La dernière réunion...

Projet M1 – SciML

Semaine 5

Rapide

Synthétique

Par Williams & Robin

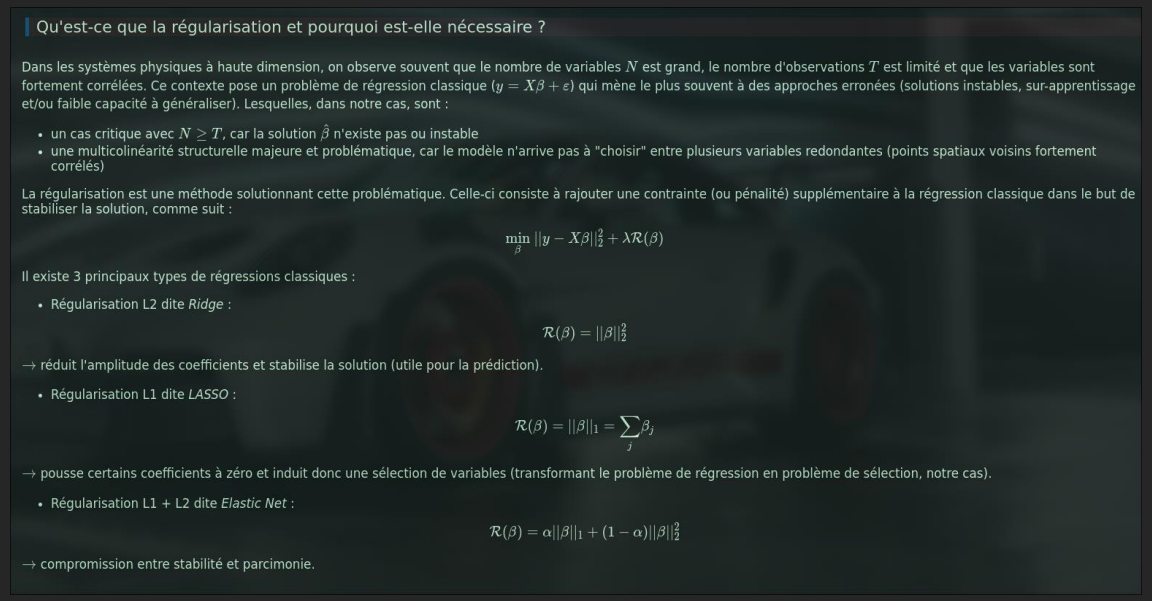
Ce qu'on a fait la semaine dernière ...

Fin pré-traitement & ML - SciML

Compteur de commits : 106

4 notebooks :

10 - Analyse multivariée (modification)
14 Bis - Réduction d'état par sélection
17 - Mémoire markovienne
18 - Modélisation ML classique



Qu'est-ce que la régularisation et pourquoi est-elle nécessaire ?

Dans les systèmes physiques à haute dimension, on observe souvent que le nombre de variables N est grand, le nombre d'observations T est limité et que les variables sont fortement corrélées. Ce contexte pose un problème de régression classique ($y = X\beta + \varepsilon$) qui mène le plus souvent à des approches erronées (solutions instables, sur-apprentissage et/ou faible capacité à généraliser). Lesquelles, dans notre cas, sont :

- un cas critique avec $N \geq T$, car la solution $\hat{\beta}$ n'existe pas ou instable
- une multicolinéarité structurelle majeure et problématique, car le modèle n'arrive pas à "choisir" entre plusieurs variables redondantes (points spatiaux voisins fortement corrélés)

La régularisation est une méthode solutionnant cette problématique. Celle-ci consiste à rajouter une contrainte (ou pénalité) supplémentaire à la régression classique dans le but de stabiliser la solution, comme suit :

$$\min_{\beta} ||y - X\beta||_2^2 + \lambda \mathcal{R}(\beta)$$

Il existe 3 principaux types de régressions classiques :

- Régularisation L2 dite *Ridge* :

$$\mathcal{R}(\beta) = ||\beta||_2^2$$

→ réduit l'amplitude des coefficients et stabilise la solution (utile pour la prédiction).

