POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

HOMEWORK REPORT

Enigma: Mathematical Modeling and Simulating the Cardiac Function

SCIENTIFIC COMPUTING TOOLS FOR ADVANCED MATHEMATICAL MODELLING

Authors: Alessandro Anichini, Giovanni Maria Bonvini and Davide Galbiati

Academic year: 2023-2024

Brief abstract

Electro-anatomical mapping is a technique that allows to capture both electrical activity (intracardiac electrograms) and anatomical data in order to display 2D (3D) fields of quantities of medical importance. In particular, 'electro' refers to electrical data the system, in this case, collects from intracardiac electrograms, via a catheter. These signals help in assessing the activation sequence of the heart. The word 'anatomical' refers to the 3-dimensional, in our case simply 2-dimensional, location information that provides of the complex geometry of the organ and the real time location of the probe (1 mm accuracy). 'Mapping' finally refers to recording and displaying such collected fields.
The mapping catheter record the chamber's electrical activity at different locations reached by the clinician moving the catheter (changing the curvature and controlling the depth) in the chamber. Then, each sensor registers the local electrical activity as an electrogram recording. Finally, a bipolar electrogram recording results from the difference of unipolar signals acquired by two pairs of electrodes on the same branch (measure the electrical activity in between). Reconstructing the time activation of such electromagnetic propagation is possible measuring the time interval between ECG reference and local activity peak-evolution. Hence, the wavefront propagation is displayed via isochrones and a voltage map propagating in the heart tissue.

1. Checkpoint 1: Reconstruction of activation time and approximation of conduction velocity

In the first task we developed a data-driven numerical strategy to reconstruct the activation time and the conduction velocity field from the sparse measurements given by a mapping catheter.
The first checkpoint requires the development of an algorithm to interpolate/extrapolate the activation time field from timings recorded in the 20 electrodes of the mapping catheter. The available data consists of 100 recordings, which include the location of the measurements and the recorded timings [s].

1.1. Mathematical formulation of the problem

This task is mainly data-driven and does not require a sophisticated mathematical formulation. The algorithm should output the a reconstruction of the activation times evaluated at each point of a structured grid discretized in (1501,1501) points on

$$\Omega = (-1.5, 1.5) \times (-1.5, 1.5)\,[\text{cm}^2] \tag{1}$$

To recover the conduction velocity [cm/s] it sufficies to apply the theorem of the inverse of the derivative. Let $T : R^2 \to R$ be the time activation $T(x, y)$ and $X : R \to R^2$ be the inverse of T such that $(x, y) = X(t)$. From the theorem of the inverse, we have:

$$X'(t) = \frac{1}{\nabla T(x,y)}$$

It's worth noticing that such field is a vectorial field defined on Ω . For our purpose, we evaulated its magnitude as:

$$v(t) = \frac{1}{(\nabla_x + \nabla_y)^{1/2}}$$

1.2. Methods

Our dataset was composed of 100 recordings of 20 samples of the time activation field associated to their coordinates (x, y). To implement such reconstruction we used the Gaussian process methodology.
In probability theory and statistics, a Gaussian process is a stochastic process (a collection of random variables defined on a bidimensional space (x,y)), such that every finite collection of those random variables has a multivariate normal distribution. The distribution of a Gaussian process is the joint distribution of all those (infinitely many) random variables, and as such, it is a distribution over functions of space.
Despite being a very complicated stochastic process it can be modelled and equivalently represented with its covariance which takes the name of kernel. There are many Kernel that could be taken into cosideration, for example Matern and RBF. The key parameter of each kernel is the length scale which is indeed a measure of "how close" two points have to be to influence each other significantly.
We trained the Gaussian process on the 100 recordings using hyperparameter tuning of a Gaussian Process (GP) regression model using cross-validation. The primary goal is to find the optimal kernel and its parameters to minimize the mean squared error (MSE) of the model. K-cross validation performs the training for each of the k parts, thereby avoiding issues of overfitting, but also avoiding asymmetrical sampling (and therefore biased) of the observed sample, typical of splitting the data into only two parts (i.e., training/validation). In other words, the observed sample is divided into groups of equal size, one group is iteratively excluded at a time, and an attempt is made to predict it with the non-excluded groups, in order to verify the quality of the predictive model used. The selection of the sub-samples is done as:

```
n_split = 10
KF = KFold(n_splits=n_split,shuffle=True)
```

As for the choice of the lenght scale parameter we tuned on an interval of 20 choices in (0.1, 2). As for kernels we selected Matern and RBF tuned on the aforementioned lenghtscales as

```
for il, l in enumerate(ls_values):
    kernels = [Matern(length_scale=l), RBF(length_scale=l)]
```

The Matérn kernel (with smoothness of default 1.5) is given by:

$$k_{3/2}(d) = \sigma^2 \left(1 + \frac{\sqrt{3}d}{\ell}\right) \exp\left(-\frac{\sqrt{3}d}{\ell}\right)$$

where:
- $d = \|\mathbf{x} - \mathbf{x}'\|$ is the Euclidean distance between two points of the time sample $\mathbf{x}$ and $\mathbf{x}'$,
- σ^2 is the variance parameter,
- ℓ is the length scale parameter.

The RBF kernel is given by:

$$k_{\text{RBF}}(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right)$$

where:
- σ^2 is the variance parameter,
- ℓ is the length scale parameter,
- $\|\mathbf{x} - \mathbf{x}'\|$ denotes the Euclidean distance between points $\mathbf{x}$ and $\mathbf{x}'$.

