Cours 10: Réduction de dimension pour les EDP

Emmanuel Franck[*],

19 septembre, 2024
Master CMSI, M2, Strasbourg

[*] MACARON project-team, Université de Strasbourg, CNRS, Inria, IRMA, France

Outline

Principe de la réduction d’ordre

Réduction linéaire

- POD
- Réduction et algorithme Glouton

Réduction nonlinéaire

- Méthodes de réduction quadratique
- Méthodes d’apprentissage de variété
- Auto-encodeur convolutif et sur graphe
- Autres approches

Principe de la réduction d'ordre

Principe de la réduction d'ordre

- On considère l'équation elliptique:

$$L(u(t,\mathbf{x};\boldsymbol{\mu})) = \partial_t u(t,\mathbf{x},\boldsymbol{\mu}) - \epsilon\Delta u + \mathbf{a}\cdot\nabla u(t,\mathbf{x},\boldsymbol{\mu}) + cu(t,\mathbf{x},\boldsymbol{\mu}) = f(\mathbf{x})$$

avec $\boldsymbol{\mu}$ l'ensemble des paramètres de l'EDP (paramètres de la source, coefficients physiques etc).

Méthodes numériques

- On se donne un espace d'approximation de dimension n
- On écrit un problème d'optimisation ou un système linéaire sur les paramètres θ
- **Objectif**: choisir un espace d'approximation assez riche pour approcher la solution pour n'importe quelles données du problèmes/-paramètres et avoir de la convergence.

Réduction d'ordre

- On construit un espace d'approximation de dimension d à partir de simulations précédentes pour certains paramètres $\boldsymbol{\mu}_i$
- On écrit un problème d'optimisation ou un système linéaire sur les paramètres $\mathbf{z}$
- **Objectif**: construire un espace d'approximation de très basse dimension pour un nombre réduits de données associées aux même type de paramètres.

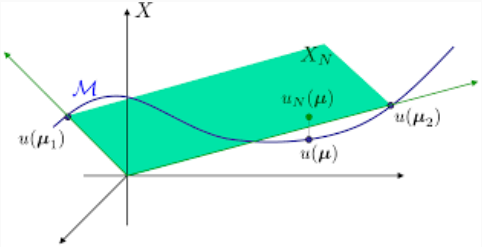
- **Point clé**: On construit l'espace d'approximation a partir de données. Deux étapes:
  - Comme construire l'espace d'approximation ?
  - Comment construire le model sur les variables latentes ?

Variété des solutions

Soit V_μ l'ensemble de paramètres considéré. L'espace des solutions de l'EDP associés aux paramètres $\boldsymbol{\mu} \in V_\mu$ est noté:

$$\mathcal{M} = \left\{ u(., t, \boldsymbol{\mu}) \in L^2(\Omega), \qquad \boldsymbol{\mu} \in V_\mu \right\}$$

est une variété de dimension m qu'on appelle la variété des solution.



- Déterminer la variété en essayant de déterminer $u(., t, \boldsymbol{\mu})$.
- Si cette variété très assez plate on peut l'approcher par un espace vectoriel de dimension $d \leqslant m$. On parle de réduction linéaire. On parle de réduction nonlinéaire.

Réduction linéaire vs nonlinéaire pour l'elliptique

Méthodes numériques linéaires

- Espace d'approximation:

$$u(\mathbf{x}) = \sum_{i=1}^{n} \theta_i \phi_i(\mathbf{x})$$

avec $\phi_i(\mathbf{x})$ des bases éléments finis ou autre et $\theta \in \mathbb{R}^n$.

Méthodes numériques nonlinéaires

- Espace d'approximation:

$$u(\mathbf{x}) = nn(\mathbf{x}; \theta)$$

avec $(\mathbf{x}; \theta)$ un réseau de neurones ou un autre modèle nonlinéaire.

Réduction d'ordre linéaire

- Espace d'approximation:

$$\mathcal{M} = \left\{ u(., \boldsymbol{\mu}) = \sum_{i=1}^{K} \theta_i(\boldsymbol{\mu}) \phi_i(\mathbf{x}) \in L^2(\Omega), \boldsymbol{\mu} \in V_{\mu} \right\}$$

avec $\phi_i(\mathbf{x})$ construites à partir de données. $K << n$. Les paramètres seront à déterminer pour chaque $\boldsymbol{\mu}$)

Réduction d'ordre nonlinéaire

- Espace d'approximation:

$$\mathcal{M} = \left\{ u(., \boldsymbol{\mu}) = nn(\mathbf{x}, \theta(\boldsymbol{\mu})) \in L^2(\Omega), \boldsymbol{\mu} \in V_{\mu}, \theta \in \mathbb{R}^K \right\}$$

avec $nn(.)$ construites à partir de données. $K << n$. Les paramètres seront à déterminer à pour chaque $\boldsymbol{\mu}$)

Réduction linéaire vs nonlinéaire pour le temporel

Méthodes numériques linéaires

- Espace d'approximation:

$$u(t, \mathbf{x}) = \sum_{i=1}^{n} \theta_i(t)\phi_i(\mathbf{x})$$

avec $\phi_i(\mathbf{x})$ des bases éléments finis ou autre et $\theta \in \mathbb{R}^n$. Les paramètres seront à déterminer à chaque temps.