- Régularisation L1 dite *LASSO* :

$$\mathcal{R}(\beta) = ||\beta||_1 = \sum_j \beta_j$$

→ pousse certains coefficients à zéro et induit donc une sélection de variables (transformant le problème de régression en problème de sélection, notre cas).

- Régularisation L1 + L2 dite *Elastic Net* :

$$\mathcal{R}(\beta) = \alpha||\beta||_1 + (1-\alpha)||\beta||_2^2$$

→ compromission entre stabilité et parcimonie.

```python
from sklearn.linear_model import Ridge
from sklearn.multioutput import MultiOutputRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit, GridSearchCV
from sklearn.metrics import mean_squared_error
import numpy as np

# Modèle de base
ridge_base = MultiOutputRegressor(Ridge())

# Pipeline de prétraitement et de modèle
pipe_ridge = Pipeline([
    ("scaler", StandardScaler()),
    ("ridge", ridge_base)
])

# Hyperparamètres
param_grid_ridge = {
    "ridge__estimator__alpha": [0.01, 0.1, 1, 10, 100]
}

# Apprentissage passé puis on valide sur le futur
tscv = TimeSeriesSplit(n_splits=5)

grid_ridge = GridSearchCV(
    pipe_ridge,
    param_grid_ridge,
    cv=tscv,
    scoring="neg_mean_squared_error",
    n_jobs=-1
)

grid_ridge.fit(X_train, y_train)

ridge_best = grid_ridge.best_estimator_

print("Meilleurs hyperparamètres Ridge :", grid_ridge.best_params_)
print("MSE CV Ridge :", -grid_ridge.best_score_)
```

```
Meilleurs hyperparamètres Ridge : {'ridge__estimator__alpha': 100}
MSE CV Ridge : 0.718700647354126
```

Relus, uniformes et lisibles

Ce qu'on a fait en 10 jours ...

ML & SciML

Compteur de commits : 123

3 modèles et plus :

Neural ODE (NSDE et NSDE Latent en test)

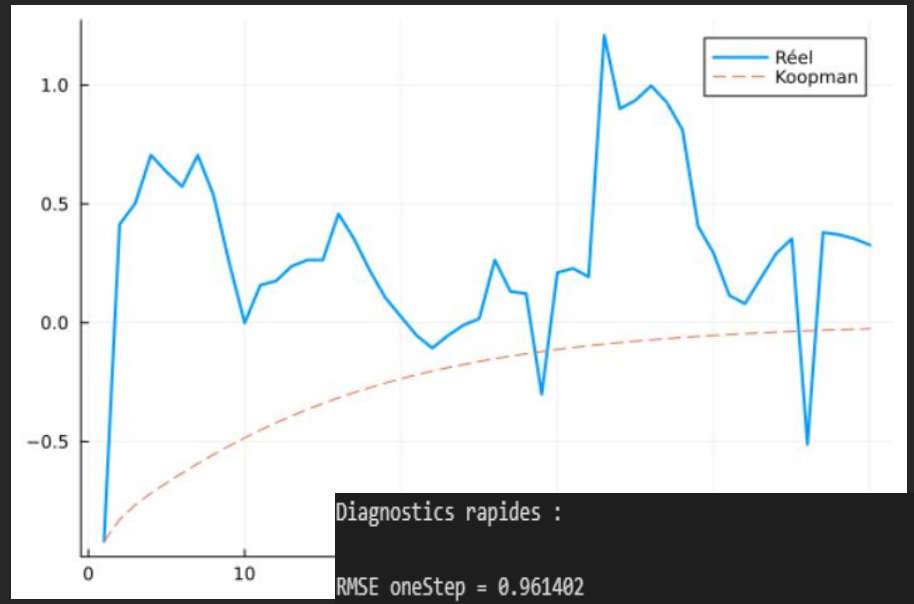
Koopman

SINDy

Modèles classiques (Kernel Ridge et Random Forest)

Notebook – Tuto Julia

Premiers résultats




```
Diagnostics rapides :

RMSE oneStep = 0.961402

RMSE multiStep moy = 1.1296895

RMSE multiStep (5 premiers pas) =
loat32[0.0, 0.5881988, 1.2120327, 0.99952126, 0.9884687]

Mode EOF numéro 1 réel contre prédit (5 premiers temps) :

Réel = Float32[0.71609473, 0.6209254, 0.4138982, 0.55332565, 0.6048181]

Prédit = Float32[0.71609473, 0.6845454, 0.6485234, 0.61538213, 0.58562994]
```

Ce qu'on a fait en 7 jours ...