For the training we iterate over the chosen training partions storing each prediction

```
for n in range(n_split):
    for i in idx[n][0]:
        x_meas = CP1data[i][0]
        y_meas = CP1data[i][1]
```

```
            t_meas = CP1data[i][2]

            x = np.array([x_meas, y_meas]).transpose()
            y = np.array(t_meas)
            model = GaussianProcessRegressor(kernel=k)
            model.fit(x, y)
            train_models.append(model)
```

For the validation set every time field in the validation array is computed as the mean of every trained-predictor gaussian models and MSE between real time field and predicted one

```
    err_kpart = []
    for j in idx[n][1]:
        x_meas = CP1data[j][0]
        y_meas = CP1data[j][1]
        t_meas = CP1data[j][2]
        coord = np.vstack((x_meas.flatten(), y_meas.flatten())).T
        ...
        means = [m.predict(coord) for m in train_models]
        t_hat = np.mean(np.array(means), axis=0)
        err_kpart.append(mean_squared_error(t_meas, t_hat))
    err_k.append(np.mean(err_kpart))
```

Finally for every validation set we append the mean square error evaluated on every element of each validation fold. Then, we evaluate the mean of the error for the whole validation sets and find the best choice of Kernel and ls.

```
    err_k.append(np.mean(err_kpart))
    err[il,ik] = np.mean(err_k)
    best_idx = np.argwhere(err==np.min(err))[0]
    best_ls = ls_values[best_idx[0]]
    if best_idx[1] == 0:
        best_kernel = Matern(length_scale=best_ls)
    else:
        best_kernel = RBF(length_scale=best_ls)
```

After this passages have been performed, the model is ready to be used to predict the whole field on Ω

```
    model = GaussianProcessRegressor(kernel=Matern(length_scale=1.2))
    model.fit(x,y)

    # coordinates of the grid
    coord = np.vstack((X.flatten(), Y.flatten())).T

    time_pred = model.predict(coord).reshape(X.shape)
```

First the model is fit on the new data. The parameters of the kernel have been determined at training time. Hence, covariance function is estimated on the input feature and partial time output. Thanks to covariance function we can make a suitable prediction averaging on all the possible prediction measures (function) entailed by the distribution probability (of functions), i.e. the Gaussian process itself.
For the second part, we had to estimate (cfr. mathematical formulation) the conduction velocity. Theorically, if the time recostruction of the time field is good enough the prediction of velocity field should not be an issue. In reality, we had bumped into some overestimation problems. As a matter of fact, when the field is recostructed near the starting point, its propagation is far more rapid that in the other regions of the domain. This does not allow to recostruct the time gradient properly with the usual methods of python such as (`np.gradient`). At first, some abrupt rescaling using the maximum value of the field and an empirical threshold has been performed, with the result of effectively lowering the maximum, but the side effect has been a global and undesired rescaling of all the points.

```
    threshold = 110
    max_found = np.max(vel_pred)
    scaling_factor = threshold / max_found
    vel_pred = vel_pred * scaling_factor
```

Hence, we decided to go for another gaussian process. Every time that a time field is predicted, it's velocity recostruction is calculated in the easiest way:

```
    vel_pred = 1.0/( np.sqrt((grad_AT[0]/(3.0/1500))**2+(grad_AT[1]/(3.0/1500))**2) )
```

Then we stored five random samples of such field choosing 50 velocity points and their coordinates.
Over these new samples, five gaussian processes are fitted to estimated a new recostructed field, withe the hope that the new interpolation could smooth the field. In the end, a mean of such field is performed and returned as result. The following snippet of code describes this:

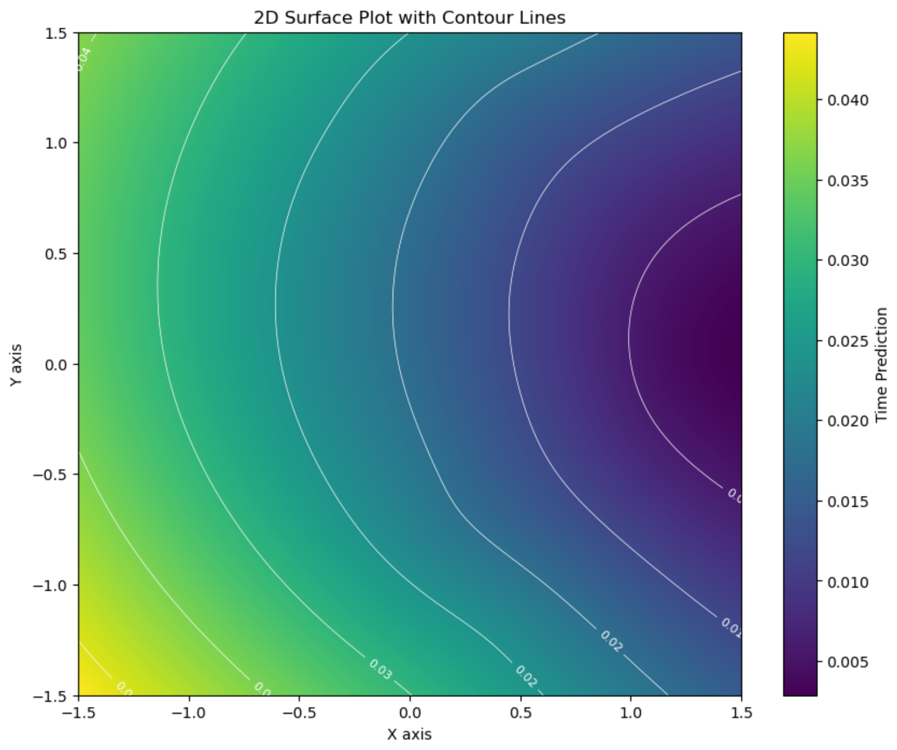
```
train_models = []
    for i in range(5):
            x_meas = samples[i][0]
            y_meas = samples[i][1]
            vel_pred_prior = samples[i][2]

            x = np.array([x_meas,y_meas]).transpose()
            y = np.array(vel_pred_prior)
            model = GaussianProcessRegressor(kernel=best_kernel)
            model.fit(x,vel_pred_prior)
            train_models.append(model)
    # prediction of the mean of every GP on the full grid
    means_GPs = [m.predict(coord) for m in train_models]
    # estimate in the mean over all the GPs
    vel_pred = np.mean(means_GPs,axis=0).reshape(X.shape)
```

