1 Find F

Set $(d,n) \in \mathbb{N}^*$ and $(x_i)_{i\in[\![0;n-1]\!]} \in \mathbb{R}^d$ distinct one another. We seek a polynomial $F \in \mathbb{K}[X]$ such that $\mathbb{K} \in \{\mathbb{R},\mathbb{C}\}$ and

$$\forall i \in [\![1;n]\!],\ F(x_i) = 0 \text{ and } \exists x_n :\ F(x_n) \neq 0$$

1.1 Case 1D

In the case $d = 1$, Vandermonde polynomials are a good start. The only thing to add to the construction is a point where the polynomial F is not null. Assuming the orthobarycentre of the family $(x_i)_{i\in[\![0;n-1]\!]}$ is not already inside, let's take the orthobarycentre of these points defined as $x_n := \frac{1}{n}\sum_{i=0}^{n-1} x_i$.
The system induced is this one :

$$\begin{pmatrix} 1 & x_0 & \dots & x_0^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & \dots & x_{n-1}^n \\ 1 & x_n & \dots & x_n^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \\ a_n \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}$$

that one can solve thanks to the inversion of the Vandermonde matrix, which is non-singular since all the points are distinct one another thanks to the assumption.

1.2 Case 2D

The work done here for $d = 2$ is meant to be easily generalizable to even greater values of d. Set $\forall i \in [\![0;n-1]\!], x_i$ has as coordinates $(x_{i,0}, x_{i,1})$.

1.2.1 Matrix method

Start of an idea (inspired from Vandermonde matrix in 1D)

Globally, F can be defined in $\mathbb{R}_{n+1}[X]$ considering its roots $(x_i)_{i\in[\![0;n-1]\!]}$ and the point x_n where it is not null. Therefore, set x_n the orthobarycentre with the same assumption as previously and $(\alpha_{ij})_{\{(i,j)\in\mathbb{N}:i+j\leq n\}} \in \mathbb{R}^{2n}$ such that

$$\forall (x,y) \in \mathbb{R}^2,\ F(x) = \sum_{\{(i,j)\in\mathbb{N}:i+j\leq n\}} \alpha_{ij} x^i y^j$$

$$\begin{pmatrix} 1 & x_{0,0} & x_{0,1} & x_{0,0}x_{0,1} & \dots & x_{0,1}^n \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1,0} & x_{n-1,1} & x_{n-1,0}x_{n-1,1} & \dots & x_{n-1,1}^n \\ 1 & x_{n,0} & x_{n,1} & x_{n,0}x_{n,1} & \dots & x_{n,1}^n \end{pmatrix} \begin{pmatrix} \alpha_{00} \\ \vdots \\ \alpha_{0(n-1)} \\ \alpha_{0n} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}$$

Actually, Vandermonde does not give any proof of the non-singularity of the left matrix and one could thought about putting more combinations of the coordinates of each point to extract a non-singular matrix after. Yet, finding this extracted matrix remains computationally heavy in addition to being uncertain.

In hindsight

The realm this idea steps into is the realm of principal component analysis. It remains interesting considering the fast differentiation of the polynomial one could get after. Nevertheless, it could force us to already use machine learning which will require some time to compute accurately and will always remain an approximation of the true solution we seek.

1.2.2 Direct approach

In this kind of situation, constructing the polynomial

$$F : (x, y) \mapsto \prod_{m=0}^{n-1} (x - x_i)(y - y_i)$$

quickly comes to mind. Nevertheless, F would be to often null on the domain and could induce great errors after. So, the idea is to replace the given factor by one null only at a given point as shown here:

Definition 1.1.

$$\forall (x, y) \in \mathbb{R}^2,\ F_r(x, y) = \prod_{k=0}^{n-1} ((x - x_k)^2 + (y - y_k)^2)$$

Definition 1.2.

$$\forall (x, y) \in \mathbb{R}^2,\ F_c(x, y) = \prod_{k=0}^{n-1} ((x - x_k) + i(y - y_k))$$

Remark (comparison).

- $\forall (x, y) \in \mathbb{R}^2,\ F_r(x, y) = |F_c(x, y)|^2$
- the coefficients of F_c are faster to compute but harder to evaluate than the ones of F_r

2 Hyperparameters

Here is the scheme of model tuning followed:

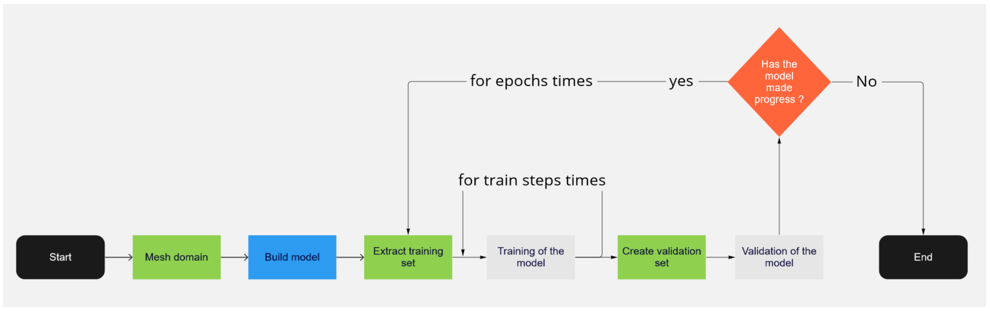

Figure 1: scheme of a model tuning

Remark. The red box involves using an early stopping callback to stop the epoch loop when the model begins to overfit. A `patience` integer set how far the system will go back in history. If during `patience` epochs the has not made progress the epoch loop is stopped.

The hypertuning comes when one wants to construct a model and a training that will end to a low validation error. Let's list all the hyperparameters involved in this scheme applied for the Poisson equation on a square domain:

- `grid_length`: integer setting the number of points in the domain to `grid_lengthxgrid_length`
- `l_units`: list of integers depicting the number of hidden layers and the number of neurons per layer of the sequential machine learning model
- `l_activations`: list depicting the activation functions of each layer of the model
- `noise`: integer in $\{0, 1\}$ conditioning the use of a Gaussian layer of mean 0 after the input layer
- `stddev`: the standard deviation of the Gaussian layer when it is used
- `optimizer`: string depicting the optimizer used
- `learning_rate`: float setting the learning rate of the previous optimizer
- `epochs_max`: integer setting the number of maximum epoch loops
- `n_trains`: integer setting the number of train loops
- `batch_size`: integer setting the number of points extracted from the mesh to construct the training set at each epoch loop

- `patience`: integer setting how many epoch loops the system can go on without progressing, after that the trial is ended

An automatization of the hypertuning by hand and using Keras Tuner has already been led. Nevertheless, some work must still be done to assure a low validation error with minimal cost of time and hardware. A few bugs, impeding a proper convergence, also seems to persist in some of our implementations.