10\,p$ en le fixant à 0.94. Commenter.

4.3 Cas industriel

Il faut 150 modes pour tenir compte de la variabilité des températures pour l'application industrielle visée.

Dans *02-tp2.py* : fixer *dim=150*. Effectuer la régression linéaire en ne prenant en compte que les 2 premières variables explicatives, puis en prenant en compte les 150 variables explicatives. Les coefficients β_0, β_1 et β_2 prédits sont-ils stables ?

En présence de 150 variables explicatives, l'ensemble d'entraînement n'est pas assez grand pour obtenir des prédictions robustes. Les résultats dépendent fortement de l'échantillonnage des données. Pour obtenir un indicateur d'erreur robuste (moins sensible à l'échantillonnage) pour le choix du modèle linéaire, il est possible d'évaluer le modèle par validation croisée.

Programmer la k-fold cross validation dans *01-tp2.py*.

4.4 Méthode de régression pénalisée

Appliquer une méthode de régression pénalisée (au choix : Ridge Regression, LASSO ou ElasticNet, cf. documentation de scikit-learn sur internet). Utiliser la validation croisée codée précédemment pour optimiser le choix du ou des hyperparamètres de la méthode de régression choisie. Les résultats sont-ils meilleurs qu'avec la régression linéaire par moindres carrés ?

4.5 Régression linéaire avec fonctions quadratiques

Peut-on améliorer les prédictions en combinant des fonctions de base pour obtenir un modèle quadratique en α ?

5 Clustering de données

Intuitivement, un petit groupe de données devrait admettre un modèle plus simple qu'un grand groupe. Cependant, moins on a de données plus il est difficile d'apprendre un modèle générique à partir de ces données. Reste à bien définir le clustering des données et la pertinence de celui-ci.

5.1 Visualisation des données en dimension réduite

La visualisation de données quantitatives en 2D ou en 3D est l'une des motivations pour mettre en œuvre une méthode de réduction de dimension.
On limite notre attention à des données centrées notée $\mathbf{X}^c \in \mathbb{R}^{n\times p}$:

$$\mathbf{X^c} = \mathbf{X} - \frac{1}{n}\mathbf{1}_n\,\mathbf{1}_n^T\,\mathbf{X}$$

où $\mathbf{1}_n \in \mathbb{R}^n$ est un vecteur composé de 1 rangés en colonne et n est le nombre d'individus. L'analyse en composantes principales (ACP) non-normée permet d'extraire de données centrées $\mathbf{X}^c$ des facteurs principaux $\mathbf{v}_\alpha \in \mathbb{R}^p$, $\alpha = 1, \dots p$. En retenant les deux premiers facteurs principaux ($\alpha = 1, 2$), c'est à dire les deux facteurs les plus importants, il est possible de visualiser en 2D les deux composantes principales $\mathbf{C} \in \mathbb{R}^{n\times 2}$:

$$\mathbf{C} = \mathbf{X^c}\,\mathbf{V}, \quad \mathbf{V} = [\mathbf{v}_1,\ \mathbf{v}_2]$$

.

Récupérer les données dans *partages/BigMeca/Cours_3/*
Calculer les deux premières composantes principales $\mathbf{C} \in \mathbb{R}^{n\times 2}$ avec *decomposition.PCA* de *scikit-learn* (fichier *01-PCA.py*).
Visualiser ces deux premières composantes principales avec :

```
import matplotlib.pyplot as plt
F=plt.figure(figsize=(15, 8))
plt.title("Deux premières composantes principales")
plt.xlabel("C1")
plt.ylabel("C2")
plt.plot(C[ :,0],C[ :,1],'.')
plt.show()
```

Les données ont elles une structure particulière ?

5.2 Clustering des températures en dimension 2

L'ACP fait partie de la famille des méthodes descriptives des données, en apprentissage non supervisé (on n'indique pas à un algorithme ce que l'on souhaite découvrir dans les données). Les méthodes de clustering font également parties de cette famille de méthodes.

