MMGP: a Mesh Morphing Gaussian Process-based machine learning method for regression of physical problems under nonparametrized geometrical variability

Fabien Casenave

Brian Staber

Xavier Roynard

NeurIPS 2023

Introduction

Learning solutions to nonlinear PDEs

- Learn and predict physical quantities of interest from **mesh-based representations**
- Geometric variability and arbitrary mesh connectivities
- Recent works mostly relying on deep learning (see, e.g., Pfaff et al. 2020)

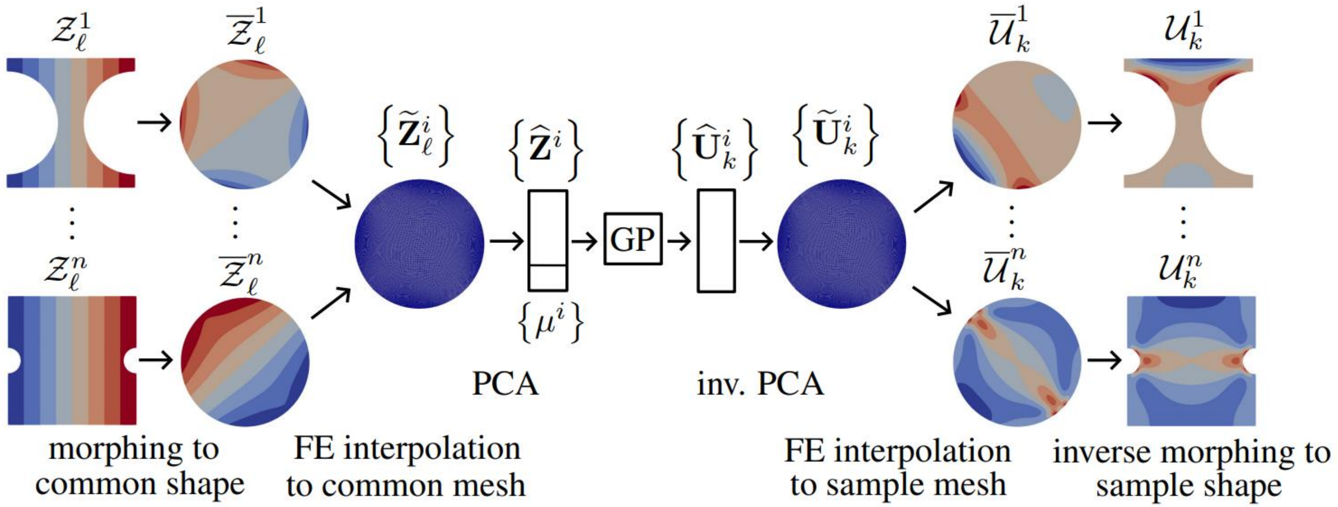
Contributions

- Efficient learning method relying on **deterministic preprocessings** and **classical machine learning** tools
- Comes with **predictive uncertainties out off the shelf**
- Trainable on CPU hardware
- Competitive with recent graph neural networks, *on our experiments*

Assumptions

- All geometries share a **common topology**
- **Meshes contain CAD info** (points and surfaces of physical interest)
- Meshes have « good quality » (used for num. simulation)
- Geometrical variations are controlled, **noise free** and **constrained to avoid extreme distortion** (admissible designs)

Mesh morphing Gaussian process



Remarks

- The morphing algo is chosen a priori
- **Shape embedding** is morphing + FE interpolation of the vertices coordinate fields seen as continuous fields + dim. red. using PCA
- The overall red. dim. **is highly nonlinear due to morphing**

- GPs are in **low dim. inputs and outputs**
- GPs enjoy **universal approximation theorem**
- GPs enable **efficient predictive UQ**

Preprocessing steps

Morphing strategies

- Many algorithmes available in the literature
- Tutte’s barycentric mapping
- Radial Basis Functions

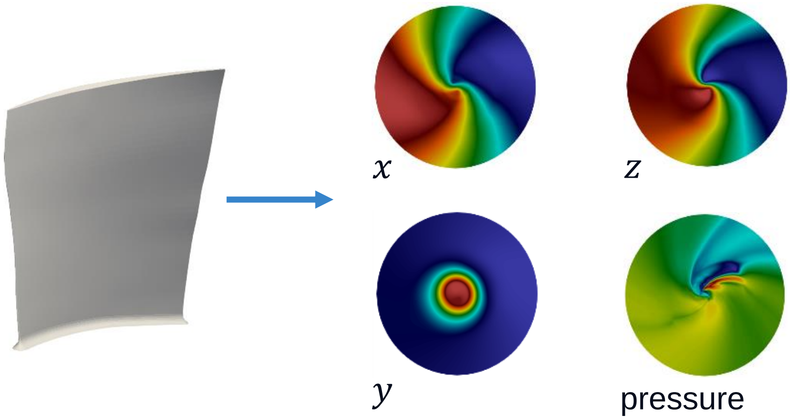
Finite element interpolation

- Compatible with any element type ($\mathcal{P}_k, \mathcal{Q}_k, \ldots$)
- Efficient C++ implementation

