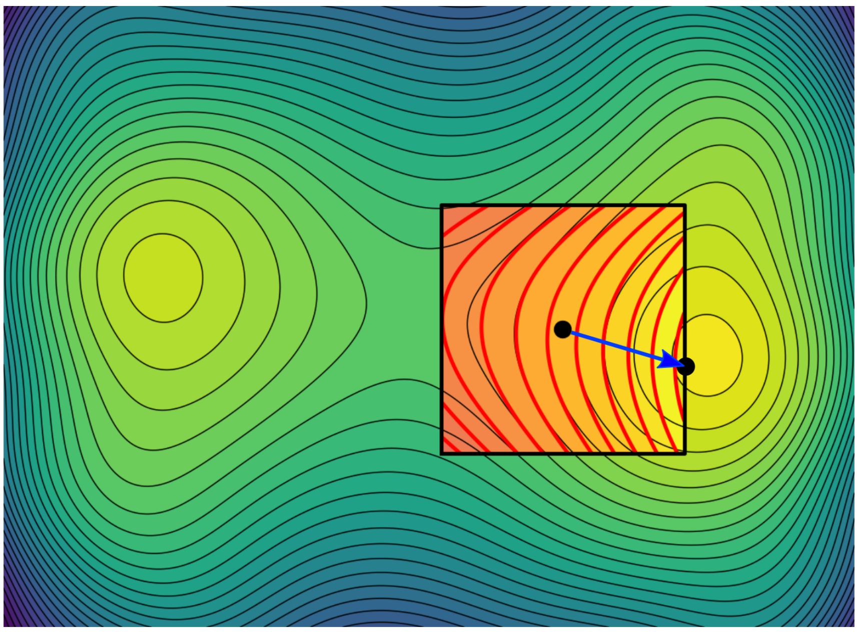
Internship report

Solving ill-posed inverse problems involving partial differential equations with neural networks

Mattéo Clémot

Supervisors :
Elisa Riccietti (ENS de Lyon, LIP, Team OCKHAM)
Stefania Bellavia (University of Florence)

October – December 2022



Contents

Introduction

Context

Partial Differential Equations (PDEs) are used in a wide variety of problems in different domains like physics, biology or engineering. They can be numerically solved with several classical and well-studied methods, like finite differences or finite elements. Recent works have proposed to use neural networks to approximate solutions of PDEs. It led to the emergence in 2017 of *Physics Informed Neural Networks* (PINNs), which use the knowledge of the underlying law of physics of a system to reduce the size of the domain of admissible solutions. Some results similar to other state-of-the-art methods for solving PDEs have been obtained with this framework. In addition, it gives considerable advantages in high dimensional problems or problems with difficult geometries.

Objective

PINNs have mostly been studied for solving problems that are well-posed. Ill-posed problems, that are problems whose solutions are unstable under data perturbation, are generally harder to solve. For instance, given the following PDE:

$$\Delta u(x) + c(x)u(x) = f(x),$$

we would like to recover the functional parameter $c(x)$ from the function f and noisy measurements of the solution u. The aim of this internship is to solve such ill-posed inverse problems, using regularizing training strategies. Indeed, due to the ill-posedness of the problem, we expect classical training methods not to give a satisfying solution.

Plan

Section 1 introduces PINNs and how they are trained to approximate the solution of a PDE. It also shows some results concerning several PDEs.

Section 2 introduces inverse PDE problems and the issues with their ill-posedness.

Section 3 presents the architecture used to solve such problems and some regularization techniques to deal with their ill-posedness.

Section 4.1 is related to a regularizing optimization method for inverse problems: the trust-region Levenberg-Marquardt method.

Section 4.2 is related to a regularizing optimization method for a constrained formulation of PDE inverse problems, known as Sequential Linear Programming.

Implementation

All the code for the experiments was written in Python, with PyTorch.

1 Physics Informed Neural Networks

Physics Informed Neural Networks (PINNs) were introduced in [RPK19], aiming to solve numerically partial differential equations. The main idea is to optimize a neural network to represent the solution, with a loss function taking into account both the measurements and the physics described by the PDE.

1.1 Framework

Let $\mathcal{N}$ be a (possibly nonlinear) differential operator, Ω a subset of $\mathbb{R}^d$ and $\partial\Omega$ its frontier. We want to find the solution $u^* : \mathbb{R}^d \to \mathbb{R}$ to the following problem:

$$\begin{cases} \mathcal{N}[u] = 0 & \text{on } \Omega \\ u = \psi & \text{on } \partial\Omega \end{cases} . \tag{1}$$

For instance, in this work we will be particularly interested in the elliptical problem corresponding to

$$\mathcal{N}[u] = -a\Delta u + cu - \varphi$$

where $a \in \mathbb{R}$ is a constant, Δ is the Laplace operator, and $c, \varphi : \mathbb{R}^d \to \mathbb{R}$ functional parameters. However, it is also possible to consider more complicated EDPs that can be nonlinear (see Subsection 1.3).

1.2 Method

We consider a neural network with parameters θ, $u_\theta : \mathbb{R}^d \to \mathbb{R}$, taking as input coordinates in $\mathbb{R}^d$, that we will optimize to make it approximate the solution u^*. In other words, we want to find the parameter values θ^* such that $\|u_{\theta^*} - u^*\|$ is as small as possible.

Architecture The considered architecture is the classical multi-layer perceptron (MLP), with dense layers, as illustrated in Figure 1. Concerning the activation function, instead of making the classical choice of ReLU, tanh or softplus, we choose a periodic function, namely sine. This is justified by [SMB+20], which argues that such an activation function permits to improve both convergence and stability of the optimization.

Loss To solve Problem 1, we can formulate the network optimization problem as the sum of a fitting term and a term penalizing the solutions that do not respect the EDP:

$$\min_\theta \int_\Omega \mathcal{N}[u_\theta]^2 + \int_{\partial\Omega} (u_\theta - \psi)^2$$

In practice, these losses will be only approximated. We consider the following point sets, which in practice will be chosen as a d-dimensional grid over $\Omega = [0,1]^d$:

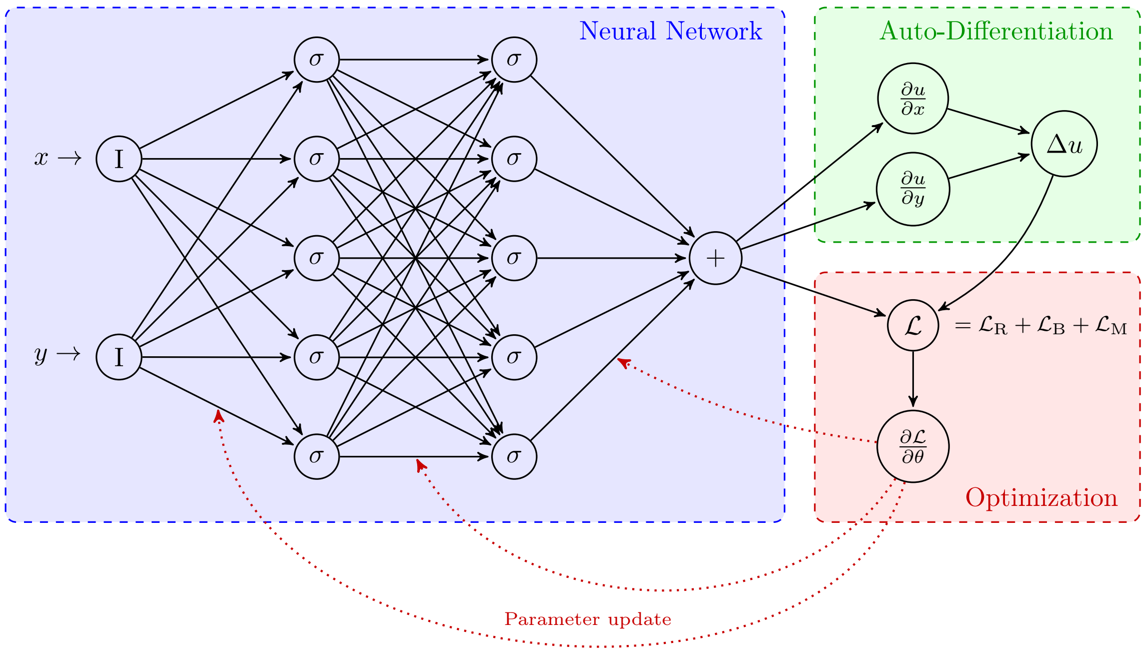
- $\mathbf{x}_R \in \Omega^{n_R}$ are the points where we compute the PDE residual ;
- $\mathbf{x}_B \in (\partial\Omega)^{n_B}$ are the points where we will test the value of u on the boundary ;
- $\mathbf{x}_M \in \Omega^{n_M}$ are the points where we are given a measure of u, as a "clue", if provided. We write $\bar{u}$ the vector of these measures.

We then define the following terms of the loss:

$$\mathcal{L}_\text{R} = \frac{1}{n_R}\|\mathcal{N}[u_\theta](\mathbf{x}_R)\|^2 \tag{2}$$

$$\mathcal{L}_\text{B} = \frac{1}{n_B}\|u_\theta(\mathbf{x}_B) - \psi(\mathbf{x}_B)\|^2 \tag{3}$$

$$\mathcal{L}_\text{M} = \frac{1}{n_M}\|u_\theta(\mathbf{x}_M) - \bar{u}\|^2 \tag{4}$$



Neural Network
Auto-Differentiation
Optimization
$x \rightarrow$
$y \rightarrow$
I
I
σ
$+$
$\frac{\partial u}{\partial x}$
$\frac{\partial u}{\partial y}$
Δu
$\mathcal{L}$
$= \mathcal{L}_\mathrm{R} + \mathcal{L}_\mathrm{B} + \mathcal{L}_\mathrm{M}$
$\frac{\partial \mathcal{L}}{\partial \theta}$
Parameter update

Figure 1: Example of neural network architecture for a 2-dimensional PDE direct problem, with 2 hidden layers of 5 neurons, and dense layers. In our work, we took $\sigma = \sin$ as the activation function.

The optimization problem is thus rewritten:

$$\min_\theta \lambda_\mathrm{R}\mathcal{L}_\mathrm{R} + \lambda_\mathrm{B}\mathcal{L}_\mathrm{B} \underbrace{+\lambda_\mathrm{M}\mathcal{L}_\mathrm{M}}_{\substack{\text{if measures}\\\text{provided}}}$$

where $\lambda_\mathrm{R}, \lambda_\mathrm{B}, \lambda_\mathrm{M} > 0$ are coefficients.

In practice, the derivatives (second-order in the case of the elliptical problem) in $\mathcal{N}[u_\theta](\mathbf{x}_R)$ are computed thanks to auto-differentiation (with respect to the inputs of the network). The overall method is summarized in Figure 1.

Optimization We mainly use and compare two optimization algorithms : Adam and L-BFGS, with are first-order descent direction algorithms (the second one approximates the hessian with first order informations). Both gives satisfying results.

1.3 Examples

Figures 2, 3, 4 show resolutions of several temporal, one-dimensional, classical PDEs:

- heat equation:

$$\frac{\partial T}{\partial t} + \frac{\partial^2 T}{\partial x^2} = 0$$

- inviscid Burgers' equation:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0$$

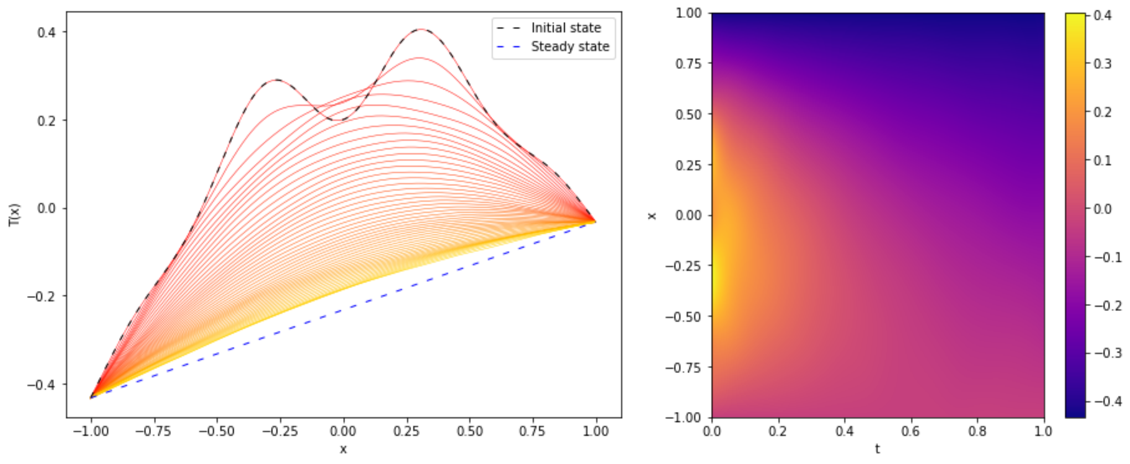


Figure 2: Resolution of the 1D heat equation $\frac{\partial T}{\partial t} + \frac{\partial^2 T}{\partial x^2} = 0$, from $t = 0$ (red) to $t = 1$ (yellow).