Appliquer la méthode *k-means* aux 2 composantes principales $\mathbf{C}$ (utiliser *cluster.KMeans* de *scikit-learn*, fichier *02-kMeans.py*). Que doit-on choisir pour cela ?
Proposer une façon de visualiser les clusters.
Vu la forme des deux premières composantes principales, est-il justifié de réaliser un clustering spectral ?

5.3 Clustering des températures en dimension 10

Plus la dimension de l'espace qui contient chaque individu est grande (il s'agit de p), moins la notion de distance est pertinente, dans le sens où tous les points deviennent équidistants.

Appliquer la méthode *k-means* aux 10 composantes principales pour construire 5 clusters (modifier *01-PCA.py* puis utiliser *02-kMeans.py*)
Peut-on visualiser les clusters (fichier *02-kMeans.py*) ?
Comparer le temps de calcul des clusters en dimension 10 à celui nécessaire en dimension 2.
Répéter 10 fois la construction de clusters en dimension 10 et celle en dimension 2. Commenter la stabilité des centroïds.
Est-il raisonnable selon vous d'appliquer le clustering directement à des champs de température calculés avec un maillage éléments finis constitué de plus de 10^5 nœuds ($p > 10^5$) ?

5.4 Métrique pour le clustering des températures

La métrique utilisée pour définir des similitudes entre données doit être adaptée en fonction de l'objectif du traitement de données.

Ici, on souhaite faire le lien entre un champ de température et la base réduite mécanique qui permet de prévoir les déplacements induits par ce champ de température. Il s'agit d'une métrique basée sur l'étude de l'effet des données étudiées (l'effet des températures sur les déplacements via un modèle thermo-mécanique).

Pour constituer l'ensemble des données étudiées, nous avons réalisé n simulations par éléments finis en faisant varier aléatoirement le champ de température. Chaque simulation a produit des prévisions de déplacement sur tous les nœuds d'un maillage et pour n_t instants dans un intervalle $[0,\ t_f]$. Pour chaque champ de température $T^{(i)}$, $i = 1, \dots n$, les déplacements simulés ont été rangés dans une matrice $\mathbf{X}^{u,(i)} \in \mathbb{R}^{n_t \times \mathcal{N}}$. Puis l'ACP non normée et non-centrée a été appliquée à $\mathbf{X}^{u,(i)}$ afin d'obtenir une base réduite $\mathbf{V}^{(i)} \in \mathbb{R}^{\mathcal{N} \times N^{(i)}}$ qui contient les $N^{(i)}$ premiers facteurs principaux tels que :

$$\|\mathbf{X}^{u,(i)} - \mathbf{X}^{u,(i)}\, \mathbf{V}^{(i)}\, \mathbf{V}^{(i)\,T}\| < \epsilon_{tol}$$

où ϵ_{tol} est une erreur de troncature choisie. Ainsi, à chaque simulation on associe un sous-espace vectoriel engendré par les colonnes de $\mathbf{V}^{(i)}$, où $\mathbf{V}^{(i)}$ est une matrice orthogonale.

Distance de Grassmann La distance de Grassmann, notée d, entre deux sous-espaces vectoriels engendrés par les bases réduites orthonormales $\mathbf{A}$ et $\mathbf{B}$, aux dimensions identiques, est donnée par la décomposition aux valeurs singulières suivante :

$$\mathbf{A}^T\mathbf{B} = \mathbf{U}\, cos(\boldsymbol{\theta})\mathbf{P}^T, \quad d(\text{span}(\mathbf{A}), \text{span}(\mathbf{B})) = \|\boldsymbol{\theta}\|$$

En procédant ainsi, deux champs de températures seront similaires si ils admettent des bases réduites similaires pour représenter les déplacements associés. La matrice de dissimilitude est constituée des "distances" entre les bases (i) et (j) :

$$D_{ij} = d(\text{span}(\mathbf{V}^{(i)}), \text{span}(\mathbf{V}^{(j)})), \quad i = 1, \dots n,\ j = 1, \dots n$$