Réduction d'ordre linéaire

- Espace d'approximation:

$$\mathcal{M} = \left\{ u(., \boldsymbol{\mu}) = \sum_{i=1}^{K} \theta_i(t, \boldsymbol{\mu})\phi_i(\mathbf{x}) \in L^2(\Omega), \boldsymbol{\mu} \in V_{\boldsymbol{\mu}} \right\}$$

avec $\phi_i(\mathbf{x})$ construites à partir de données. $K << n$. Les paramètres seront à déterminer à chaque temps et pour chaque $\boldsymbol{\mu}$)

Méthodes numériques nonlinéaires

- Espace d'approximation:

$$u(t, \mathbf{x}) = nn(\mathbf{x}; \theta(t))$$

avec $nn(\mathbf{x}; \theta(t))$ un réseau de neurones ou un autre modèle nonlinéaire. Les paramètres seront à déterminer à chaque temps.

Réduction d'ordre nonlinéaire

- Espace d'approximation:

$$\mathcal{M} = \left\{ u(., t, \boldsymbol{\mu}) = nn(\mathbf{x}, \theta(t, \boldsymbol{\mu})) \in L^2(\Omega), \boldsymbol{\mu} \in V_{\boldsymbol{\mu}}, \theta \in \mathbb{R}^K \right\}$$

avec $nn(.)$ construites à partir de données. $d << n$. Les paramètres seront à déterminer à chaque temps et pour chaque $\boldsymbol{\mu}$)

En pratique

- Pour construire l'espace d'approximation on va utiliser des simulations.
- On ne pourra donc pas construire, en général, un espace réduit analytique en $\mathbf{x}$.
- En pratique on discrétise en espace puis on construit l'espace d'approximation.

Réduction linéaire elliptique

- On se donne des données $\mathbf{u}(\boldsymbol{\mu}) \in \mathbb{R}^n$ une solution discrétisée en espace associée aux paramètres $\boldsymbol{\mu}$

$$\mathbf{u}(\boldsymbol{\mu}) = \mathbf{u}_{ref} + \Phi \hat{\mathbf{u}}(\boldsymbol{\mu})$$

avec $\mathbf{u}_{ref} \in \mathbb{R}^n$ a déterminer et $\hat{\mathbf{u}}(\boldsymbol{\mu}) \in \mathbb{R}^K$ avec $K <<< n$.

Réduction nonlinéaire elliptique

- On se donne des données $\mathbf{u}(\boldsymbol{\mu}) \in \mathbb{R}^n$ une solution discrétisée en espace associée aux paramètres $\boldsymbol{\mu}$

$$\mathbf{u}(\boldsymbol{\mu}) = G(u(\boldsymbol{\mu}))$$

avec $\mathbf{u}_{ref} \in \mathbb{R}^n$ a déterminer et $\hat{\mathbf{u}}(\boldsymbol{\mu}) \in \mathbb{R}^K$ avec $K << n$

Réduction linéaire temporelle

- On se donne des données $\mathbf{u}(t, \boldsymbol{\mu}) \in \mathbb{R}^n$ une solution discrétisée en espace associée aux paramètres $\boldsymbol{\mu}$

$$\mathbf{u}(t, \boldsymbol{\mu}) = \mathbf{u}_{ref} + \Phi \hat{\mathbf{u}}(t, \boldsymbol{\mu})$$

avec $\mathbf{u}_{ref} \in \mathbb{R}^n$ a déterminer et $\hat{\mathbf{u}}(t, \boldsymbol{\mu}) \in \mathbb{R}^K$ avec $K <<< n$.

Réduction nonlinéaire temporelle

- On se donne des données $\mathbf{u}(t, \boldsymbol{\mu}) \in \mathbb{R}^n$ une solution discrétisée en espace associée aux paramètres $\boldsymbol{\mu}$

$$\mathbf{u}(t, \boldsymbol{\mu}) = G(\hat{\mathbf{u}}(t, \boldsymbol{\mu}))$$

avec $\mathbf{u}_{ref} \in \mathbb{R}^n$ a déterminer et $\hat{\mathbf{u}}(t, \boldsymbol{\mu}) \in \mathbb{R}^K$ avec $K <<< n$

Réduction linéaire

1er objectif

- On se donne des m données issues de simulation.
  - **Cas elliptique**:
  $$X = [\mathbf{u}_h(\mu_m) - \mathbf{u}_{ref}, ..., \mathbf{u}_h(\mu_m) - \mathbf{u}_{ref}]$$
  - **Cas temporelle**:
  $$X = [\mathbf{u}_h(t_1, \mu_1) - \mathbf{u}_{ref}, ..., \mathbf{u}_h(t_m, \mu_m) - \mathbf{u}_{ref}]$$
- En general $\mathbf{u}_{ref}$ est construit facilement. Par exemple ce la peut être la moyenne des Snapshots

Objectif

On souhaite construire, à partir des données, $\Phi \in \mathbb{R}^{n,K}$ avec n la dimension de l'espace d'approximation et K la dimension de l'espace réduit.

Decomposition en mode orthogonaux

- On a donc $X \in \mathcal{M}_{n,m}(\mathbb{R}$ note de matrice contenant les simulations spatiales. On parle matrice de snapshots.
- Comment construire $\Phi \in \mathcal{M}_{n,K}$? Première solution: la POD.

POD

La méthode de POD consiste à construire l'opérateur Φ_K comme la solution du problème de minimisation:

$$\min_{\Phi_K \in \mathbb{R}^{n \times K}} \left\| X - \Phi_K \Phi_K^T X \right\|_F^2, \quad \text{sous contrainte } \Phi_K^T \Phi_K = I_d$$

avec $\|A\|_F$ la norme de Frobenius sur les matrices.