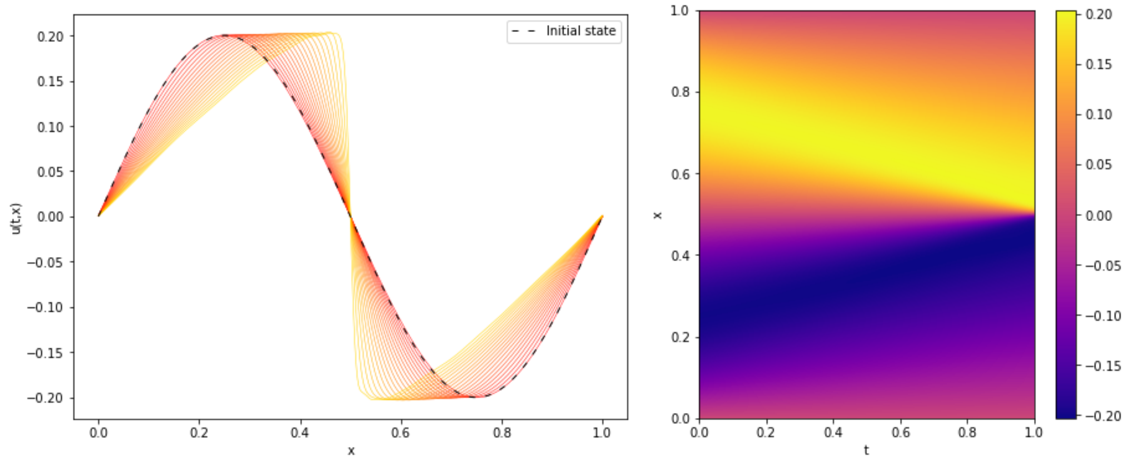


Figure 3: Resolution of the 1D inviscid Burgers' equation $\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0$, from $t = 0$ (red) to $t = 1$ (yellow).

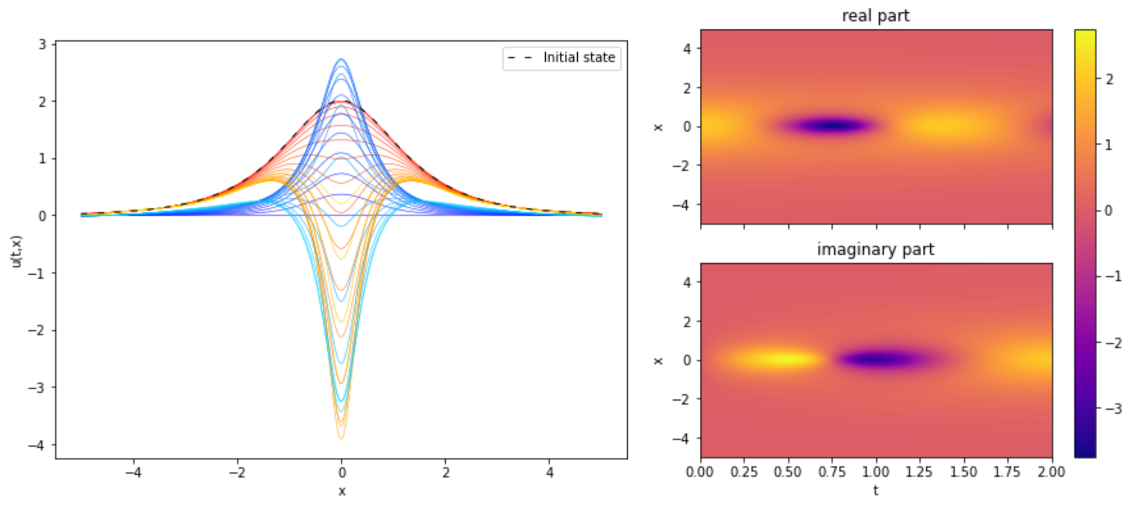


Figure 4: Resolution of the 1D non-linear Schrödinger equation $i\frac{\partial u}{\partial t} + \frac{1}{2}\frac{\partial^2 u}{\partial x^2} + |u|^2 u = 0$, from $t = 0$ (red for $\Re(u)$, dark blue for $\Im(u)$) to $t = 1$ (yellow for $\Re(u)$, light blue for $\Im(u)$). Initial conditions from [RBPK17].

For this equation and a sinusoidal initial condition, the solution tends to a discontinuity, which is hard to represent with a neural network as its output is $\mathcal{C}^\infty$.

- non-linear Schrödinger equation:

$$i\frac{\partial u}{\partial t} + \frac{1}{2}\frac{\partial^2 u}{\partial x^2} + |u|^2 u = 0$$

This equation is for complex-valued functions, so the network used to solve it had two output instead of only one. The initial and bounding conditions were those of the dataset of [RBPK17], and the optimization process was able to reproduce the complex behaviour of the solution. Note that, as expected, the error compared to the ground truth (from the same dataset) tends to increase with t.

2 Inverse PDE problems

We now consider the PDE inverse problem : given (noisy) measures $\bar{u}$ from the solution u, we would like to find a functional parameter, for instance c in the above elliptical problem.

2.1 Ill-posedness of inverse problems

A well-posed problem in the sense of Hadamard verifies the three following properties:

- existence: a solution exists;
- unicity: the solution is unique;
- stability: the solution changes continuously with the parameters of the problem.

Inverse problems are typically ill-posed, often because of the third condition not being verified. This makes them harder to solve, since a small change in the input may change radically the computed solution. In particular, noise in the input is particularly difficult to deal with in inverse problems.

We present below two examples of inverse problems involving PDEs.

1D benchmark problem We take $\Omega = [0, 1]$ and set the following constants and functions (see Figure 5):

$$\begin{cases} c(x) = \sqrt{2}\cos(2\pi x) \\ u(x) = \cos(2\pi x) + 2 \quad \text{(we have Neumann boundary conditions } u'_{|\partial\Omega} = 0) \\ \varphi = -u'' + cu \end{cases}$$

We aim to recover c from the knowledge of u (which is possibly given noisy) and φ. We can consider this problem is ill-posed in the sense that the following application, which recovers c from an input u

$$\begin{array}{ccc} \mathcal{C}^2(\Omega, \mathbb{R}^*) & \to & \mathcal{C}^2(\Omega, \mathbb{R}) \\ u & \mapsto & \frac{\varphi + u''}{u} \end{array}$$

is not continuous relative to the $\|\cdot\|_\infty$ norm. Figure 5 illustrates the ill-posedness of this problem by plotting a function u and lightly noisy variants, with their corresponding solutions c which differ distinctly.