Récupérer les données dans *partages/BigMeca/Cours_3/*
Calculer la dissimilitude entre deux bases réduites $\mathbf{V}^{(1)}$ et $\mathbf{V}^{(2)}$, à l'aide de la formule ci-dessus (*03-Grassmann_distance.py*).
Récupérer la matrice de dissimilitude entre bases réduites. Combien de simulations a-t-on réalisé ? (fichier *05-kMedoids.py*)
Essayer d'appliquer la méthode k-means pour obtenir 5 clusters de données.
Appliquer la méthode k-médoïds pour obtenir 5 clusters de données à partir de la matrice de dissimilitude entre bases réduites (fichier *05-kMedoids.py*).
Appliquer une méthode de Multi Dimensional Scaling (voir Scikit-learn) pour visualiser des données en 2D en connaissant une matrice de dissimilitude des bases réduites (fichier *04-visu_MDS.py*).
Comparer les clusters obtenus par k-médoïs sur la matrice de dissimilitude des bases réduites et le clustering k-médoïds effectué sur les champs de température.

6 Réduction d'ordre de modèle

6.1 Projection des équations éléments finis sur une base réduite

Soit $\mathcal{U}_h = \text{span}(\phi_i)_{i=1}^{\mathcal{N}}$ l'espace d'approximation utilisé pour générer un modèle par éléments finis à l'aide des fonction de forme $(\phi_i)_{i=1}^{\mathcal{N}}$. On considère un sous espace de $\mathcal{U}_h$ engendré par les vecteurs d'une famille libre de vecteurs $(\psi_k)_{k=1}^{N}$ de coordonnées V_{ik} tel que :

$$\psi_k(\mathbf{x}) = \sum_{i=1}^{\mathcal{N}} \phi_i(\mathbf{x})\, V_{ik} \quad \mathbf{x} \in \Omega$$

où Ω est le domaine couvert par le maillage par éléments finis. On note $\mathcal{U}_r$ le sous espace de $\mathcal{U}_h$ engendré par les vecteurs $(\psi_k)_{k=1}^{N}$.

Soit $\mathbf{u}_r$ un élément de $\mathcal{U}_r$. Comme $\mathcal{U}_r \subset \mathcal{U}_h$, il existe un vecteur $\mathbf{q}_r \in \mathbb{R}^{\mathcal{N}}$ et un vecteur $\boldsymbol{\gamma} \in \mathbb{R}^{N}$ tels que :

$$\mathbf{u}_r(\mathbf{x}) = \sum_{i=1}^{\mathcal{N}} \phi_i(\mathbf{x})\, q_{ri} \quad \mathbf{x} \in \Omega, \quad \mathbf{q}_r = \mathbf{V}\,\boldsymbol{\gamma}$$

Ainsi :

$$\mathbf{u}_r(\mathbf{x}) = \sum_{k=1}^{N} \psi_k(\mathbf{x})\, \gamma_k \quad \mathbf{x} \in \Omega$$

Trouver la forme de l'énergie potentielle dont la minimisation (ou la stationnarité) permet d'obtenir le système linéaire des équations éléments finis suivant :

$$\mathbf{K}\,\mathbf{q} = \mathbf{F}$$

où $\mathbf{q} \in \mathbb{R}^{\mathcal{N}}$ est le vecteur des inconnues nodales permettant d'interpoler le champ de déplacement $\mathbf{u} \in \mathcal{U}_h$ avec les fonctions de forme éléments finis. $\mathbf{K}$ est une matrice symétrique définie positive. L'énergie potentielle sera notée $J(\mathbf{q})$.

Quelle équation obtient-on en minimisant l'énergie potentielle dans $\mathcal{U}_r$? Pour répondre à cette question, étudiez la minimisation de $J_r(\gamma) = J(\mathbf{V}\,\gamma)$.

Le système d'équation réduit est celui d'un modèle d'ordre réduit qui peut être utilisé pour simuler l'effet de champs de température de façon plus rapide qu'avec le modèle éléments finis. A-t-on moins d'équations à résoudre ?

6.2 Construction d'une base réduite par apprentissage automatique

En exploitant le programme en Python 1parameter_training.py, vous pouvez générer différentes simulations élastiques pour des températures interpolées entre deux champs de températures donnés, Temp0.node et Temp2.node.