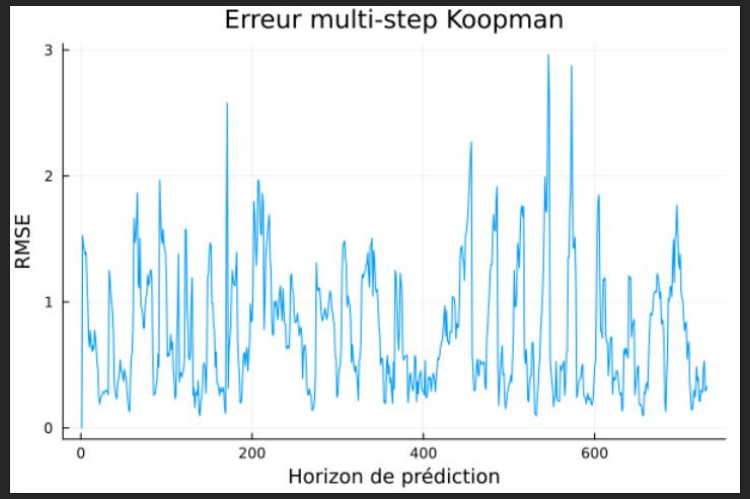
Les résultats majeurs

Williams

Latex présentation de Julia + notebook tuto

Approfondissement du ML classique
- Kernel Ridge
- RandomForest avec Gradient Boosting