- On chercher donc a construire une compression/decompression linéaire optimal de la matrice X et composé de vecteurs orthogonaux.

Lien entre la POD et la SVD

Théorème de la SVD

La decomposition en valeur singulière (SVD) d'une matrice $X \in M_{n,m}(\mathbb{R})$ de rang r est donnée par

$$X = U\Sigma V^t = \sum_{i=1}^{r} \sigma_i u_i v_i^t$$

avec $U \in M_{n,r}(\mathbb{R})$ orthogonal, $V \in M_{m,r}(\mathbb{R})$ et Σ une matrice diagonale dans $M_{r,r}(\mathbb{R})$ avec $\sigma_1 > ... > \sigma_r > 0$. Soit la troncature:

$$\hat{X} = \sum_{i=1}^{K} \sigma_i u_i v_i^t$$

avec $K \leqslant r$ est la meilleur approximation de X (au sens de la norme de Frobinus) parmi les matrices de rang K. Cela équivaut à:

$$\| X - X^* \|_F = \min_{B \in M_{n,m}, RgB = K} \| X - B \|_F$$

- On peut obtenir U et Σ en diagonalisant XX^t. U sont les vecteurs propres et Σ contient les racines carrés de XX^t.

1er objectif

Solution de la POD

La solution $\Phi_K \in M_{n\times K}(\mathbb{R})$ est donnée par

$$\Phi_K = U_K$$

avec U_K les K premiers vecteurs de la matrice U de la SVD de la matrice de snapshots $X = U\Sigma V^t$.

- Démonstration:
  - On prend la matrice X pour lequel on calcul une décomposition en valeur singulière
    $$X = U\Sigma V^t$$
  - Le théorème de la SVD nous indique que
    $$X_K = U_K \Sigma_K V_K^t$$
    ou il s'agit de la restriction aux K premiers éléments est la meilleure approximation possible de rang K de la matrice X donc on a
  - On voit que si on prend $\Phi_K = U_K$ on a $\min_{B\ rg(B)=K} \|X - B\|_F^2 = \|X - X_K\|_F^2$
    $$\Phi_K \Phi_K^T X_k = U_K U_K^t X_k = U_K \underbrace{U_K^t U_K}_{=I_D} \Sigma_K V_K^t = X_K$$
  - On voit donc que le choix de $\Phi_K = U_K$ correspond à prendre la meilleure approximation de rang K de X et est donc la solution du problème d'origine.
- Exemple de POD: Lien

Méthode Gloutonne pour les problèmes stationnaires

Principe

L'idée de construire itérativement une base qui définira notre opérateur Φ en sélectionnant à chaque étape le snapshots qui permet de minimiser l'erreur d'approximation.

- **Algorithme** Glouton:
  - On choisit un premier vecteur
$$\Phi_1 = \frac{\mathbf{u}_h(\mu_1)}{\| \mathbf{u}_h(\mu_1) \|} \in \mathcal{M}_{n,1}(\mathbb{R})$$
  - Pour $l = 2$ et $l \leqslant K$:
    - Pour chaque i on construit
$$e_i = \| \mathbf{u}_h(\mu_i) - \Pi_{\mathcal{B}}(\mathbf{u}_h(\mu_i)) \|_2^2$$
avec $\Pi_{\mathcal{B}}(f)$ la projection de f sur la base $\mathcal{B} = (\Phi_1, ..\Phi_{l-1})$
    - On définit
$$\Phi_l^* = \text{argmax}_i \, e_i$$
    - Avec une itération de méthode de Gram-Schmidt on calcul Φ_l un vecteur de $Vec(\Phi_1, .., \Phi_{l-1}, \Phi_l^*)$ tel que $\Phi_1, .., \Phi_l$ forme une base orthonormale pour le produit scalaire $(X, (.,.))$.
  - Notre opérateur final est composé des vecteurs de bases calculés donc $\Phi_K = [\Phi_1, ..., \Phi_k]$.

- La projection orthogonal est donné par $\Pi_{\mathcal{B}}(\mathbf{f}) = \sum_{l=1}^{K} (\mathbf{f}, \phi_l)\phi_l$.
- L'étape de calcul de e_i peut être accélérée par des estimateurs d'erreur.

Méthode Gloutonne pour les problèmes temporels

- **Idée**: Soit on voit le temps comme un paramètre et on applique une méthode Gloutonne. Soit on combine POD et méthode gloutonne.
- **Algorithme** POD-Glouton:
  - On choisit un premier paramètre μ_1 associé à des snapshots.
  - On se donne comme base initiale $\mathcal{B}$ composé de N_0 snapshots orthogonalisés temporels associé a μ_1

$$\mathcal{B} = [\mathbf{x}(t_0, \mu_1), \ldots, \mathbf{x}(t_{N_t}, \mu_1)]$$

  - Pour $l = 1$ et $l \leqslant K$:
    -
    - On se donne comme base local $\mathcal{B}$ une version orthonormalisée de

$$\mathcal{B} = [\mathbf{x}(t_0, \mu_1), \ldots, \mathbf{x}(t_{N_t}, \mu_1)]$$

    - Pour chaque i on construit

$$e_i = \sum_{t=1}^{N_t} \| \mathbf{x}(t, \mu_i) - \Pi_{\mathcal{B}}(\mathbf{x}(t, \mu_i)) \|_2^2$$

      avec $\Pi_{\mathcal{B}}(f)$ la projection de f sur la base $\mathcal{B}$
    - On définit

$$\phi_l^* = \text{argmax}_i \, e_i$$

    - On calcul pour tout t:

$$\mathbf{s}(t, \mu_i^*) = \mathbf{x}(t, \mu_i^*) - \Pi_{\mathcal{B}}(\mathbf{s}(t, \mu_i^*))$$

    - On calcul la POD sur $[\mathbf{s}(t_1, \mu_i^*), \ldots, \mathbf{s}(N_t, \mu_i^*)]$ et conserve le premier mode qui devient ϕ_l le l'ème mode de la base $\mathcal{B}$
  - Notre opérateur final est composé des vecteurs de bases calculés donc $\Phi_K = [\phi_1, \ldots, \phi_K]$.