Remark. *It is continuous relative to the "$W^{2,\infty}$ Sobolev space" (with norm $\|u\|_{W^{2,\infty}} = \max\{\|u\|_\infty + \|u'\|_\infty + \|u''\|_\infty\}$), but we are not supposed to have the knowledge of u'' (nor u') so this hypothesis doesn't seem to be reasonable.*

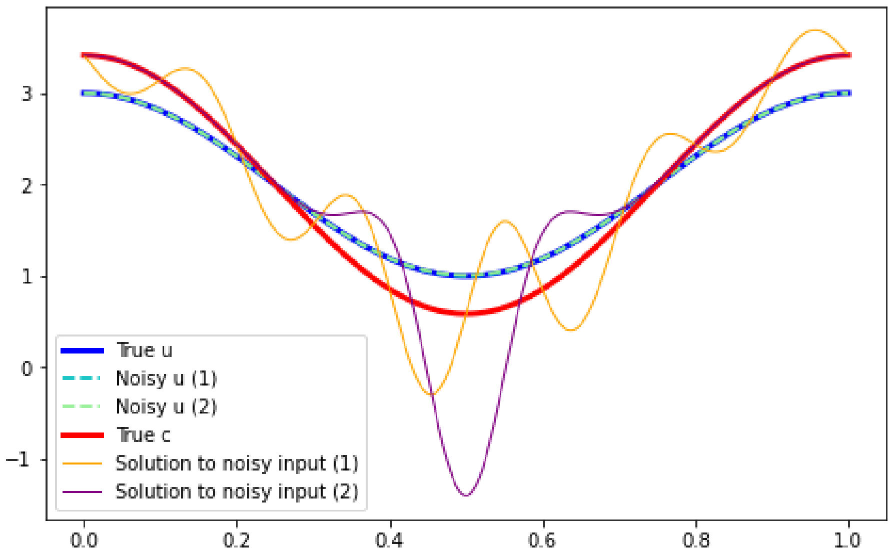


Figure 5: Ill-posedness of the 1D benchmark inverse problem. The first noisy input has as noise term $10^{-3}\sin(10\pi x)$, the second one has as noise term $10^{-2}\exp(-100(x-1/2)^2)$. These noise terms are not even visible on this graph.

2D benchmark problem We take $\Omega = [0,1]^2$ and set the following functions (see Figure 7):

$$\begin{cases} c(x,y) = 1.5\sin(2\pi x)\sin(3\pi y) + 3\left(\left(x-\frac{1}{2}\right)^2 + \left(y-\frac{1}{2}\right)^2\right) \\ u(x,y) = 16x(1-x)y(y-1)+1 \quad \text{(we have Dirichlet boundary conditions } u_{|\partial\Omega} = 1) \\ \varphi = -\Delta u + cu \end{cases}$$

2.2 Method without regularization

Let's consider again PINNs to solve these problems. More precisely, we now consider a neural network (see Figure 6, left)

$$\begin{array}{rcl} \Phi_\theta : \mathbb{R}^d & \to & \mathbb{R}^2 \\ \mathbf{x} & \mapsto & (u_\theta(\mathbf{x}), c_\theta(\mathbf{x})) \end{array}$$

that we will try to optimize so that u_θ fits the measures $\bar{u}$ and that the PDE is verified (i.e. the residual $\mathcal{N}[u_\theta, c_\theta]$ is as small as possible). We suppose we can access the following data :

- $\mathbf{x}_R \in \Omega^{n_R}$ are the points where we compute the PDE residual ;
- $\mathbf{x}_B \in (\partial\Omega)^{n_B}$ are the points where we will test the value of u on the boundary ;
- $\mathbf{x}_M \in \Omega^{n_M}$ are the points where we are given a (possibly noisy) measure of u. We write $\bar{u}$ the vector of these measures, and $\bar{\varphi}$ the values of φ on those points.

$$\mathcal{L}_\mathrm{R} = \frac{1}{n_R}\|\mathcal{N}[u_\theta, c_\theta](\mathbf{x}_R)\|^2 \tag{5}$$

$$\mathcal{L}_\mathrm{B} = \frac{1}{n_B}\|u_\theta(\mathbf{x}_B) - \psi(\mathbf{x}_B)\|^2 \tag{6}$$

$$\mathcal{L}_\mathrm{M} = \frac{1}{n_M}\|u_\theta(\mathbf{x}_M) - \bar{u}\|^2 \tag{7}$$

Then the optimization process is similar to the previous Section. We minimize the following loss function:

$$\min_\theta \lambda_\mathrm{R}\mathcal{L}_\mathrm{R} + \lambda_\mathrm{B}\mathcal{L}_\mathrm{B} + \lambda_\mathrm{M}\mathcal{L}_\mathrm{M}.$$

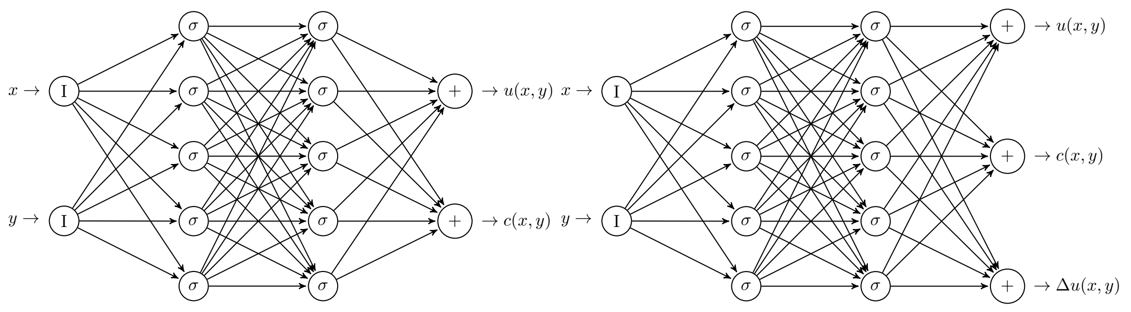


Figure 6: **Left:** Neural network architecture for the 2D elliptical inverse problem, with 2 hidden layers of 5 neurons. **Right:** 3-output neural network architecture for the 2D elliptical inverse problem, with 2 hidden layers of 5 neurons. We have a third output that we want to give an estimation of the Laplacian.