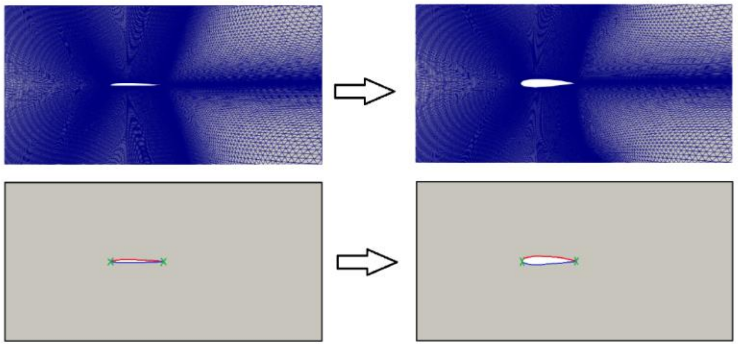
Dimensionality reduction

- Principal component analysis
- Snapshot proper orthogonal decomposition

Morphing examples



Tutte’s barycentric mapping



Radial Basis Functions

Mesh morphing Gaussian process

Training for an output scalar

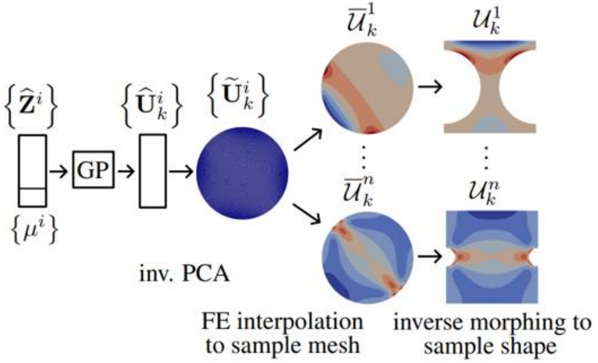
- Train a 1D Gaussian process
- Inputs: low-dimensional embeddings of the meshes
- Outputs: scalar quantity of interest

$$\mathbb{E}[w^\star] = \mathbf{k}_\star^T(\mathbf{K}+\sigma^2\mathbf{I})^{-1}\mathbf{w}_{m_0}\,,$$
$$\mathbb{V}[w^\star] = K_{\star\star} - \mathbf{k}_\star^T(\mathbf{K}+\sigma^2\mathbf{I})^{-1}\mathbf{k}_\star$$

Predictive mean and variance of the 1D Gaussian process

Training for an output field

- Train a multi-output Gaussian process
- Inputs: low-dimensional embeddings of the meshes
- Outputs: low-dimensional embeddings of the output fields
- Inverse PCA, finite element interpolation, and inverse morphing to predict the field
- Uncertainty quantification through Monte Carlo



Numerical experiments

Datasets

- Three datasets in computational fluid dynamics and solid mechanics
- Geometric variabilities
- Meshes with different number of nodes and adjacency matrices
- Multiple output scalars and output fields

Table 1: Summary of the considered datasets with d_Ω: dimension of the physical problem, p: number of input scalars, d: number of output fields, m: number of output scalars.

Datasets	train/test sizes	d_Ω	p	d	m	Avg. # nodes
`Rotor37`	1000/200	3	2	2	4	29,773
`Tensile2d`	500/200	2	6	6	4	9,425
`AirfRANS`	800/200	2	2	3	2	179,779
`AirfRANS-remeshed`	800/200	2	2	3	2	19,527

Numerical experiments

Setup & results

- Comparative study with graph convolutional neural networks (**GCNN**) and MeshGraphNets (**MGN**)
- Hyperparameter tuning with grid searches for GNNs
- **MMGP** is highly competitive both in accuracy and training computational times

Table 2: Means and standard deviations (gray) of the relative RMSE and Q^2 scalar regression coefficients for all the considered datasets and quantities of interest (QoI) (best is **bold**).