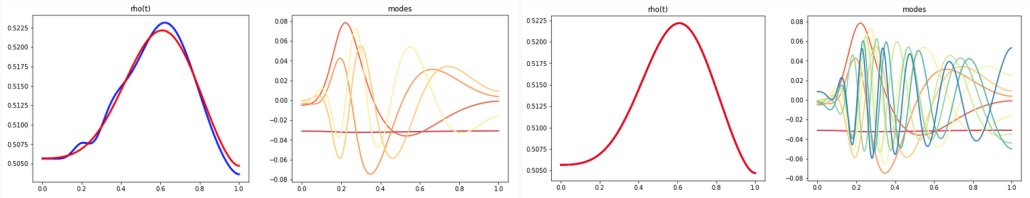
Réduction nonlinéaire

Limite de la réduction linéaire

- Jusqu'a présent on a supposer qu'un espace vectoriel de faible dimension suffisait a approcher la variété des solutions. Est ce toujours le cas ?
- On va regarder des résultats de compression + reduction (cours suivant) pour Burgers visqueux:

$$\partial_t \rho(t,x) + \partial_x \left(\frac{\rho^2}{2} \right) = \frac{1}{R_e} \partial_{xx} \rho$$

- On prend $R_e = 40$, (5 et 10 modes):



Conclusion

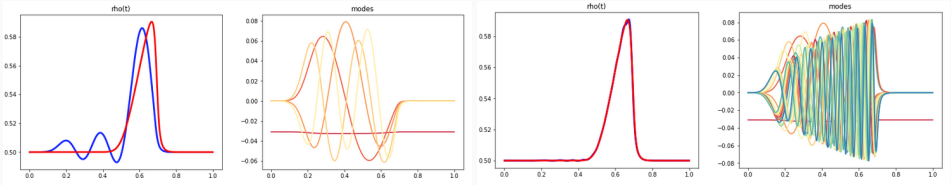
Pour les EDP de transport et fortement nonlinéaire, il est difficile de trouver une espace vectoriel de basse dimension pour approcher les données. Pour cette raison on considère des méthodes de réduction nonlinéaires.

Limite de la réduction linéaire

- Jusqu'a présent on a supposer qu'un espace vectoriel de faible dimension suffisait a approcher la variété des solutions. Est ce toujours le cas ?
- On va regarder des résultats de compression + reduction (cours suivant) pour Burgers visqueux:

$$\partial_t \rho(t,x) + \partial_x \left(\frac{\rho^2}{2} \right) = \frac{1}{R_e} \partial_{xx} \rho$$

- On prend $R_e = 4000$, (5 et 20 modes):



Conclusion

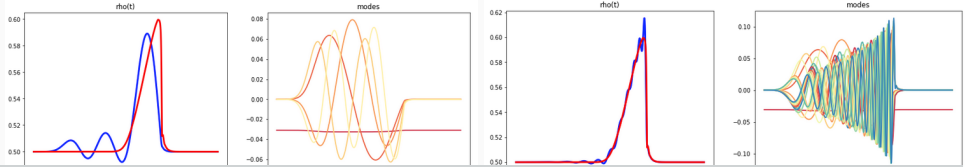
Pour les EDP de transport et fortement nonlinéaire, il est difficile de trouver une espace vectoriel de basse dimension pour approcher les données. Pour cette raison on considère des méthodes de réduction nonlinéaires.

Limite de la réduction linéaire

- Jusqu'a présent on a supposer qu'un espace vectoriel de faible dimension suffisait a approcher la variété des solutions. Est ce toujours le cas ?
- On va regarder des résultats de compression + reduction (cours suivant) pour Burgers visqueux:

$$\partial_t \rho(t,x) + \partial_x \left(\frac{\rho^2}{2} \right) = \frac{1}{R_e} \partial_{xx} \rho$$

- On prend $R_e = 400000$, (5 et 20 modes):



Conclusion

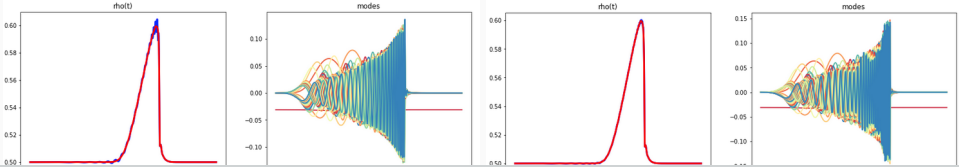
Pour les EDP de transport et fortement nonlinéaire, il est difficile de trouver une espace vectoriel de basse dimension pour approcher les données. Pour cette raison on considère des méthodes de réduction nonlinéaires.