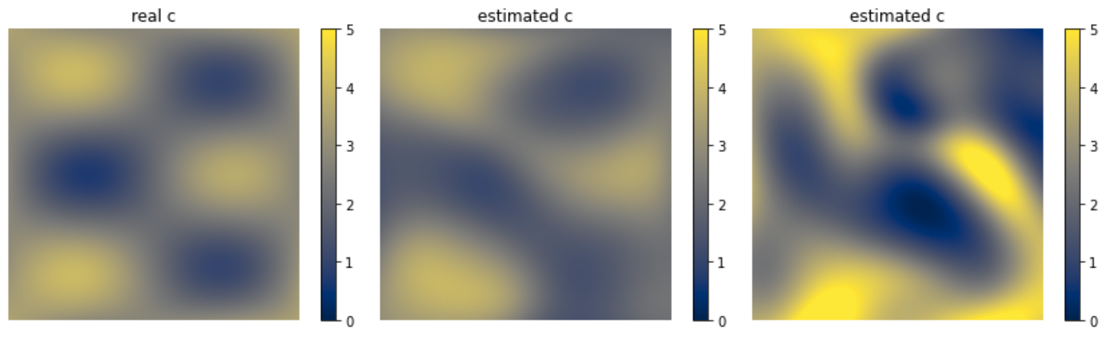


Figure 7: **Left:** ground truth c parameter in the 2D elliptical problem from above. **Center:** estimated c with noiseless measures $\bar{u}$ and ADAM optimizer. The MSE is 0.15. **Right:** estimated c with a gaussian noise ($\sigma = 2 \times 10^{-2}$) on the measures $\bar{u}$ and ADAM optimizer. The MSE is 1.26.

See Figures 7 and 8 for the results of this method on the to test problems from above. This method seems able to deal with such inverse problem when there is no noise on the measures of the solution. However, as soon as the measures are noisy, we do not achieve retrieving a good estimation of c. In the next Sections, we have a look at different regularization method to deal with this problem.

2.3 Alternative architecture: three-output network

In the experiments, we observed that while u is well-approximated, Δu presents a high error. Thus, being the error on Δu the most critical, we explore the possibility to use a three-output network (see Figure 6, right)

$$\begin{array}{rcl} \Phi_\theta : \mathbb{R}^d & \rightarrow & \mathbb{R}^3 \\ \mathbf{x} & \mapsto & (u_\theta(\mathbf{x}), c_\theta(\mathbf{x}), \ell_\theta(\mathbf{x})) \end{array}$$

such that $\Psi_\theta(\mathbf{x}) \simeq (u(\mathbf{x}), c(\mathbf{x}), \Delta u(\mathbf{x}))$. Then, instead of using the Laplacian computed thanks to auto-differentiation in the PDE residual, we use this third output. We also have to add a "consistency" loss function forcing the third output to match the Laplacian computed from the first output thanks to auto-differentiation. Thus, the loss function becomes

$$\min_\theta \lambda_{\mathrm{R}} \widetilde{\mathcal{L}_{\mathrm{R}}} + \lambda_{\mathrm{B}} \mathcal{L}_{\mathrm{B}} + \lambda_{\mathrm{M}} \mathcal{L}_{\mathrm{M}}.$$

with the following modified residual term:

$$\widetilde{\mathcal{L}_{\mathrm{R}}} = \frac{1}{n_R} \left(\|\ell_\theta(\mathbf{x}_R) - \Delta u_\theta(\mathbf{x}_R)\|^2 + \|\mathcal{N}[u_\theta, c_\theta, \ell_\theta](\mathbf{x}_R)\|^2 \right)$$

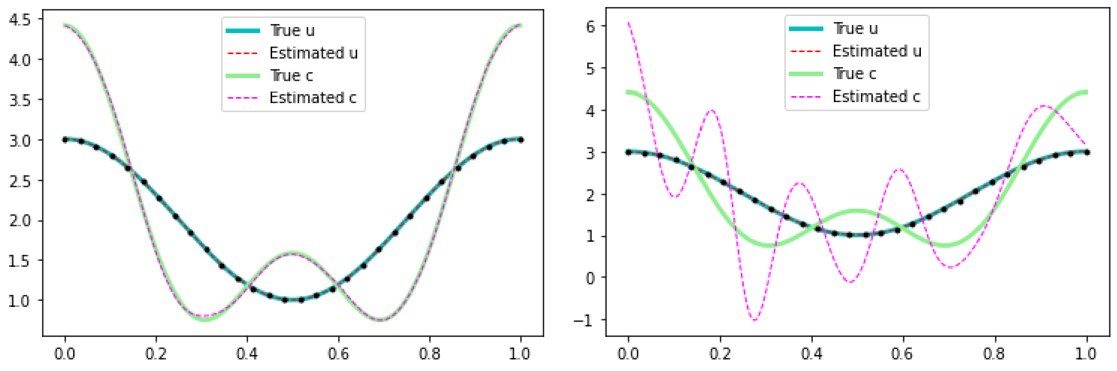


Figure 8: **Left:** Resolution of the 1D elliptical problem with no noise on the measures $\bar{u}$. **Right:** Resolution of the 1D elliptical problem with noise ($\sigma = 10^{-2}$) on the measures $\bar{u}$.

where, in the case of the elliptical inverse problem, $\mathcal{N}[u_\theta, c_\theta, \ell_\theta] = -\ell_\theta + c_\theta u_\theta - \varphi$.

Remark. *We scale the Laplacian output by a factor 100, to put all three outputs in a similar range.*

3 Regularizations for ill-posed inverse problems

We have seen above that our problem is ill-posed, in the sense that the solution may be highly sensitive to small changes in the input. In this section, we study regularization techniques to prevent obtaining a wrong output, corresponding to the solution for input overfitted to noise.

- Explicit regularization: it consists to add an explicit term to the loss (for instance penalizing the norm of the output like Tikhonov regularization);
- Implicit regularization: all other regularizations. For instance early stopping of the optimization process, the choice of the model, the use of stochastic optimization, the use of regularizing optimization...

3.1 Optimizing u and c together is regularizing

Because the error on u is much smaller that the error on c, we tried to use two separate networks Φ and Ψ, one to approximate u with the measures $\bar{u}$ (and the boundary conditions), the other to approximate c thanks to the PDE. We first optimize Φ

$$\arg\min_{\theta_\Phi} \frac{\lambda_M}{N^2} ||R_M(\theta_\Phi, x_M)||^2 + \frac{\lambda_B}{B^2} ||R_B(\theta_\Phi, x_B)||^2$$

so that $\Phi(\theta_\Phi, \cdot) \simeq u$. Then we optimize Ψ to approximate c by minimizing the PDE residual using this estimation of u:

$$\arg\min_{\theta_\Psi} \frac{1}{M^2} ||R_i(\theta_\Psi, x_R)||^2$$

where here $R_i(\theta, x_R) = -\Delta_{x_R}\Phi(\theta_\Phi, x_R) + \Psi(\theta_\Psi, x_R)\Phi(\theta_\Phi, x_R) - \varphi(x_R)$.

This method does not work. Indeed the first network approximates u with some error, that will give a wrong Δu. The second network, with the PDE residual as unique loss, will simply converge towards $\frac{\phi+\Delta u}{u}$ which is not the correct c due to this error.