It's worth noticing that the training has been done sampling from all prediction times, 50 velocity samples, in order to identify kernel and lenght scale which revealed to be the same as before (Matern, 1.2).

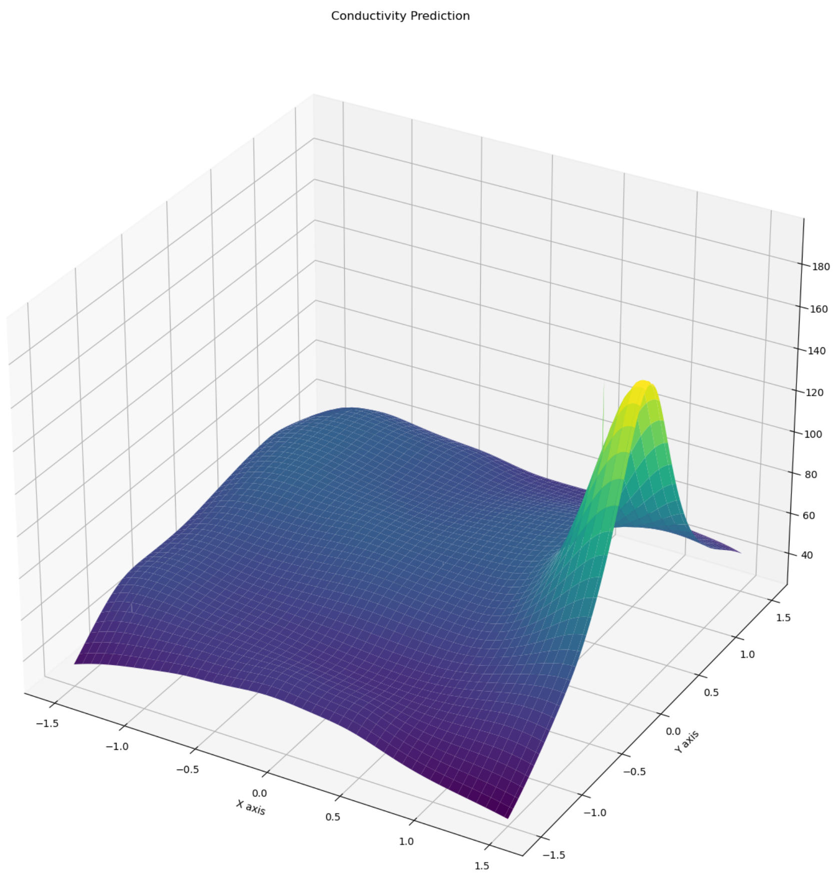
1.3. Numerical results

Numerical results have been pretty satisfying, especially for the first task where we got an accuracy of 0.96/1. Below, the resulted temporal field.



Time field

As for velocity, unfortunately the recostruction is slightly less accurate and it's not very robust on the sampling methodology and on the predicted time field. Maybe, some more improvements could have been done, like uniformly sampling, thus making homogeneous spacing.



Velocity field

Report your numerical results (e.g. final parameters estimate or computed biomarkers)

2. Checkpoint 2: Parameter estimation problem

The goal of Checkpoint 2 is to find the 3 parameters *fiber angle*, *anisotropy ratio* and *starting point* of a given patient for which we are given 20 time samples t in 20 space cooridnates (x,y). We are given a dataset of 5 patients with 20 time samples each in the usual spatial domain $(-1.5, 1.5)x(-1.5, 1.5)$ and the corresponding 3 parameters for each one of them. The parameters are bounded in the following intervals:

- *fiber angle* $\in [-\pi/10, \pi/10]$
- *anisotropy ration* $\in [1, 9]$
- *starting point* $\in [-1.5, 1.5]$

For simplicity, we will now call the 3 parameters μ_1, μ_2, μ_3.