A lecture du fichier 1parameter_training.py, combien de simulation sont-elles réalisées ?

Exécuter les commandes suivantes :

— use_zset_rom

— python 1parameter_training.py

En attendant la fin du calcul, étudier le contenu de 1parameter_training.py et commentez le (dans le fichier lui-même).

Visualiser quelques résultats, en ouvrant un autre terminal puis en utilsant la commande Zmaster FE_param_ini ou Zmaster FE_param.

6.3 Apprentissage automatique pour un modèle à un paramètre

Le logiciel Zset permet de construire une base réduite avec la commande ***incremental_pod_new. Cette base réduite est stockée dans le fichier dof_param.basis. Le fichier dof_param.snaps contient les coordonnées réduites des déplacement pour tous les pas de temps calculés au cours de la simulation par éléments finis. Visualisez la base réduite dof_param avec les commandes Zrun mesher_visu_modes, puis Zmaster dof_param.ut. Utiliser le bouton FEA pour voir les composantes U1, U2, U3 de chaque mode de la base réduite. Utiliser Magnification 1000. pour amplifier la déformée produite par les modes.

On propose de refaire la construction d'une base réduite à partir des champs éléments finis calculés. Les données de simulation sont dans les fichiers FE_0.node FE_1.node FE_2.node. Dans le programme python exercise_working_on_svd.py les fichiers FE_i.node sont récupéré et leur contenu est rangé dans une matrice $\mathbf{X} \in \mathbb{R}^{m \times \mathcal{N}}$. La base réduite calculée par Zset est rangée dans la matrice $\mathbf{V}zset \in \mathbb{R}^{\mathcal{N} \times N}$.

Y a-t-il vraiment réduction du modèle avec Zset ?

Recalculer une base $\mathbf{V}$ en applicant la décomposition aux valeurs singulières (tronquée) à $\mathbf{X}$, pour avoir :

$$\mathbf{X} = \mathbf{W}\,\mathbf{\Sigma}\,\mathbf{V}^T$$

on pourra utiliser numpy.linalg.svd ou la svd de scikit-learn. Le choix de la sélection des valeurs singulières non nulles ou significatives devra se faire en exploitant des données de validation qui sont différentes des données d'entrainement. Les données d'entrainement sont celles contenues dans $\mathbf{X}$.

Définir puis calculer l'erreur de projection de $\mathbf{X}$ sur la base réduite $\mathbf{V}_{zset}$.

Calculer la distance de Grassmann entre $\mathbf{V}$ et $\mathbf{V}_{zset}$, $d(\text{span}(\mathbf{V}), \text{span}(\mathbf{V}_{zset}))$.

Calculer l'erreur de projection des données de validation.

Recommencer l'étude à un paramètre en réalisant un plan d'expérience aléatoire pour générer les données de simulation. Vous pourrez utiliser les commandes suivantes : import numpy as np
Liste = np.random.randint(0,m,Ntrain,dtype='I')
pour générer un liste d'indices dans $\{0, \ldots, m\}$ contenant Ntrain valeurs.

6.4 Exploitation du modèle d'ordre réduit

Construire un modèle d'ordre réduit avec la commande Zrun mesher_RID. Vous constaterez que le maillage rid.geof est réduit. Il s'agit d'une méthode d'hyper-réduction. Le rang de $\mathbf{K}\,\mathbf{V}$ étant au maximum N, il est possible de sélectionner quelques lignes (au moins N) de $\mathbf{K}\,\mathbf{V}$ pour résoudre de façon approchée les équations du modèle d'ordre réduit. Les équations hyper-réduites s'écrivent :

$$\mathbf{V}[\mathcal{F},:]^T\,\mathbf{K}[\mathcal{F},:]\mathbf{V}\,\boldsymbol{\gamma} = \mathbf{V}[\mathcal{F},:]^T\,\mathbf{F}[\mathcal{F}]$$

avec

$$\mathcal{F} = \{i \in \{1,\ldots,\mathcal{N}\},\ \int_{\Omega\setminus\Omega_r} \phi_i^2\, d\Omega = 0\}$$

où Ω_r est le domaine d'intégration réduit couvert par le maillage réduit.