Limite de la réduction linéaire

- Jusqu'a présent on a supposer qu'un espace vectoriel de faible dimension suffisait a approcher la variété des solutions. Est ce toujours le cas ?
- On va regarder des résultats de compression + reduction (cours suivant) pour Burgers visqueux:

$$\partial_t \rho(t,x) + \partial_x \left(\frac{\rho^2}{2} \right) = \frac{1}{R_e} \partial_{xx} \rho$$

- On prend $R_e = 400000$, (40 et 60 modes):



Conclusion

Pour les EDP de transport et fortement nonlinéaire, il est difficile de trouver une espace vectoriel de basse dimension pour approcher les données. Pour cette raison on considère des méthodes de réduction nonlinéaires.

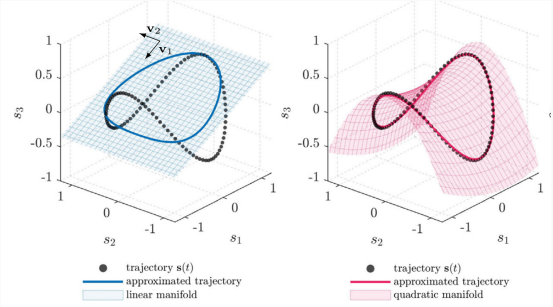
Réduction quadratique

- La première idée être de ce restreindre a des variété quadratique donc a des décodeur quadratique.
- **Décodeur linéaire**:

$$\mathbf{u}(t,\mu) = \mathbf{u}_{ref} + \Phi\hat{\mathbf{u}}(t,\mu)$$

- **Décodeur Quadratique**:

$$\mathbf{u}(t,\mu) = \mathbf{u}_{ref} + \Phi\hat{\mathbf{u}}(t,\mu) + \bar{\Phi}(\hat{\mathbf{u}}(t,\mu) \otimes \hat{\mathbf{u}}(t,\mu))$$



- Comment déterminer Φ et $\bar{\Phi}$?

Réduction quadratique II

Idée

Le terme linéaire est obtenu par POD et le terme quadratique par apprentissage linéaire.

- On prend donc $\Phi = \Phi_K$ issu de la POD. A partir de la on peut construire des snapshots réduits:

$$\widehat{\mathbf{u}}_j = \Phi^\top (\mathbf{u}_j - \mathbf{u}_{\text{ref}})$$

- Il nous reste a résoudre:

$$\overline{\mathbf{V}} = \underset{\in \mathbb{R}^{n \times K^2}}{\arg\min} \sum_{i=1}^{m} \left\| \mathbf{u}_i - \mathbf{u}_{\text{ref}} - \Phi \widehat{\mathbf{u}}_j - \overline{\Phi}(\widehat{\mathbf{u}}_i \otimes \widehat{\mathbf{u}}_i) \right\|_2^2,$$

avec m le nombre de données.

- On le reformule comme un problème aux moindre carrés auquel on ajoute une pénalisation

$$\overline{\Phi} = \underset{\overline{\Phi} \in \mathbb{R}^{n \times K^2}}{\arg\min} \frac{1}{2} \left\| \mathbf{W}^\top \overline{\mathbf{V}}^\top - \mathcal{E}^\top \right\|_F^2,$$

avec

$$\mathbf{W} := \begin{pmatrix} | & | & & | \\ \widehat{\mathbf{u}}_1 \otimes \widehat{\mathbf{u}}_1 & \widehat{\mathbf{u}}_2 \otimes \widehat{\mathbf{u}}_2 & \dots & \widehat{\mathbf{u}}_K \otimes \widehat{\mathbf{u}}_K \\ | & | & & | \end{pmatrix} \in \mathbb{R}^{K^2 \times m}.$$

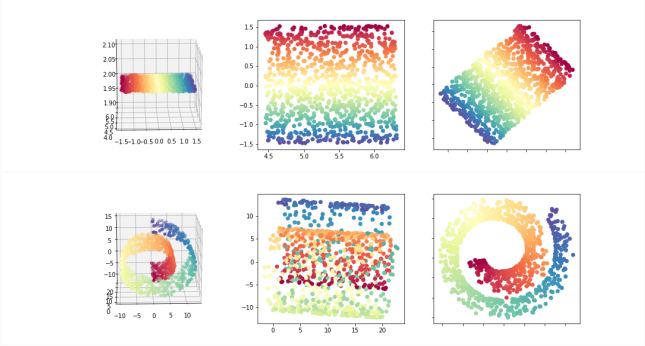
et

$$\mathcal{E} := \left(\mathbf{I} - \Phi \Phi^\top\right)(\mathbf{U} - \mathbf{U}_{\text{ref}})$$

le terme d'erreur sur les données.

Réduction de dimension et apprentissage de variété

- On souhaite maintenant aller au dela de variété quadratique.
- On peut utiliser des méthodes de réduction de dimension issue de l'apprentissage machine appelées: apprentissage de variété.
- On va comparer
  2 méthodes classiques: ACP (équivalent à la POD en statistique) et MDS (qu'on introduira):



- Ces deux méthodes qui propose un plongement linéaire ne permettent pas de capturer la variété dont sont issues les données.

Méthode MDS

- On suppose que on a nos données $(\mathbf{u}_1, ..., \mathbf{u}_m)$ ou $\mathbf{u}_i \in \mathbb{R}^n$.
- On souhaite déterminer des représentants en petite dimension $(\hat{\mathbf{u}}_1, ..., \hat{\mathbf{u}}_m)$ et des opérateurs qui permet de passer de la l'un à l'autre.

Idée de la méthode MDS

Proposer un opérateur de réduction de dimension qui préserve les distance des données entre elles lors du plongement.