2.1. Mathematical formulation of the problem

The mathematical formulation of the problem is the following:
Minimize over $\mu = (\mu_1, \mu_2, \mu_3)$ the functional $C(\mu)$, defined as

$$C(\mu) = \sum_{i=1}^{20} |t_i - F(x_i, \mu)|^2 \tag{2}$$

where t_i is the i-th measurement of the chosen patient, $F(x_i, \mu)$ is the forward map from three parameters μ to the time measurements t_i and x_i is the location (x_i, y_i) of the i-th time measurement. So the optimal μ_{opt} will be given by:

$$\mu_{opt} = \text{argmin}_{(\mu_1,\mu_2,\mu_3)}\, C(\mu) \tag{3}$$

The domain of minimization is the following:

- $\mu_1 \in [-\frac{\pi}{10}, \frac{\pi}{10}]$
- $\mu_2 \in [1, 9]$
- $\mu_3 \in [-1.5, 1.5]$

2.2. Methods

To find the 3 parameters for a given unknown patient we went through the following procedure:

1. Build a surrogate model for the Eikonal solver *anysotropic_FMM*
2. Build a cost functional to minimize over the parameter space
3. Perform a minimization of the cost functional with a grid search over the discretized parameter space

2.2.1 Surrogate model

We inted to build a surrogate model for the *anysotropic_FMM* that takes as input: the 3 parameters (*fiber angle*, *anisotropy ratio*, *starting point*), (x,y) and provides as output the times t in the corresponding space coordinates.
To do so, we took advantage of a big dataset that was created by another team and then shared. The dataset is a map that takes as input a tuple (μ_1, μ_2, μ_3) and provides as output the corresponding time field reconstructed with the *anysotropic_FMM*. The map contains 6800 uniformly sampled tuples in the parameter space.
The **surrogate model** that we decided to adopt consists in a Neural Network that takes as input: the 3 parameters (μ_1, μ_2, μ_3), the spatial coordinates (x_i, y_i) and provides as output the corresponding single time t_i. The choice of this was driven by the fact that we have no need to predict the time field in the whole domain but only on the 20 given points (x_i, y_i). This allows us to speed up training and prediction, as well as gain accuracy and reduce computational effort.
The NN ha the following characteristics:

- 3 layers with $64, 128, 64$ neurons
- The training set was generated by taking 100 random values for (*theta_fiber*,*a_ratio*, *y0*) and 100 random locations (x,y) for each value of (*theta_fiber*,*a_ratio*, *y0*) in the dataset, so a total of 10000 datas.
- Final loss function (mse) of 3×10^{-7} at last epoch.
- Training took only about 30 seconds, since training dataset is relatively small.

We here show the evolution of the loss function for our Neural Network.

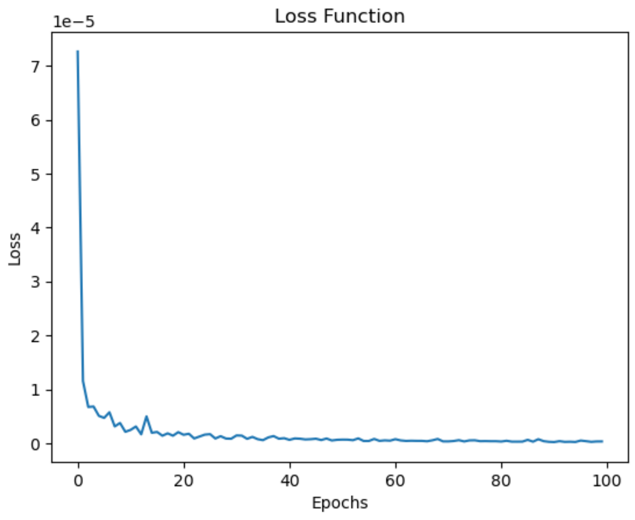
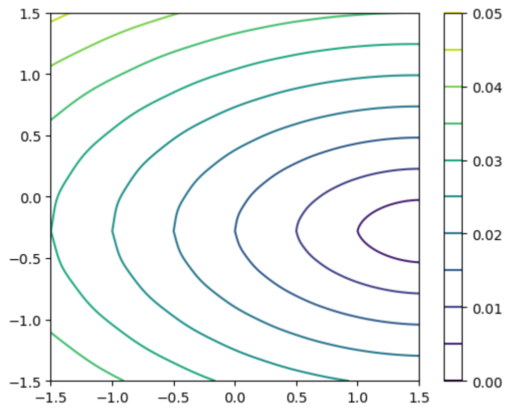
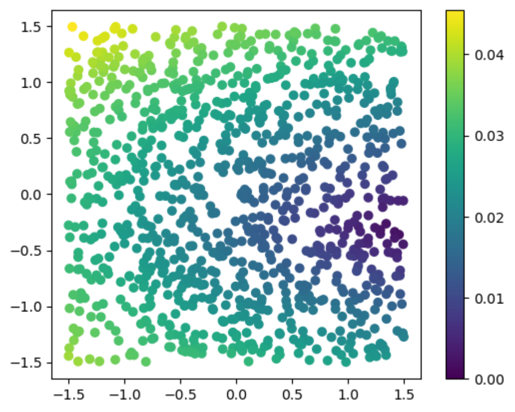


Figure 1: Loss function evolution over epochs

We here show the prediction of the NN performed for 1000 randomly chosen spatial points next to the actual solution of the Eikonal equation that the NN is trying to predict. For this validation we randomly chose $\mu_1 = 0.020$, $\mu_2 = 3.878$ and $\mu_3 = -0.277$.



Eikonal solution



Neural Network prediction

2.2.2 Cost functional

The cost functional that we intend to minimize is $C(\mu)$, defined in Equation 2. The cost functional takes as input the 3 parameters, the 20 samples (x,y,t) and evaluates the error between the 20 time samples $\{t_i\}_{i=1}^{20}$ and the estimated ones with the NN $\{\hat{t}_i\}_{i=1}^{20}$ in the 20 positions $\{(x_i, y_i)\}_{i=1}^{20}$. The error function defined in the following way implies that the NN needs to perform 20 evaluation for each selected tuple (μ_1, μ_2, μ_3) and therefore for each evaluation of the cost functional itself. This will have a significant impact in the efficiency of the method, nontheless we consider this a suitable efficiency-accuracy trade-off.