Utiliser ce modèle d'ordre réduit avec la commande Zrun HROM.

Quel est le gain en temps de calcul ?

Comparer les résultats obtenus avec ceux de la méthode des éléments finis.

7 Apprentissage supervisé d'un classifieur pour la recommandation de labels

On dispose de champs de température dans une matrice $\mathbf{X} \in \mathbb{R}^{m\times\mathcal{N}}$, pour $m = 1000$ réalisations et $\mathcal{N} = 2500$ noeuds d'un modèle de type éléments finis. On dispose d'un label

pour chaque champ de température dans un vecteur $\mathbf{Y} \in \mathbb{R}^m$, avec $Y_i \in \{0, 1, 2, 3, 4, 5\}$. Ces labels proviennes d'un clustering par k-médoïdes sur une variété de Grassmann. C'est à dire que les points d'un même cluster sont des champs de température qui on un effet mécanique représenté par des bases réduites proches selon la distance de Grassmann. Ces points partagent donc une base réduite de champs de dépacement qui est à la fois précise et de petite dimension.

On souhaite réaliser l'apprentissage supervisé d'un classifieur, C, tel que $y = C(x \in \mathbb{R}^{\mathcal{N}}) \in \{0, 1, 2, 3, 4, 5\}$ à partir d'un ensemble d'apprentissage $(\mathbf{X}_{train}, \mathbf{Y}_{train})$ et d'un ensemble de validation $(\mathbf{X}_{val}, \mathbf{Y}_{val})$. Puis on souhaite réaliser un test final de précision du classifieur sur les données de test $(\mathbf{X}_{test}, \mathbf{Y}_{test})$.

Récupérer les données dans *partages/BigMeca/Cours_5_6/*
Réaliser une partition des données en données de test, données d'apprentissage et données de validation, en appliquant deux fois la commande *train_test_split* de scikit-learn. Prendre 20% de données pour le test, puis 20% du reste pour la validation.
Choisir une méthode de classification parmi k-Nearest Neighbors, Logistic Regression, GaussienNB, en lisant la documentation de scikit-learn.
Utiliser les données de validation pour choisir un pré-traitement de donnée avant apprentissage (voir *StandardScaler* de scikit-learn)
Evaluer la précision du classifieur avec la fonction *score* de scikit-learn, sur les données de validation pour choir les paramètres du classifieur, puis sur les données de test quand le classifieur est figé (en fin de séance uniquement).
Quel est le meilleur score obtenu ?