Learning u and c together seems to act like a kind of implicit regularization, which achieves to (partially) cancel the error on Δu, and thus c.

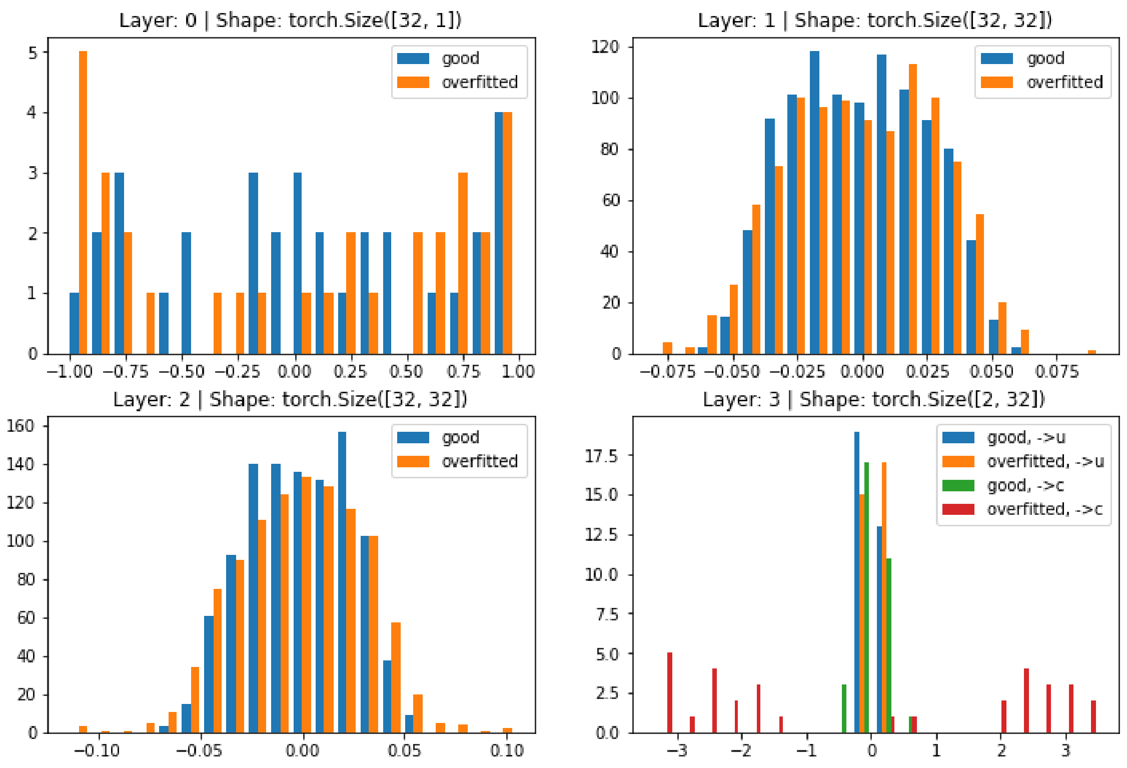


Figure 9: Histograms of the weights of the layers of two networks with 3 hidden layers of 32 neurons, one giving the right c, the other overfitted on noise, for the 1D elliptic inverse problem. Except for the last layer, the Tikhonov regularization on the weights of the network seems not extremely relevant.

3.2 Explicit regularizations

We tested the use of two different explicit regularization terms, both of Tikhonov type, which is one of the most frequent regularization technique and consists to modify the loss by adding a 2-norm term.

In the first one we penalize using the square 2-norm of c_θ: $\mathcal{L}_{\text{Tik}} = \|c_\theta(\mathbf{x}_R)\|^2$. We have also tested with the term $\|\nabla c_\theta(\mathbf{x}_R)\|^2$, with the idea that it may avoid oscillating outputs.

In the second one, we use the square 2-norm of the parameters θ. The analysis of the networks obtained with and without noise on the input have shown that hidden layers have a very similar distribution of weights, except for the last layer. More precisely, when the network has overfitted the noise on the input, the weights of the last layer towards the output for the c estimation have a wider distribution (see Figure 9). This a led us to try the following regularization term:

$$\mathcal{L}_{\text{Tik}'} = \|\theta_{\text{last layer}}\|^2.$$

Results None of these attempts have been really convincing. The main difficulty is to choose the coefficient in front of the term. With a too small coefficient, it will have no effect, while with a too big coefficient, it may worsen the estimation: for example, the $\|c_\theta\|^2$ term may make the c estimation wrongly close to 0, while $\|\nabla c_\theta\|^2$ term may make it wrongly almost constant. As for the norm of the parameters, it enabled to avoid oscillating estimation but not to obtain the right solution.

3.3 Early stopping

Another classical – implicit – regularization constists to stop early the optimization process, with the idea that it will avoid the overfitting of the noise in the input. Indeed when there is noise,

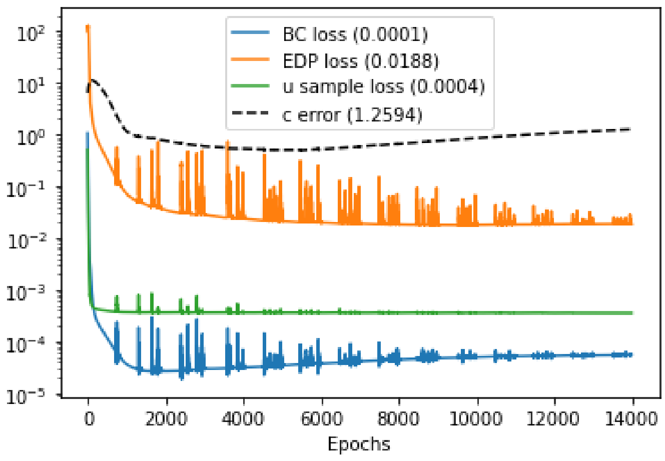


Figure 10: Too much iterations, leading to noise overfitting, in the 2D noisy elliptical inverse problem. The real error on the estimation of c (in dotted black) is decreasing until approximately 5500 iterations, and then is increasing again even if the loss function continues to decrease.

the expected behavior of the true error is to first decrease and then increase (see Figure 10). The difficulty is to find a reliable stopping criterion, which made us not investigate further this path.

4 Research directions: regularizing methods for inverse problems

We present below two research directions that are interesting for our problem but which have not yet given convincing results. Both are based on trust-region methods, which is a class of globally convergent iterative methods. They consist to take at each iteration a model (linear, quadratic...) of the objective function, and to minimize it in the current trust region (see Figure 11). Then, the step is accepted or rejected and the trust region updated according to the ratio between the actual reduction and the reduction predicted by the model (see Equations 9, 14).