Matrice de distance et de Gram

Soit une matrice $D \in \mathcal{M}_{m,m}(\mathbb{R})$. On dit qu'elle est une matrice de distance euclidienne si il existe $d > 0$ et $\mathcal{X} = (\mathbf{u}_1, ..., \mathbf{u}_m) \in \mathbb{R}^n$ tel que

$$D_{ij} = d_2(\mathbf{u}_i, \mathbf{u}_j)$$

et la matrice de distance carrée euclidienne est définie par $S_{ij} = d_2(\mathbf{u}_i, \mathbf{u}_j)^2$. On appelle la matrice de Gram associée, l a matrice $G \in \mathcal{M}_{m,m}(\mathbb{R})$ tel que:

$$G_{ij} = \langle \mathbf{u}_i, \mathbf{u}_j \rangle$$

Cette matrice est symétrique positive. Si elle est de rang r il existe une décomposition de Cholesky $G = X^t X$ avec $X = (\mathbf{u}_1, \ldots, \mathbf{u}_n) \in \mathcal{M}_{r,n}(\mathbb{R})$.

Méthode MDS II

- On nomme une matrice centré $A^c = HAH$ avec H la matrice de centrement:

$$H = I_n - \frac{1}{n}\mathbb{1} \otimes \mathbb{1}$$

  ou $\mathbb{1}$ le vecteur remplit de 1.

- On rappelle que

$$d_2(x, y) = \sqrt{\langle x, x\rangle + \langle y, y\rangle - 2\langle x, y\rangle}$$

  donc

$$D^c_{ij} = \sqrt{G^c_{ii} + G^c_{jj} - 2G^c_{ij}}$$

- **Théorème**: $G_c = -\frac{1}{2}S_c$ (admis).
- On peut montrer (voir cours) que pour calculer un plongement qui préserve les distances on diagonalise $G = X^tX$ la ou la POD/ACP diagonalise XX^t.

Algorithme

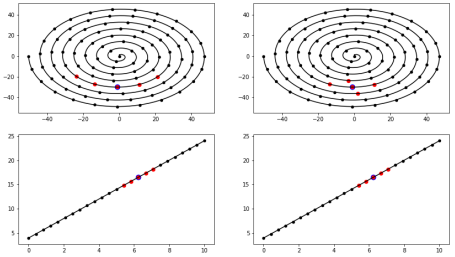
- On connait en entrée une matrice de distance D
- On calcule la matrice de Gram centrée $G = -\frac{1}{2}HD^2H$,
- On fait une décomposition spectrale de $G = \Phi\Lambda\Phi^t$ avec $\Phi = [\phi_1, \dots \phi_m]$ et $\text{Diag}([\lambda_1, \dots, \lambda_n])$
- On se restreint à avec $\Phi_d = [\phi_1, \dots \phi_K]$ et $\Lambda_k = [\lambda_1, \dots, \lambda_K]$ et on obtient la projection $Y = \sqrt{\Lambda_K}\Phi^t_K$

Méthode ISOMAP I

Idée Isomap

Pour obtenir un plongement qui détermine variété sur lequel vivent les données on va cherche un plongement qui préserve les distance au sens de la variété. On parle de géodésique.

- Distance géodésique et voisinage:



- Algorithme:
  - On va construire un graphe qui va essayer de capter la variété,
  - On calcul une approximation des distances géodésiques de la variété.
  - On applique MDS.

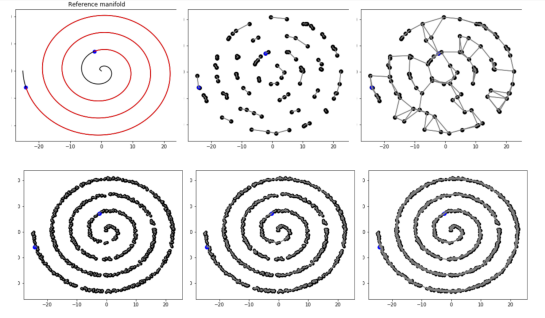
Méthode ISOMAP II

Graphe des plus proches voisins

Le graphe des plus proches voisins est le graphe à poids $\mathcal{G}(V, E)$ avec $V = (\mathbf{u}_1, \dots \mathbf{u}_m)$ les données et E l'ensemble des arêtes reliant tous les points tels que les poids sont donnés par

$$w_{ij} = \begin{cases} d(\mathbf{u}_i, \mathbf{u}_j) \text{ si } \mathbf{u}_j \in KNN(x_i), \\ \infty, \quad \text{sinon.} \end{cases}$$

avec $KNN(\mathbf{u})$ l'ensemble des voisins de $\mathbf{u}$ au sens de l'algorithme des k plus proches voisins.