Mise en place de Koopman



Robin

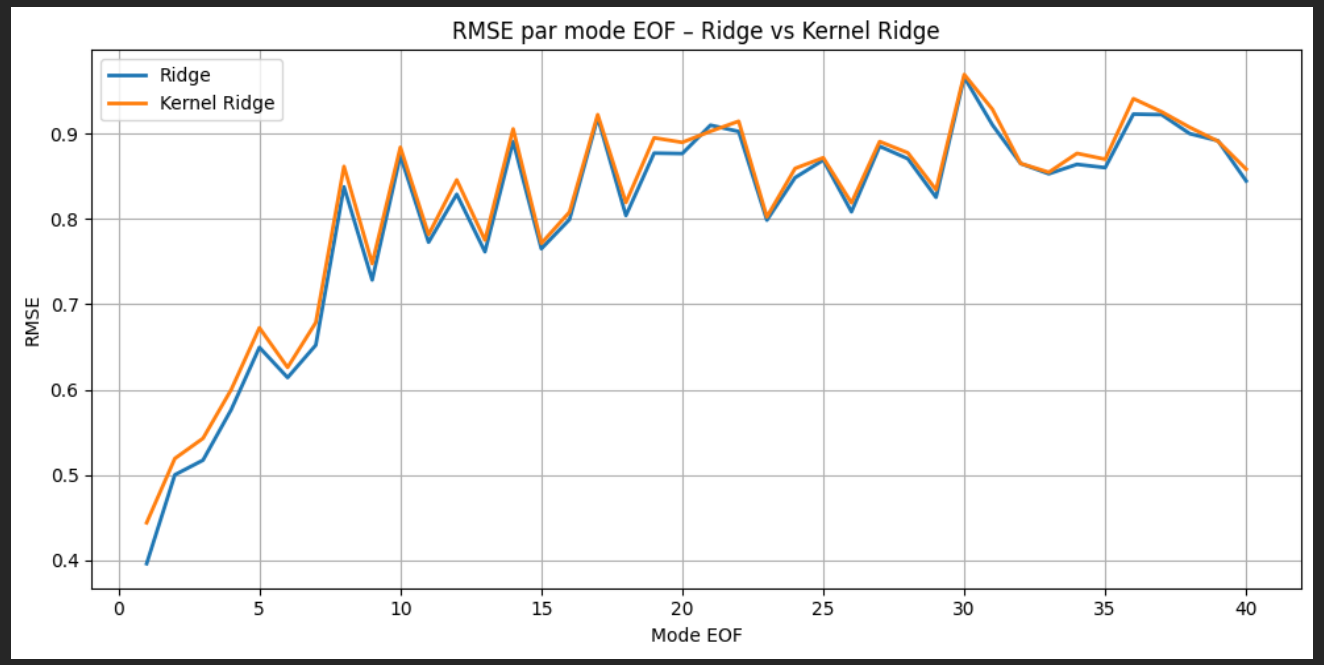
Neural ODE, SDE, SDE Latent

SINDy

Résultats de Williams

Approfondissement du ML classique

Kernel Ridge :



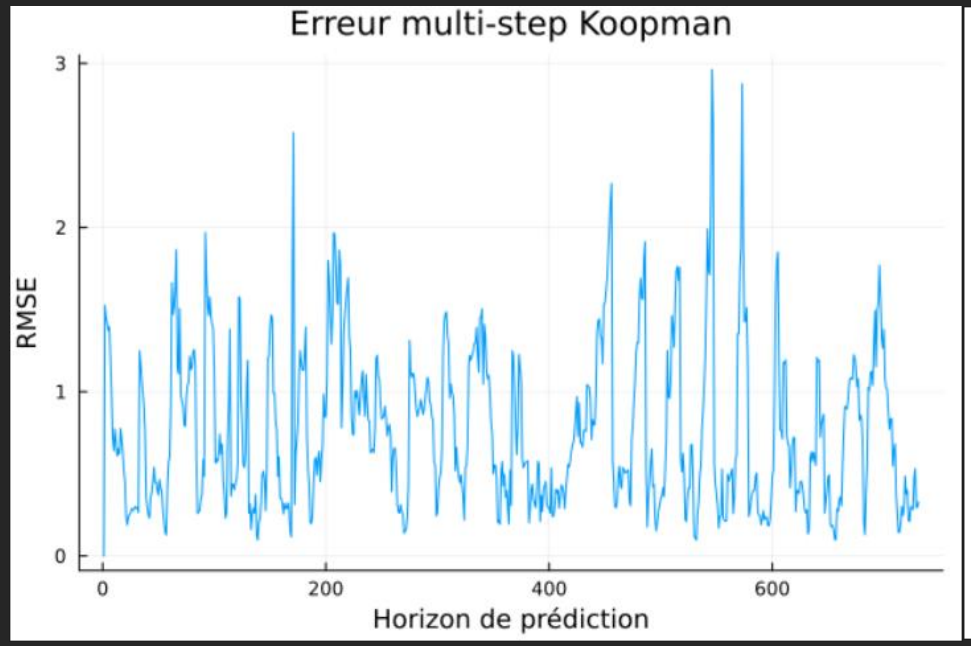
RandomForest :