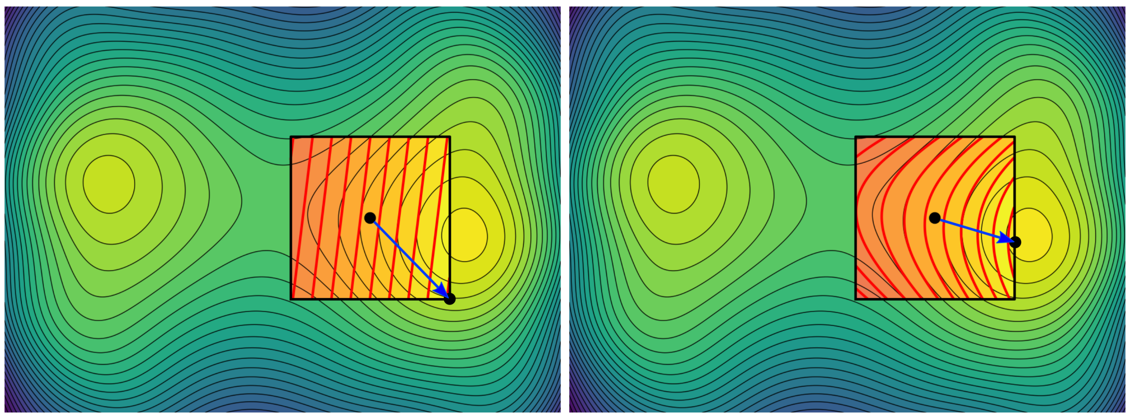


Figure 11: Illustration of trust region method steps, for a 2-variables optimization problem. The employed model is a linear model on the left and a quadratic model on the right. The trust region is an infinity-norm ball.

4.1 A regularizing method for the unconstrained formulation

In this subsection, we present a regularizing method for the same unconstrained formulation. This is a trust-region variation of the Levenberg-Marquardt method. This section leans notably on Sections 4.3 (trust-region methods) and 10.3 (Levenberg-Marquardt method) from [NW99], and on [BMR16].

4.1.1 Levenberg-Marquardt method

Nonlinear least-squares problems The methods we introduce and discuss in this section are intended to solve non-linear least-squares problems, i.e. given a nonlinear differentiable application $F : \mathbb{R}^n \to \mathbb{R}^m$:

$$\min_{\mathbf{x}} \frac{1}{2}\|F(\mathbf{x})\|^2.$$

Gauss-Newton method The commonest method to solve nonlinear least-squares problems is an iterative method known as Gauss-Newton. It consists, at each iteration, to approximate the objective function F by a linear model.

$$\min_{\mathbf{p}} \frac{1}{2}\|F(\mathbf{x}_k) + J(\mathbf{x}_k)\mathbf{p}\|^2.$$

The minimizer $\mathbf{p}_k^{GN}$ is then used to do the step and update $\mathbf{x}$:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k^{GN}.$$

Levenberg-Marquardt method The Levenberg-Marquardt method is a modification of the Gauss-Newton method, which consists to add an explicit regularization term depending on λ_k:

$$\min_{\mathbf{p}} \frac{1}{2}\|F(\mathbf{x}_k) + J(\mathbf{x}_k)\mathbf{p}\|^2 + \frac{\lambda_k}{2}\|\mathbf{p}\|^2.$$

Let $B_k = J(\mathbf{x}_k)^T J(\mathbf{x}_k)$ and $g_k = J(\mathbf{x}_k)^T F(\mathbf{x}_k)$. The minimizer $\mathbf{p}_k^{LM}(\lambda_k)$ of this minimization subproblem satisfies the equation:

$$(B_k + \lambda_k I)\mathbf{p}_k^{LM}(\lambda_k) = -g_k.$$

If $J(\mathbf{x}_k)$ is numerically rank-deficient, λ_k has to be big enough so that this linear system has a solution.

4.1.2 Trust-region Levenberg-Marquardt method

Trust-region method We can also build a trust-region optimization method from the Levenberg-Marquardt method.

$$\begin{array}{ll} \min\limits_{\mathbf{p}} & \frac{1}{2}\|F(\mathbf{x}_k) + J(\mathbf{x}_k)\mathbf{p}\|^2 \\ s.t. & \|\mathbf{p}\| \leq \Delta_k \end{array} \tag{8}$$

Indeed, we have the following lemma:

Lemma (Lemma 10.2 of [NW99]). *A vector $\mathbf{p}$ is a solution of the trust-region subproblem 8 if and only if $\mathbf{p}$ is feasible and there exists a scalar $\lambda_k \geq 0$ such that*

$$(B_k + \lambda_k I)\mathbf{p} = -g_k$$
$$\lambda_k(\Delta_k - \|\mathbf{p}\|) = 0.$$

This means that when the solution of the Gauss-Newton subproblem ($\mathbf{p}_k^{GN}$, or equivalently $\mathbf{p}_k^{LM}(0)$), lies strictly in the trust region (i.e. $\|\mathbf{p}_k^{GN}\| < \Delta_k$), it also solves the trust-region subproblem. Otherwise there exists $\lambda_k > 0$ such that the Levenberg-Marquardt solution $\mathbf{p}_k^{LM}(\lambda_k)$ solves the trust-region subproblem and lies on the boundary (i.e. $\|\mathbf{p}_k^{LM}(\lambda_k)\| = \Delta_k$).

Therefore, when the Gauss-Newton solution is not in the trust-region, we need to find a $\lambda_k > 0$ such that $\|\mathbf{p}_k^{LM}(\lambda_k)\| = \Delta_k$. For that, we will solve for λ_k the following form with Newton's method and Cholesky decomposition (see Algorithm 1):

$$\frac{1}{\|\mathbf{p}_k^{LM}(\lambda_k)\|} - \frac{1}{\Delta_k} = 0.$$

Once this is done, to decide if we accept the step, we compute the ratio between the *actual reduction* and the *predicted reduction*:

$$\pi(\mathbf{p}_k^{LM}(\lambda_k)) = \frac{\|F(\mathbf{x}_k)\|^2 - \|F(\mathbf{x}_k + \mathbf{p}_k^{LM}(\lambda_k))\|^2}{\|F(\mathbf{x}_k)\|^2 - \|F(\mathbf{x}_k) + J(\mathbf{x}_k)\mathbf{p}_k^{LM}(\lambda_k)\|^2}. \tag{9}$$

If this ratio if below some positive threshold η (= 1/4 for example), the step is rejected and the size of the trust-region is decreased by a factor γ (= 1/6 for example). Otherwise the step is accepted: $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k^{LM}(\lambda_k)$. See Algorithm 2 for the complete algorithm.