2.2.3 Optimization

For the optimization procedure we opted for a grid search over the parameter space. This choice allows us to have control over the trade-off between accuracy and efficiency, as well as a great efficiency overall thanks to the parallelization of the grid search itself.
The grid search was performed in the following way:

- A first grid search is performed by uniformly discretizing each parameter axis with 10 values each.
- A second grid search is then performed around the minimum found in the first one. We discretize uniformly the intervals of length $1/10$ of the whole axis with 7 values for each axis.
- The final minimum found is the optimal solution.

This procedure requires a total of $10^3 + 7^3 = 1343$ iterations and proved to be extremely accurate.

3. Checkpoint 3: Parametric speed field estimation problem

The goal of this Checkpoint is to find the value of the speed field $c(x, y)$ of a patient starting from a given set of 20 time samples in the space domain.

3.1. Dataset

We are given data for 100 different patients and for each one we have:

- 20 sampled measurements of time in the usual spatial domain 3x3 centered in (0,0)
- An estimate for the speed field $c(x, y)$ defined on the whole domain

It is important to highlight that each patient has the same set of parameters μ_1, μ_2 and μ_3, which are unknown, but a different speed field $c(x, y)$.

3.2. Mathematical formulation

We can subdivide the problem into 3 different tasks:

1. Find the 3 parameters μ_1, μ_2 and μ_3. The formulation is the one provided in **Checkpoint 2**.
2. Find a compressed representation of the speed field c with a reduced number of parameters c_i: $c(x, y) = \sum_{i=1}^{n} c_i b_i$, with $\{b_i\}$ being a suitable set of basis functions and n parameters.
3. Finally find the correct combination of c_i that represents the speed field of the unknown given patient.

This is done by finding a solution to the following problem:

$$c_{opt}(x,y) = \text{argmin}_{c_i, i=1,..,n} \sum_{i=1}^{20} |T(x_i, y_i, c; \mu) - t_i|^2 \tag{4}$$

where T is the activation time field reconstructed with the Eikonal equation, $\mu = (\mu_1, \mu_2, \mu_3)$ previously found and (x_i, y_i, t_i) is the sampled data. The activation time is found by solving the following problem for T:

$$\begin{cases} c(x,y) \cdot \sqrt{(\nabla T)^\top D \nabla T} = 1 & \text{in } \Omega, \\ T(x_0, y_0) = 0. \end{cases}$$

3.3. Methods

3.3.1 Parameter estimation

In order to estimate μ_1, μ_2 and μ_3 we exploited our solution to **Checkpoint 2**. The issue is that our surrogate model for CP2 was meant to be applied to healthy patient that have a constant speed field $c(x,y) = 100$ $\forall (x,y) \in \Omega$. For this reason we decided to choose a restricted set of those 100 patient that satisfied the following condition: the speed field $c(x,y)$ can have a value lower than 85 for at most 5% of the domain Ω. This restriction ensures that the patient to which we apply the model of CP2 have a speed field that is not too far from the one of a healty patient.
We are left to estimate μ_1, μ_2 and μ_3 from the following 20 patients: [4, 14, 16, 19, 22, 26, 29, 41, 46, 50, 55, 56, 61, 64, 71, 84, 87, 89, 91, 97].

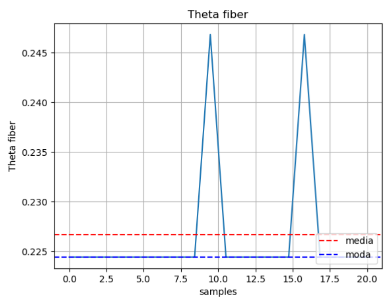


Figure 2: μ_1

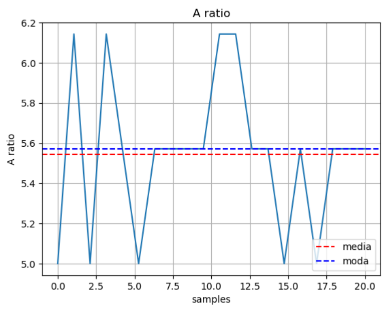


Figure 3: μ_2

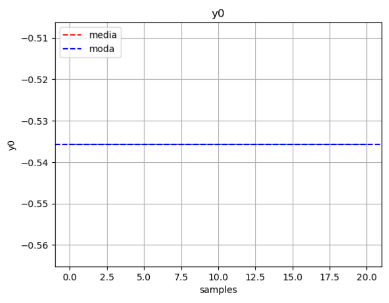
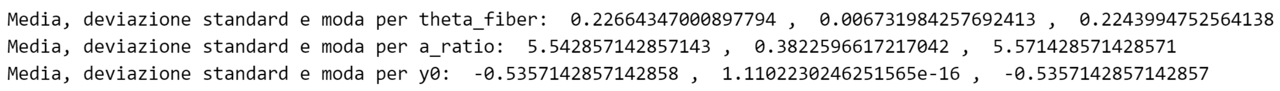