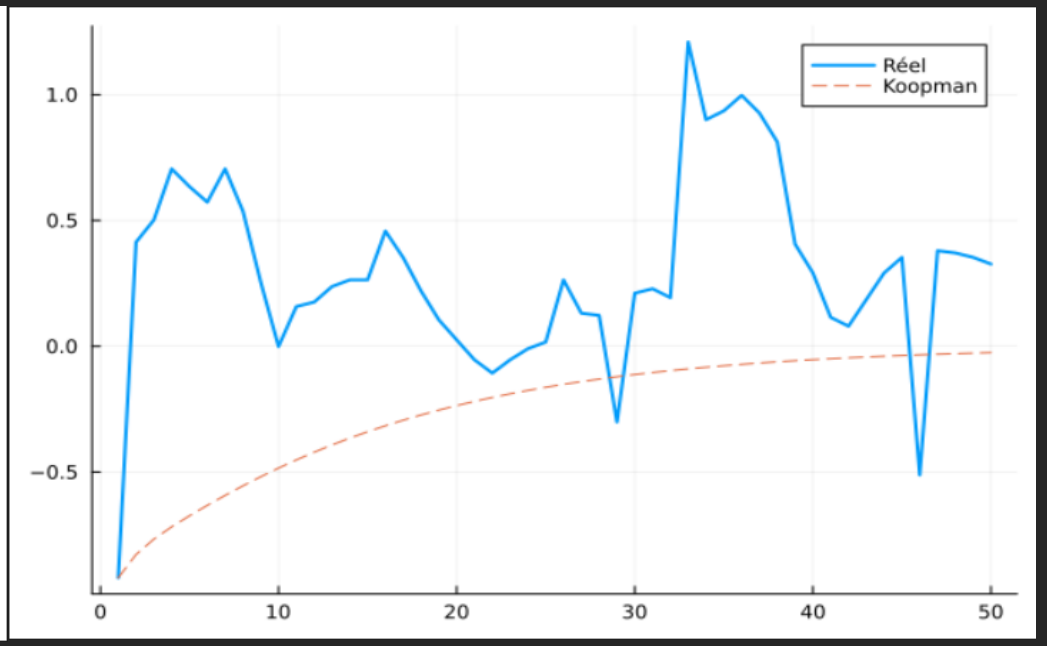
- RMSE global = 1.19
- Plus élevé que Ridge et Kernel Ridge

On observe une dégradation dès les premiers modes
Erreurs élevées et homogènes sur les modes intermédiaires et élevés.

Résultats de Williams

Implémentation de Koopman





La méthode de Koopman consiste à représenter l'évolution d'un système non linéaire à l'aide d'un opérateur linéaire agissant sur un espace d'états élargi ou réduit.

Résultats :

- ne diverge pas
- oscille entre ~0.2 et ~2–3
- reste bornée sur tout l'horizon (~700 pas)

Résultats :

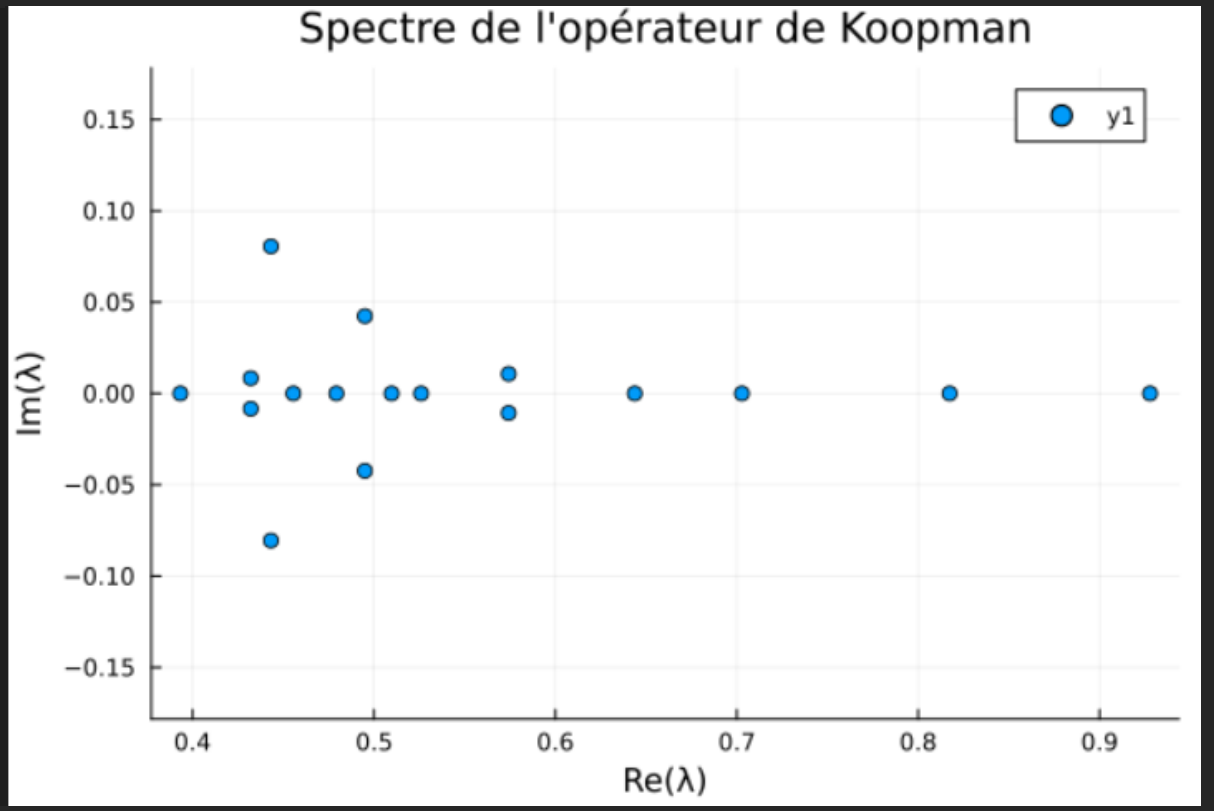
La courbe Koopman (pointillés) :

