ReadMe: Reduced-Rank Online Gaussian Process Modeling With Uncertain Inputs

This ReadMe presents how to install and execute the code of the algorithm RRONIG, presented in the paper "Reduced-Rank Online Gaussian Process Modeling With Uncertain Inputs". The code will be available on GitHub after it is published. We implemented the code in C++ and Matlab, we provide here the C++ code given that the capabilities of the C++ code are higher, as it can divide the space in several sub-domains with one model for each, and both implementations give the exact same results up to a precision of 10^{-6}.

1 Installing the RRONIG toolbox

The code provided is in C++ (version C++ 14). The library Eigen is required. In order to run the main provided as an example, the library Boost is also necessary. We also provide two files, one with hyperparameters and the other with data (measures), called test_1.csv. Moreover, a result file for RRONIG that is the file returned after running the main.cpp with the two example files is available.

2 Initialization

To create a new GP model, the function in GPPrediction must be called:

GPExample = GPPrediction(nbFrequsForBasisOfEigenfunctions, dimCubes, pos0, sigmaLinSquare, sigmaSESquare, lSESquare, sigmaNoiseSquareModel);

- *nbFrequsForBasisOfEigenfunctions* (type: VectorDimInputI) is the number of eigenfunctions used for the model along each dimension of the input (sine functions with different frequencies).
- *dimCubes* (type: VectorDimInputd) is the size of the domains studied. Generally, one cube is enough but when the space studied is very large, especially for the localization algorithm that explores a whole building, the space is divided in several cubes in order to have an accurate modeling of all the small surfaces that compose it, with a relatively low number of eigenfunctions.
- *pos0* (type: VectorDimInputd) contains the coordinates of the chosen origin.

- *sigmaLinSquare* (type: double) should most of the time be set to 0. It enables the Gaussian process to converge more easily to a non-zero mean, in case we know that we want to approximate a function with a mean far from 0 but do not know its exact value.
- *sigmaSESquare* (type: double) is the initial variance chosen for the exponential-quadratic modeling of the initial output variance.
- *lSESquare* (type: double) is the characteristic lengthscale for the exponential-quadratic modeling of the initial output variance.
- *sigmaNoiseSquareModel* (type: double) corresponds to the variance of the output noise.

VectorDimInputI is a vector of size *dimInput* and is composed of integers. VectorDimInputd is also a vector of size *dimInput* and is composed of doubles. See GPTypes.h for their definition in Eigen.

As an example, if the inputs studied are in the interval [-5,5] and the outputs are of a zero-mean and one-dimensional, one can set *nbFrequsForBasisOfEigenfunctions* = 21, *dimCubes* = 5*1.2, *pos0* = 0 (the center) and *sigmaLinSquare* = 0. *sigmaSESquare*, *lSESquare* and *sigmaNoiseSquareModel* should be learned using a maximum likelihood optimization algorithm, such as the one used in NIGP [1].

3 Update: learning from new measures

In order to update *GPExample*, the measures should be given to the model GPPrediction. A new posCovValueStruct, *newData*, can be built easily by setting its different parameters:

posCovValueStruct newData(inputValue, varianceInput, outputValue);

newData is of type posCovValueStruct, which is a structure containing three elements:

- *inputValue* (or "position" as called in the structure names, type: VectorDimInputd): the input value where we suppose that the measure was taken (but there is an uncertainty about its real value, modeled by *varianceInput*).
- *varianceInput* (or "covPos", type: MatrixDimInputd): the variance of the input.
- *outputValue* (or "value", type: VectorDimOutputd): the value measured of the output (affected by the output noise of constant variance *sigmaNoiseSquareModel*).

VectorDimOutputd is a vector of size dimOutput and is composed of doubles. MatrixDimInputd is a square matrix of size dimInput filled with doubles.

To encapsulate the measure in order to update the model *GPExample*, a vector containing the measure should be created:

posCovValueVect batch;
batch.push_back(newData);
GPExample.update(batch);

The update modifies the values of the vector *learningPrediction* (refered to as $\mathbf{\Gamma}_l$ in the paper) and the matrix *learningCovar* (Υ_l, see the paper for more details on their expressions). They contain all the necessary information to predict the value of the field in any part of the model studied.

4 Predictions from the model

To get the prediction of the model at *xPrediction*, the function *predictOutput* should be called, with an empty vector to store the output mean prediction and an empty matrix for the associated output variance:

VectorDimOutputd predictedOutputValue;
MatrixDimOutputd predictedOutputVar;
bool prediction = GPExample.predictOutput (xPrediction, predictedOutputValue, predictedOutputVar);

prediction is a boolean that indicates if *xPrediction* is situated in the domain modeled by *GPExample* (it only verifies that *GPExample* was designed to cover this area, with an appropriate value of *dimCubes* in the initialization phase).

5 Notes

Here our examples are designed for 1D inputs and outputs but multi-dimensional inputs are possible as well: for D dimensions, in GPTypes, only this line should be changed:

const unsigned int dimInput = D;

For multi-dimensional outputs, if their different dimensions are not correlated (same hypothesis as for the previously existing methods), we advise to create one GPPrediction per dimension.

References

[1] A. Mchutchon and C. Rasmussen, "Gaussian process training with input noise," in *Advances in Neural Information Processing Systems*, vol. 24, 2011.