Méthode ISOMAP III

Chemin le plus court sur un graphe

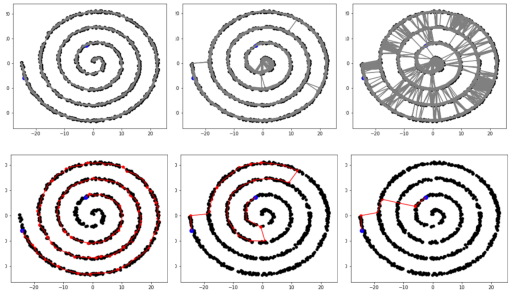
Soit $G = [V, A]$ un graphe. On définit un chemin entre x et y par:

$$\gamma(x, y) = ((x, z_1), (z_1, z_2), \ldots, (z_n, y))$$

une succession d'arête qui relie les deux point. On définit la distance par

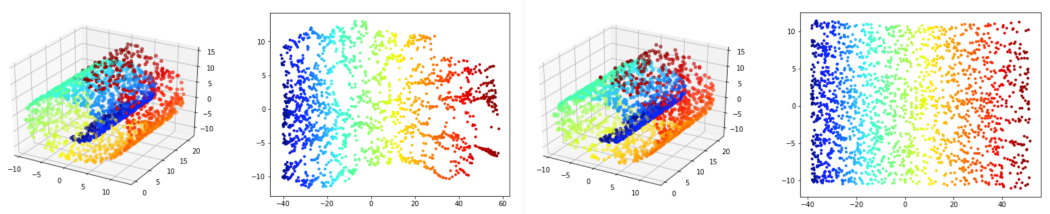
$$\text{dis}(x, y) = \min_{\gamma} |\gamma|$$

avec $|\gamma|$. La longueur du chemin est donnée par la somme des poids sur les arêtes du chemin.

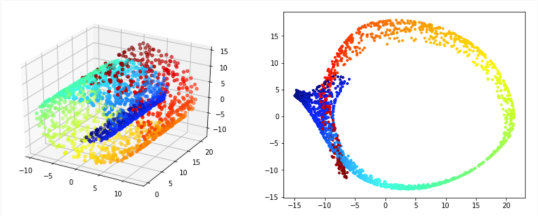


Méthode ISOMAP IV

- ISOMAP et découverte de variété.



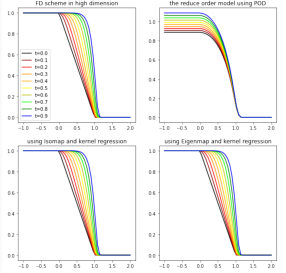
- 5 voisins et 10 voisins.



- 20 voisins.
- Il existe d'autre approches comme Eigemaps ou LLE.

Apprentissage de variété et réduction de modèle

- Comparaison entre la ses solutions de références, la POD avec plusieurs modes, ISOMAP et EigenMaps avec un mode.



- Les méthodes ISOMAP/EigenMaps compressent les exemples mais ne permet pas de compresser de nouveaux exemples.
- Si on a $(\mathbf{u}_1, ..., \mathbf{u}_m)$ on calcul $(\hat{\mathbf{u}}_1, ..., \hat{\mathbf{u}}_m)$ par ses méthodes. Ensuite on construit les opérateurs de décompression et compression par apprentissage supervisé.

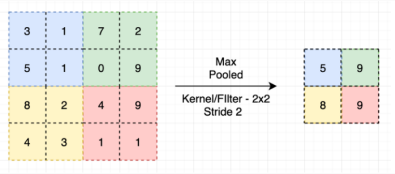
$$\min_{\theta_e} \sum_{i=1}^{N} \|E_{\theta_e}(\mathbf{u}_i) - \hat{\mathbf{u}}_i\|_2^2, \qquad \min_{\theta_d} \sum_{i=1}^{N} \left\|\mathbf{u}_i - D_{\theta_d}(\hat{\mathbf{u}}_i)\right\|_2^2$$

- Merci à C. Schnoebelen pour les simulations.

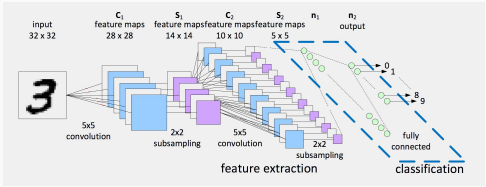
Architecture CNN

Couche de sous-échantillonnage

Soit un signal x en dimension d avec n données par dimension. L'opérateur de pooling est un opérateur de $\mathbb{R}^{n^d}$ vers $\mathbb{R}^{\left(\frac{n}{k}\right)^d}$ ou $k \times d$ pixel sont transformés en 1 pixel. Si on prend la moyenne on parle pooling moyen local. Si on prend le maximum on parle pooling maximal local.



- Architecture globale:



Auto-encodeur convolutif et reduction de dimension

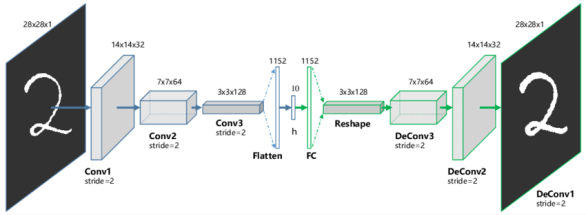
- L'idée est de voir un snapshot d'EDP comme une image et d'appliquer des réseau CNN pour construire des opérateurs de compression/décompression.

Auto-encodeur

Soit $X \subset \mathbb{R}^n$ et $Z \subset \mathbf{R}^K$ avec $K << n$. On nomme encodeur une fonction paramétrique $E_{\theta_e} : X \to Z$ et on nomme décodeur une fonction paramétrique $D_{\theta_d} : Z \to X$. Soit $(\mathbf{u}_1, \dots \mathbf{u}_n)$ de $\mathbb{R}^n$. On nomme auto-encodeur un couple encodeur-décodeur solution du problème de minimisation:

$$\sum_{i=1}^{m} \left\| \mathbf{u}_i - D_{\theta_d}(E_{\theta_e})(\mathbf{u}_i) \right\|_2^2 + R(\theta_e, \theta_d)$$

avec R un terme de régularisation.