- est lisse
- suit une tendance lente
- converge vers un régime moyen

Résultats de Williams

Implémentation de Koopman

```
RMSE one step Koopman sur la validation = 0.4287795
RMSE multi step moyen sur la validation = 0.7657934

 Diagnostics rapides :

 RMSE oneStep = 0.4287795

 RMSE multiStep moy = 0.7657934

 RMSE multiStep (5 premiers pas) =
Float32[0.0, 1.5256636, 1.4518385, 1.4195274, 1.3727897]

 Mode EOF numéro 1 réel contre prédit (5 premiers temps) :

 Réel = Float32[-0.92024505, 0.41394845, 0.5022059, 0.7051955, 0.6345787]

 Prédit = Float32[-0.92024505, -0.8277833, -0.7665781, -0.71763706, -0.6737109]
```



Ce qu'on va faire pendant les 7 prochains jours ...

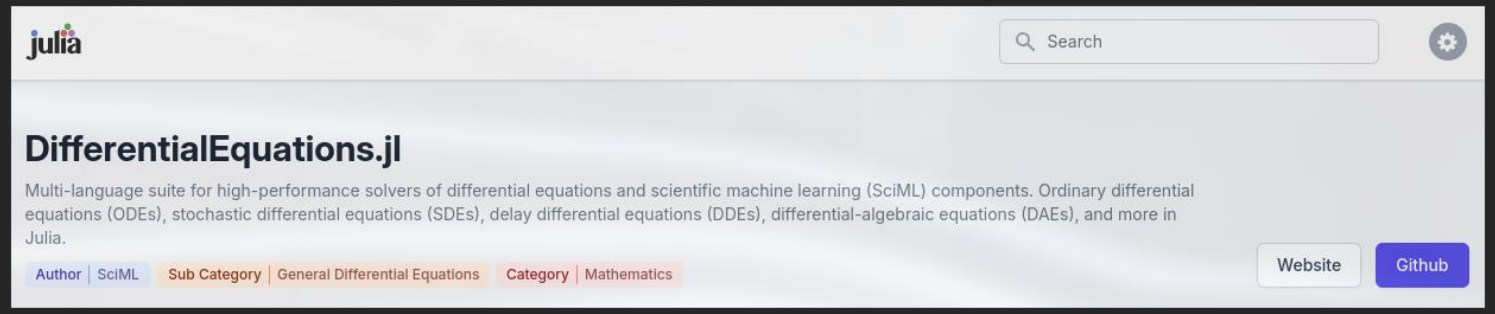
SciML

Tutoriel installation + utilisation de Julia

Rapport de présentation de Julia

Guide théorique de la modélisation SciML + choix modèles à implémenter

Implémentation de premiers modèles SciML



Merci de votre écoute !

Template inspirée de **Noé** aka **SkohTV**