Figure 4: μ_3



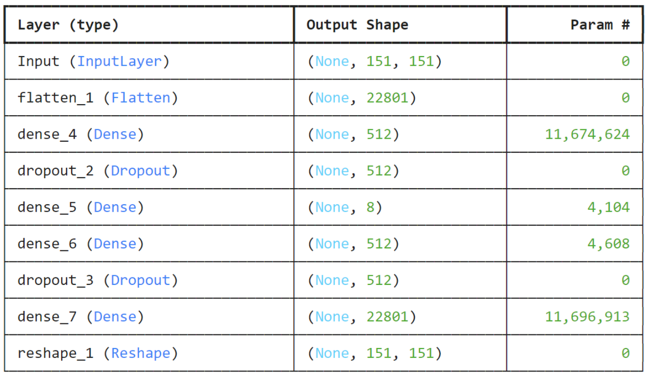
```
Media, deviazione standard e moda per theta_fiber:  0.22664347000897794 ,  0.006731984257692413 ,  0.2243994752564138
Media, deviazione standard e moda per a_ratio:  5.542857142857143 ,  0.3822596617217042 ,  5.571428571428571
Media, deviazione standard e moda per y0:  -0.5357142857142858 ,  1.1102230246251565e-16 ,  -0.5357142857142857
```

Figure 5: Parameter estimation results

So we conclude by choosing the mean value between the selected patiens, thus ending with the following estimate: $\mu_1 = 0.227$, $\mu_2 = 5.543$, $\mu_3 = -0.536$

3.3.2 Compressed representation for $c(x,y)$

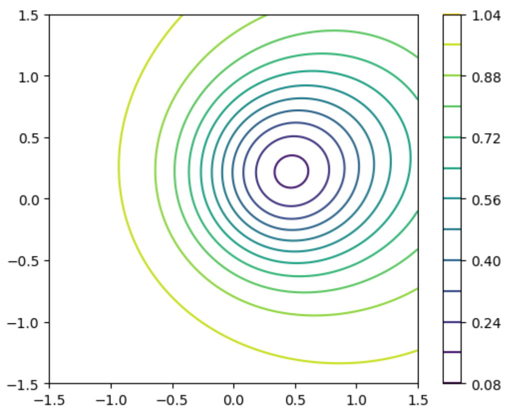
For this task we adopted a Neural Network autoencoder that compress the 151x151 original speed field into an 8-dimensional latent space. We here show the NN architecture that was adopted.



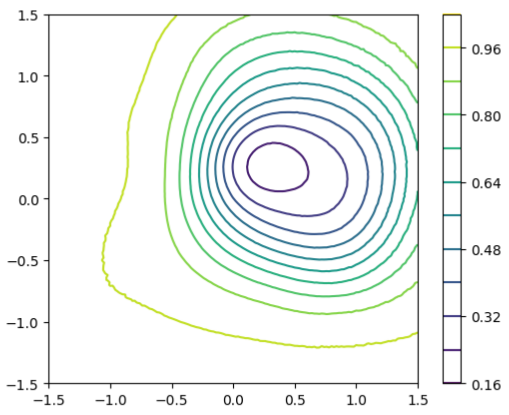
Layer (type)	Output Shape	Param #
Input (InputLayer)	(None, 151, 151)	0
flatten_1 (Flatten)	(None, 22801)	0
dense_4 (Dense)	(None, 512)	11,674,624
dropout_2 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 8)	4,104
dense_6 (Dense)	(None, 512)	4,608
dropout_3 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 22801)	11,696,913
reshape_1 (Reshape)	(None, 151, 151)	0

Figure 6: Autoencoder architecture

The encoding procedure shows some reconstruction error. This error cannot be avoided due to the low dimensionality of the latent space with respect to the initial field. Nontheless, we observe that the main features and orders of magnitude of the field are well recontructed and therefore we accept the representation.

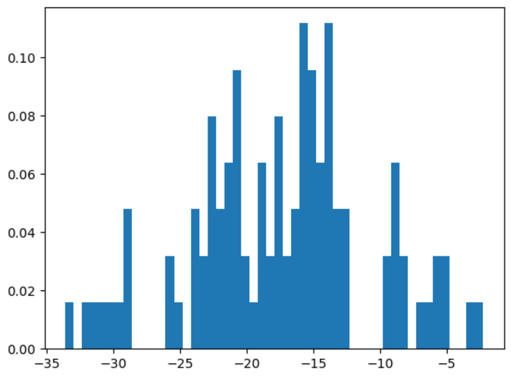


Original field

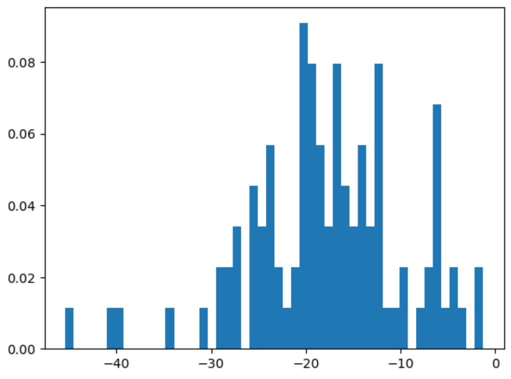


Decoded field

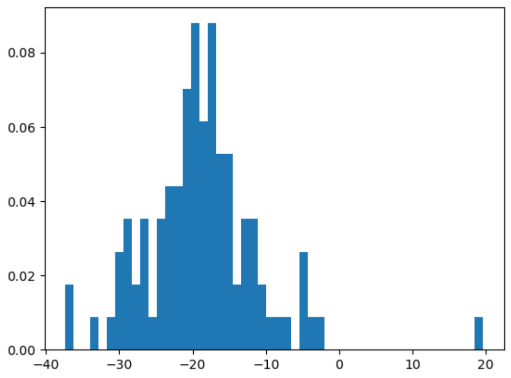
We are then able to plot the distributions of the 8 parameters for the 100 given speed fields:

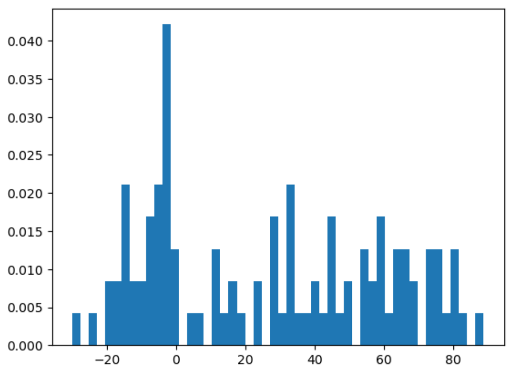


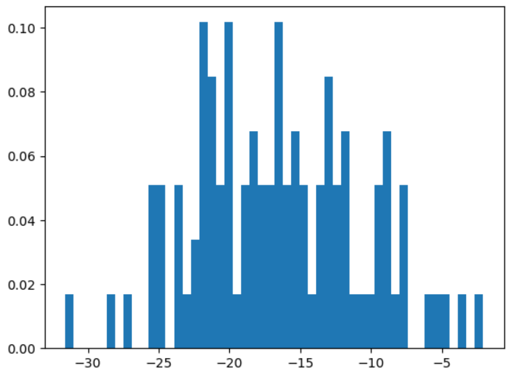
Parameter 1

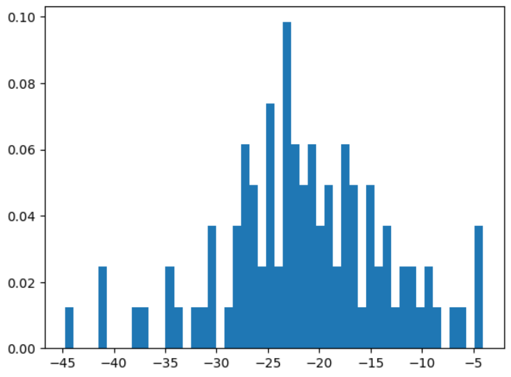


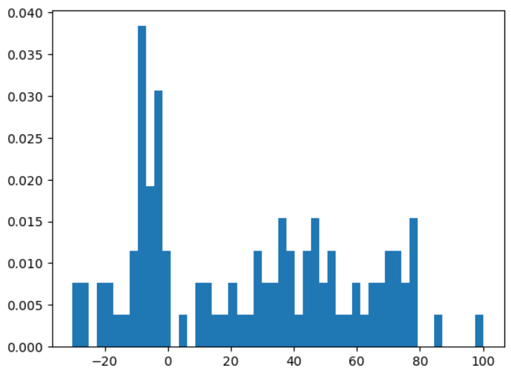
Parameter 2

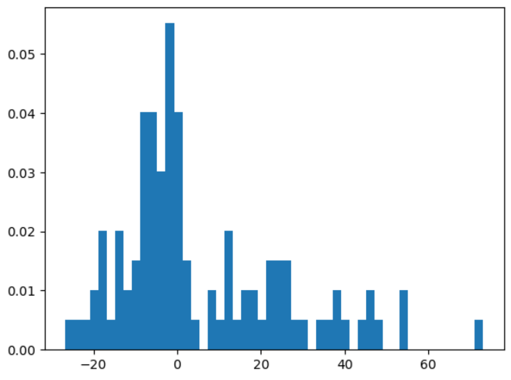

Parameter 3


Parameter 4


Parameter 5


Parameter 6


Parameter 7


Parameter 8

3.3.3 Optimization

The optimization routine would require us to minimize the cost functional previously defined. This turned out to be a quite difficult task due to the high variability of the parameters in the latent space and the relatively high number of parameters to optimize.
The solution that we propose is to sample the parameter space according to the distributions provided above and then perform a grid search over this sampled parameters. To speed up computation we compute the activation time T_c with a surrogate model for the Eikonal equation which is actually the same *FMM Eikonal solver* just scaled down to a lower refinement of $51x51$, rather than the original $151x151$. The time required for a single computation of the downscaled FMM Eikonal is around 4 sec, showing a speedup of 600-700%. If we choose, for example, to select 5 samples for each parameters we end up with a 5^8 speed field samples, which is also the

number of times that we need to compute the FMM Eikonal. This results in 434hrs of computation, which is obviously not feasible.
For this reason we opted for a different approach. We minimize over a set of n speed field that are generated by choosing the set of 8 parameters according to a randon sampling over their distribution. So basically the sampling is performed as in the grid search case, but instead of having n^8 speed field we only have n.
This ends up with a rather approximate solution since the parameter space is explored in a rather arbitrary way.