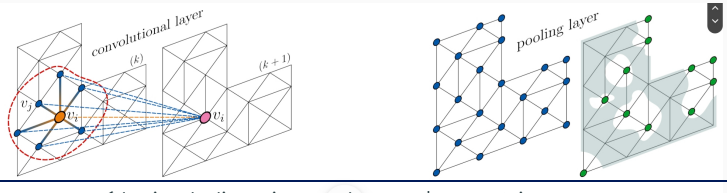
Polling sur graphe/maillage

- Ces réseaux peuvent traiter des maillages. Suffisant pour construire un auto-encodeur ?
- Pour appliquer ses réseaux à de la reduction de dimension ou les solutions vivrait une maillage général on a besoin de ”pooling”.

Remarque

La plupart des ”pooling” de la littérature ne sont pas adaptés car ils proposent un graphe plus grossier en fonctions des données **x**. Pour la réduction de dimension on a juste envie de fusionner les noeuds géométriquement proche.

- Exemple de stratégie:
  - on enlève aléatoirement des noeuds,
  - on fait du clustering,
- Dans le deux cas on stocke les graphes intermédiaires. Pour la remonter on peut faire des interpolations naives pour récupérer de l'informations aux nouveaux noeuds.



POD-NN auto-encodeur

- Les approches basés sur les réseaux ont des limites
  - Elle ne sont pas forcement adaptés à tout type de données d'entrées. Par exemple construire des auto-encodeurs pour des nuages de points (méthodes particulaires) est difficile.
  - Les approches type GNN peuvent devenir très couteuses notamment appliquées aux maillage fins.

Auto-encodeur POD-NN

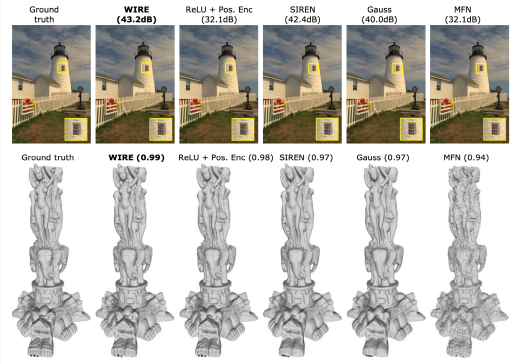
L'idée est de mélanger une approche POD avec une approche neural. On propose donc d'abord de construire ϕ_L par POD avec $n >> L >> K$. Ensuite on propose

$$D(.) = \Phi_L \circ nn_{d,\theta}, \quad E(.) = \Phi_L nn_{e,\theta} \circ \Phi_L^t$$

qu'on peut entrainer avec une fonction de coût d'un AE classique.

Representation implicite neuronales

- **Idée**: représenter les objets discret (image, function video) qu'on cherche par des réseaux type MLP basé sur les coordonnées.
- Les opérations sur ses objets peuvent devenir des opérations sur les poids.



- Utile en compression, en géométrie numérique.
- Les PINNs c'est une représentation implicite neuronale.

Auto encodeur et représentation neuronales

- Si on reprend l'idée des representation neurales, l'idée est d'avoir a la fin un objet dépendant continuement de $\mathbf{x}$

Auto-encodeur basé sur les représentations neuronales implicites

Si on appelle $\hat{\mathbf{u}}$ les variables latentes, l'idée est d'utilisé un décodeur de la forme

$$D_\theta(\mathbf{x}; \hat{\mathbf{u}})$$

avec D_θ un MLP, KAN, réseau de Fourier ou une base de fonctions (réseaux, SIndy etc)

- Cela revient de connaitre le décodeur partout et d'être invariant à la discrétisation.
- On peut ajouter une fonction de cout physique à l'entrainement. On se rapproche des PINNs paramétriques mais ici les paramètres sont découvert par l'encodeur à partir de données.
- Reste l'encodeur ? L'encodeur prend une fonction et la transforme en un vecteur. Il ne peut pas être local en $\mathbf{x}$. On va proposer deux familles de méthodes.

Auto encodeur et représentation neuronales II

Encodeur global

On va se donne une fonction **u** et des valeurs d'une fonction a des points (qu'on peut appeler senseur):

$$\{\mathbf{u}(\mathbf{x}_1), ...\mathbf{u}(\mathbf{x}_m)\}$$

L'encodeur sera un réseau de la forme:

$$E_\theta(\mathbf{u}(\mathbf{x}_1), ...\mathbf{u}(\mathbf{x}_m)) \to \hat{\mathbf{u}}, \qquad \text{ou } E_\theta((\mathbf{x}_1, ..., \mathbf{x}_m), \mathbf{u}(\mathbf{x}_1), ...\mathbf{u}(\mathbf{x}_m)) \to \hat{\mathbf{u}}$$

Typiquement cela peut être un gros MLP (les positions des points seront fixe), un encodeur CNN ou GNN etc

Encodeur par entrainement

Pour chaque donnée on calcule un code latent en inversant le décodeur.

Algorithme:

- $\forall e < n_{epochs}$
  - $\forall i < m$: on on résout
  $$\hat{\mathbf{u}}_i = \min_{\mathbf{z}} \sum_{j=1}^{n} \| \left(\mathbf{u}(\mathbf{x}_j) - D_{\theta^*}\left(\mathbf{z}; \mathbf{x}_j\right)\right\|_2^2$$
  - On résout
  $$\theta^* = \min_{\theta} \sum_{i=1}^{m} \sum_{j=1}^{n} \| \left(\mathbf{u}^i(\mathbf{x}_j) - D_\theta\left(\hat{\mathbf{u}}_i, \mathbf{x}_j\right)\right\|_2^2$$