	RRMSE			Q^2		
QoI	GCNN	MGN	MMGP	GCNN	MGN	MMGP
`Rotor37` dataset						
m	4.4e-3 (5e-4)	5.4e-3 (7e-5)	**5.0e-4** (3e-6)	0.9816 (4e-3)	0.9720 (5e-4)	**0.9998** (3e-6)
p	4.4e-3 (5e-4)	5.3e-3 (7e-5)	**4.8e-4** (1e-6)	0.9803 (5e-3)	0.9710 (9e-4)	**0.9998** (2e-6)
η	3.1e-3 (7e-4)	7.2e-3 (7e-5)	**5.0e-4** (3e-6)	0.9145 (4e-2)	0.5551 (2e-3)	**0.9979** (1e-6)
γ	2.9e-3 (6e-4)	6.5e-3 (2e-5)	**4.6e-4** (2e-7)	0.9068 (4e-2)	0.5257 (2e-3)	**0.9977** (2e-6)
P	1.7e-2 (8e-4)	1.7e-2 (2e-3)	**7.2e-3** (5e-4)	0.9863 (1e-3)	0.9866 (3e-3)	**0.9973** (4e-4)
T	3.9e-3 (1e-4)	1.4e-2 (2e-3)	**8.2e-4** (1e-5)	0.9930 (5e-4)	0.9956 (1e-3)	**0.9997** (1e-5)
`Tensile2d` dataset						
$p_{\max}$	1.6e-0 (7e-1)	**2.7e-1** (4e-2)	6.6e-1 (3e-1)	0.4310 (2e-1)	0.6400 (2e-1)	**0.9435** (2e-2)
$v_{\max}$	4.4e-2 (7e-3)	5.8e-2 (2e-2)	**5.0e-3** (3e-5)	0.9245 (3e-2)	0.9830 (1e-2)	**0.9999** (2e-5)
$\sigma_{22}^{\max}$	3.1e-3 (7e-4)	4.5e-3 (1e-3)	**1.7e-3** (2e-5)	0.9975 (1e-3)	0.9958 (1e-3)	**0.9993** (2e-5)
$\sigma_v^{\max}$	1.2e-1 (4e-2)	2.4e-2 (9e-3)	**5.0e-3** (3e-5)	0.9723 (2e-2)	0.9801 (1e-2)	**0.9997** (7e-6)
u	4.5e-2 (1e-2)	1.5e-2 (1e-3)	**3.4e-3** (4e-5)	0.9623 (2e-2)	0.9270 (1e-2)	**0.9997** (6e-6)
v	7.4e-2 (2e-2)	9.7e-2 (7e-3)	**5.5e-3** (8e-5)	0.9559 (3e-2)	0.9322 (1e-2)	**0.9995** (1e-5)
p	1.3e-1 (7e-2)	1.1e-1 (2e-2)	**4.4e-2** (1e-2)	0.5691 (1e-1)	0.2626 (1e-1)	**0.7785** (9e-2)
σ_{11}	1.0e-1 (4e-2)	2.8e-2 (3e-3)	**3.7e-3** (1e-4)	0.9304 (4e-2)	0.8693 (3e-2)	**0.9999** (2e-6)
σ_{12}	4.5e-2 (4e-3)	7.5e-3 (4e-4)	**2.4e-3** (2e-5)	0.9617 (5e-3)	0.9868 (1e-3)	**0.9999** (1e-6)
σ_{22}	3.3e-2 (3e-3)	2.7e-2 (1e-3)	**1.4e-3** (1e-5)	0.9662 (6e-3)	0.9782 (2e-3)	**0.9999** (1e-6)
`AirfRANS` dataset						
C_D	6.1e-2 (2e-2)	4.9e-2 (7e-3)	**3.3e-2** (2e-3)	0.9596 (2e-2)	0.9743 (1e-2)	**0.9831** (2e-3)
C_L	4.1e-1 (1e-1)	2.4e-1 (8e-2)	**8.0e-3** (6e-4)	0.9776 (8e-3)	0.9851 (1e-2)	**0.9999** (2e-6)
u	5.6e-2 (3e-3)	8.3e-2 (2e-3)	**1.8e-2** (9e-5)	0.9659 (3e-3)	0.9110 (3e-3)	**0.9749** (8e-5)
v	4.2e-2 (2e-3)	1.2e-1 (2e-3)	**1.5e-2** (3e-5)	0.9683 (3e-3)	0.7516 (5e-3)	**0.9806** (3e-5)
p	8.5e-2 (7e-3)	9.9e-2 (1e-2)	**5.1e-2** (2e-5)	0.9602 (8e-3)	0.9390 (2e-2)	**0.9934** (1e-5)

Table 3: Training computational times: GCNN and MGN on a Nvidia A100 Tensor Core GPU (neural network training), MMGP on a 48 cores Intel Xeon Gold 6342 CPU (Gaussian process regressors training). Between parenthesis are indicated the numbers of trainings carried-out to optimize hyperparameters (best is **bold**).