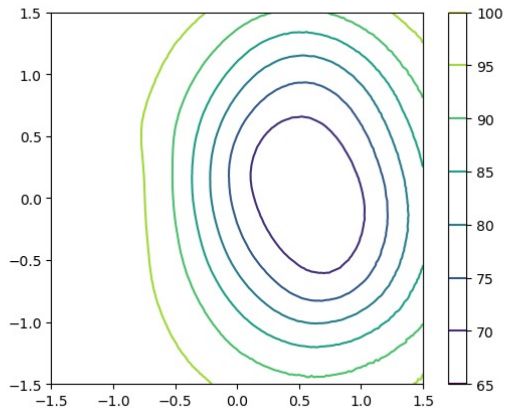


Figure 7: Original speed field

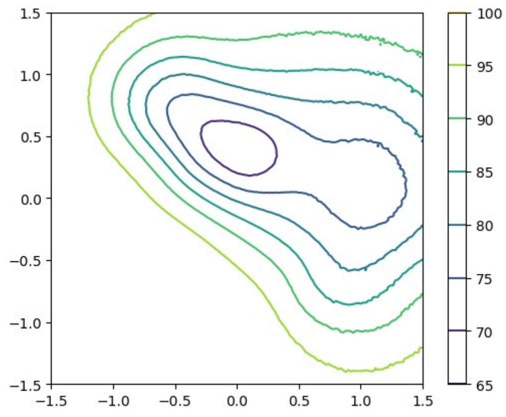


Figure 8: Reconstructed speed field

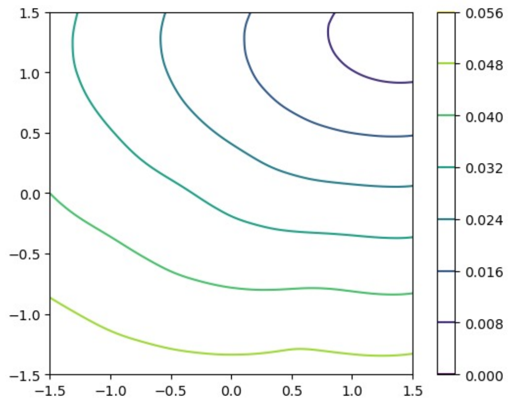


Figure 9: Activation time with original speed field

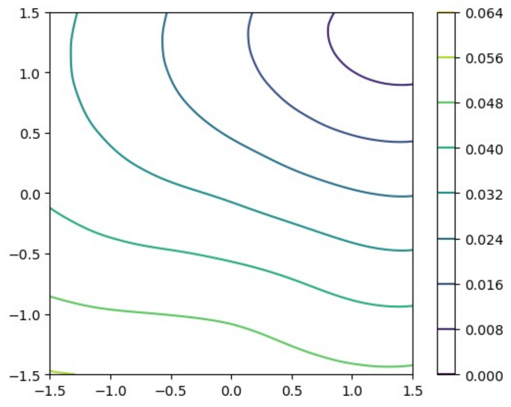


Figure 10: Activation time with reconstructed speed field

We can see in Figure 7 and 8 that the reconstructed speed field is rather approximate. Nonetheless, we want to highlight that the activation times solved with the Eikonal using the original speed field (Figure 9) and the reconstructed one (Figure 10) are very similar. This turns out to be a further issue for the minimization since the cost function may be minimized for a speed field that is far from the actual solution.

3.4. U-net

A different solution that we investigated is a straithforward image-to-image translation though U-net.
U-net is a deep learning architecture born in 2015 for image segmentation. It is based on many convolutional layers that form an U: it is divided into two parts, the encoder (downsample part) and the decoder (upsample part).

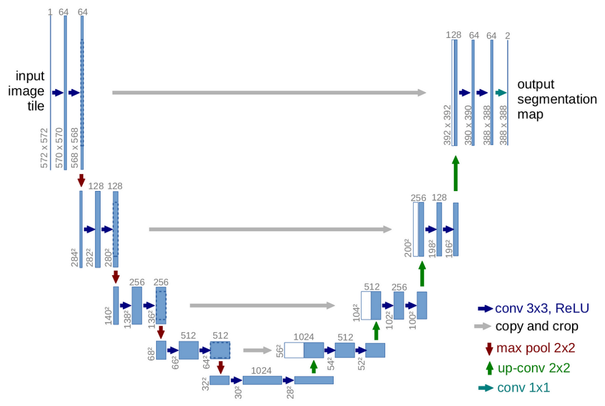


Figure 11: Plot of the Unet architecture.

We used it taking as input the reconstructed interpolation of the time field computed as in checkpoint 1 starting from the 20 time measurement for every patient and training it using as output the correspondent speed field. We treated both the time field and the speed field as a 151x151 image and we trained the network.
This is a training reconstruction of the speed field compared with the ground truth:

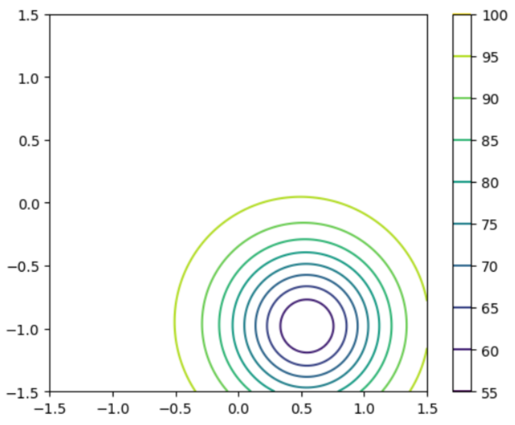


Figure 12: True speed field of a training patient.

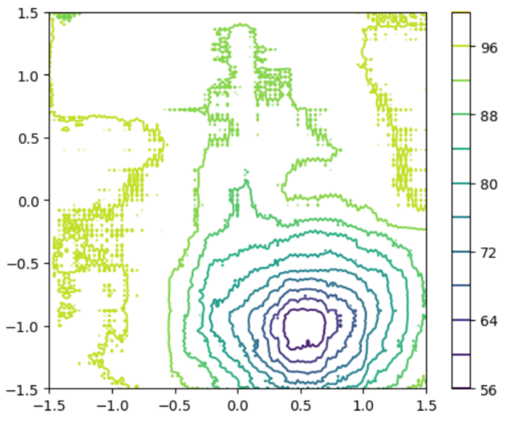


Figure 13: Reconstructed speed field of a training patient..

We got an accuracy of 1.51/2.