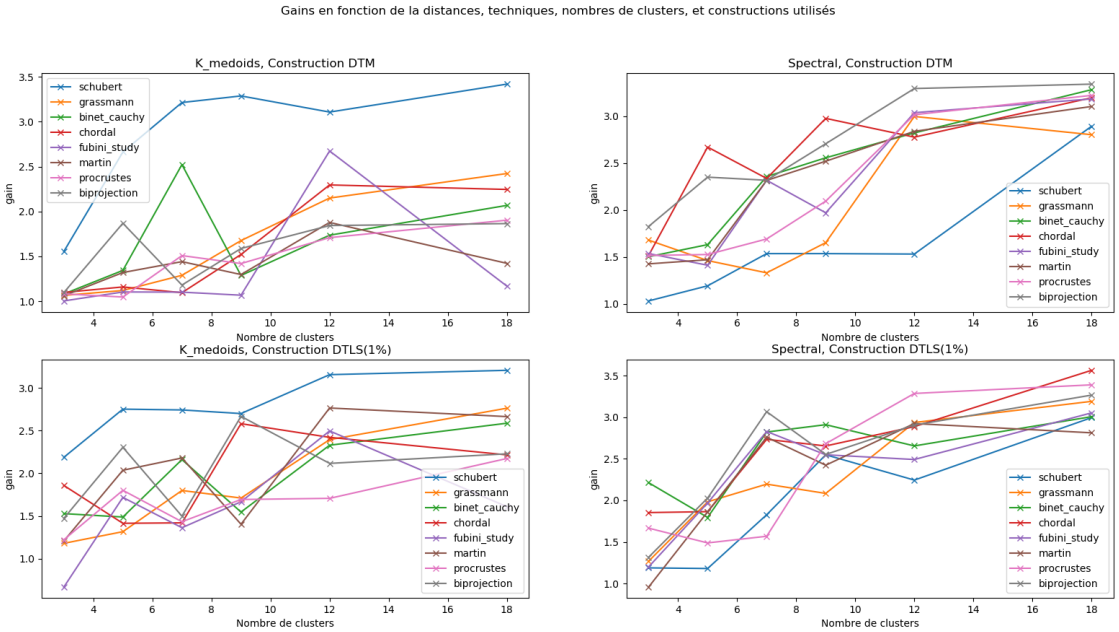
8 Conclusion

L'ensemble des tâches réalisées dans ce projet permet de développer un réseau de neurones artificiel appelé ROM-net [1]. Il prend en entré un champ de température sous la forme d'une image et fourni en sortie une prévision de dommage réalisée avec un modèle hyper-réduit. Ce modèle résulte de la sélection d'une base réduite au sein d'un cluster de données de simulation. Au cours de l'étape d'apprentissage, les clusters ont été construits en exploitant une distance de Grassmann.

Sur la figure ci-dessous, on montre le gain en précision sur les bases réduites en faisant varier le nombre de cluster, la distance choisie pour établir les clusters et la méthode de

1. Model order reduction assisted by deep neural networks (ROM-net), Thomas DANIEL, Fabien CASENAVE, Nissrine AKKARI and David RYCKELYNCK, soumis à Advanced Modeling and Simulation in Engineering Sciences, 2020

clustering choisie. La référence pour le calcul du gain est la précision obtenue par une seule base globale couvrant toutes les données d'entrainement. Pour avoir une comparaison rigoureuse des gains en précision nous avons fixé à 2 la taille de toutes les bases étudiées.



Gains en précision par l'approche de clustering des données de simulation pour établir un dictionnaire de bases réduite. Ces résultats on été obtenus par Arthur Pignet dans le cadre d'un S3 Recherche de Mines ParisTech en 2019-2020.

Pour contourner le calcul coûteux de distance de Grassmann, en phase d'exploitation du ROM-net (ou de test), un réseau de neurones convolutifs est utilisé comme classifieur dont les données d'entrée sont les champs de température.

Il y a bien un gain en précision par l'approche proposée.

9 Annexe : liste de commandes utiles

Modules to import :

– numpy, scipy.linalg, scipy.stats, matplotlib.pyplot, os

2D array A :

– A[i,j] = element i,j ; A.shape[0] = number of rows

– A.shape[1] = number of columns

Matrix of zeros :

– np.zeros((size, size))

Identity matrix :
– np.eye(size)
Tranpose of matrix A :
– A.transpose(), or np.transpose(A)
Matrix product :
– np.dot(A,B)
Element-wise division (arrays X and Y) :
– X/Y
Matrix whose columns equal to a 1D array X :
– np.transpose(np.tile(X, (nColumns, 1)))
Row (resp. column) i of a matrix A :
– A[i, :] (resp. A[:,i])
1D array from list L :
– np.array(L)
Eigenvectors of a symmetric matrix M :
– eigValues, eigVects = scipy.linalg.eigh(M)
Useful functions (element-wise if a is an array) :
– np.sin(a), np.exp(a), np.power(a,b), np.sqrt(a)
Max/min/mean/standard deviation of a 1D array X :
– X.max(), X.min(), X.mean(), X.std()
Generate time sequence :
– np.arange(0., tFinal + dt, dt) or np.linspace(0., tFinal, int(tFinal/dt)+1)
For the construction of the covariance function :
– use np.meshgrid, Gaussian kernel density estimation : see scipy.stats.gaussian_kde
Terminal instructions :
– os.system(« Linux command to be executed »)
« For » loop :
– for i in range(nIterations) : # first iteration i = 0, last iteration i = nIterations-1