Dataset	GCNN	MGN	MMGP
`Rotor37`	(200 ×) 24 h	(6 ×) 13 h 14 min	(10 ×) **2 min 49 s**
`Tensile2d`	(200 ×) 1 h 25 min	(6 ×) 6 h 50 min	(10 ×) **1 min 38 s**
`AirfRANS`	(200 ×) 5 h 15 min	(6 ×) 5 h 00 min	(10 ×) **5 min 47 s**

Numerical experiments

Example of MMGP predictions (AirfRANS dataset)

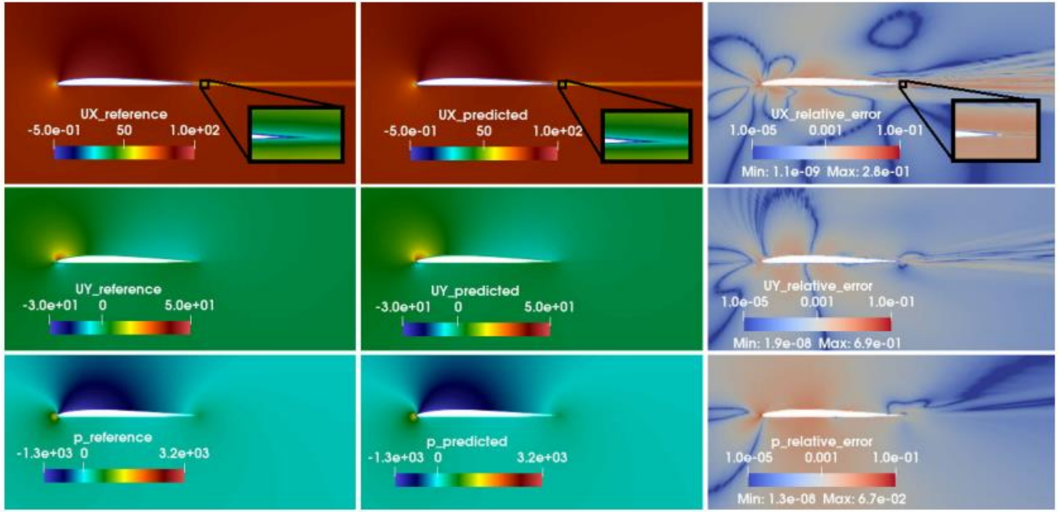


Figure 3: (`AirfRANS`) Test sample 787, fields of interest u (UX), v (UY) and p: (left) reference, (middle) MMGP prediction, (right) relative error.

Table 5: (`AirfRANS`) Relative errors (Spearman's rank correlation) for the predicted drag coefficient C_D (ρ_D) and lift coefficient C_L (ρ_D) for the four models of [14, Table 19], as well as GCNN, MGN and MMGP. These scalars of interest are computed as a postprocessing of the predicted fields (best is **bold**).

Model	Relative error C_D	Relative error C_L	Spearman's correlation ρ_D	Spearman's correlation ρ_L
MLP	6.2e+0 (9e-1)	2.1e-1 (3e-2)	0.25 (9e-2)	0.9932 (2e-3)
GraphSAGE	7.4e+0 (1e+0)	1.5e-1 (3e-2)	0.19 (7e-2)	0.9964 (7e-4)
PointNet	1.7e+1 (1e+0)	2.0e-1 (3e-2)	0.07 (6e-2)	0.9919 (2e-3)
Graph U-Net	1.3e+1 (9e-1)	1.7e-1 (2e-2)	0.09 (5e-2)	0.9949 (1e-3)
GCNN	3.6e+0 (7e-1)	2.5e-1 (4e-2)	0.002 (2e-1)	0.9773 (4e-3)
MGN	3.3e+0 (6e-1)	2.6e-1 (8e-2)	0.04 (3e-1)	0.9761 (5e-3)
MMGP	**7.6e-1** (4e-4)	**2.8e-2** (4e-5)	**0.71** (1e-4)	**0.9992** (2e-6)

(AirfRANS) PICPs of 93.05% and 93.5% for the outputs CL and CD

Conclusion and outlooks

Unlike deep graph neural networks, MMGP

- involves many **deterministic steps** (fewer things to learn)
- involves **low dimensional and easy to train** models
- can **handle large meshes**
- provides built-in **predictive uncertainties**

Limitations

- Morphing not in real time (required in inference)
- Does not use physics

Outlooks

- **Optimize morphing** for PCA compression