Algorithm 1: Trust-region lambda subproblem. Algorithm 4.3 from [NW99]

```
Data: $\lambda_0$, $\Delta_k > 0$
$\lambda = \lambda_0$;
repeat
    Apply Cholesky decomposition: $B + \lambda I = R^T R$ where $R$ is upper-triangular;
    Solve for $p$: $R^T R p = -g$;
    Solve for $q$: $R^T q = p$;
    $\lambda = \lambda + \left(\frac{\|p\|}{\|q\|}\right)^2 \frac{\|p\| - \Delta}{\Delta}$
until $\Delta_k - \|\mathbf{p}_k^{LM}(\lambda)\| < \varepsilon$;
return $\lambda$
```

Algorithm 2: Trust-region Levenberg-Marquardt step

```
Data: $\theta_k, \mu_k, \eta = \frac{1}{4}, \gamma = \frac{1}{6}, \nu = 1.1, q$
Evaluate $B_k = J(\theta_k)^T J(\theta_k)$ and $g_k = J(\theta_k)^T (F(\theta_k))$;
$\Delta_k = \mu_k \|F(\theta_k)\|$;
repeat
    Solve for $\mathbf{p}$: $B_k \mathbf{p} = -g_k$;
    if $\|\mathbf{p}\| < \Delta_k$ then
        $\mathbf{p}_k = \mathbf{p}$;
    else
        With Algorithm 1, find $\lambda_k > 0$ and associated $\mathbf{p}_k = \mathbf{p}_k^{LM}(\lambda_k)$ s.t. $\|\mathbf{p}_k^{LM}(\lambda_k)\| = \Delta_k$;
    Compute $\pi_k(\mathbf{p}_k)$ with Equation 9;
    if $\pi_k(\mathbf{p}_k) \geq \eta$ then
        $\Delta_k = \gamma \Delta_k$;
until $\pi_k(p_k) < \eta$;
Set $\theta_{k+1} = \theta_k + \mathbf{p}_k$;
$q_k = \|F(\theta_k) + J(\theta_k)\mathbf{p}_k\| / \|F(\theta_k)\|$;
$\mu_{k+1} = \begin{cases} \mu_k/6 & \text{if } q_k < q \\ 2\mu_k & \text{if } q_k > \nu q \\ \mu_k & \text{otherwise} \end{cases}$;
```

4.1.3 Application and results

Application to our problem We get back to our inverse PDE problem. We optimize the p parameters θ of our network $\Phi_\theta : \mathbf{x} \mapsto (u_\theta(\mathbf{x}), c_\theta(\mathbf{x}))$ with the previous trust-region Levenberg-

Marquardt method using the following objective function F.

$$\begin{array}{rccl} F: & \mathbb{R}^p & \to & \mathbb{R}^{m+r} \\ & \theta & \mapsto & (u_\theta(\mathbf{x}_M) - \bar{u},\ \mathcal{N}[u_\theta, c_\theta](\mathbf{x}_R)) \end{array}$$

Results We have not had the time to make this method work on our inverse PDE problem. However, we have implemented it and noticed that it was able to solve simple optimisation problems, like fitting the output of a network to given measures.

4.2 A regularizing method for a constrained formulation

In this subsection, we propose to see our inverse problem as a constrained optimization formation that we solve with a method kwown as Sequential Linear Programming (SLP). This section leans notably on Sections 18.5 (Trust-Region SQP Methods) of [NW99] and on [RBS19].

4.2.1 Constrained formulation

For now, all our formulations of the inverse problem consisted to roughly:

- fit the estimated u with the measures $\bar{u}$;
- minimize the residual of the PDE.

But with noisy measures, the first point may be questioned. We could rather use a constraint to say that the error on u must not be bigger than some threshold, namely the level of noise. In other words, we would solve this following constrained optimization problem:

$$\begin{array}{cl} \min\limits_{\theta} & \mathcal{L}_{\mathrm{R}}(\theta) \\ s.t. & \mathcal{L}_{\mathrm{M}}(\theta) \leq \delta \end{array}$$

where δ is the expected level of noise on the measures.

4.2.2 Sequential Linear Programming

We want to solve the following constrained minimization problem, using the Sequential Linear Programming (SLP) framework.

$$\begin{array}{cl} \min\limits_{\theta} & f(\theta) \\ s.t. & g(\theta) \leq \delta \end{array} \tag{10}$$

Penalization The first step consists to transform the constrained into a penalization term weighted by ν:

$$\min_{\theta} \quad \Phi(\theta) \triangleq f(\theta) + \nu \max\{g(\theta) - \delta, 0\} \tag{11}$$

Step and linearization The idea is to solve the problem iteratively, with steps of the form $\theta_{k+1} = \theta_k + d_k$ where d_k is the minimizer in a trust-region of the following subproblem, corresponding to a linear model ℓ of the objective function.

$$\begin{array}{cl} \min\limits_{d_k} & \ell(d_k) \triangleq f(\theta_k) + \nabla f(\theta_k) \cdot d_k + \nu_k \max\{g(\theta_k) + \nabla g(\theta_k) \cdot d_k - \delta, 0\} \\ s.t. & \|d_k\|_\infty \leq \Delta_k \end{array} \tag{12}$$

with the infinite norm on d_k to have linear constraints.

Remark. *We can also take a more complicated model for the subproblem, as a quadratic model (with requires to compute the hessian), leading to the method known as Sequential Quadratic Programming (SQP).*

We translate this into a true linear program by introducing a dummy variable t, that we solve with a linear solver:

$$\begin{array}{rl} \min\limits_{d_k,t} & f(\theta_k) + \nabla f(\theta_k) \cdot d_k + \nu_k t \\ s.t. & \|d_k\|_\infty \leq \Delta_k \\ & t \geq 0 \\ & t \geq g(\theta_k) + \nabla g(\theta_k) \cdot d_k - \delta \end{array} \tag{13}$$

Step acceptance We accept the step according to the ratio between the *actual reduction* and the *predicted reduction*:

$$\rho_k = \frac{\Phi(\theta_k) - \Phi(\theta_k + d_k)}{\ell(0) - \ell(d_k)} \tag{14}$$

More precisely, if this ratio is too small, we shrink the trust region and do the step again. In addition, if this ratio is sufficient big, we can even widen the trust region. See Algorithm 3.

Algorithm 3: SLP step

```
Data: $\theta_k, \Delta_k, \nu_k, 0 < \rho_{\text{bad}} < \rho_{\text{good}} < 1$
Evaluate $\nabla f(\theta_k)$, $\nabla g(\theta_k)$;
Solve linear sub-problem 13;
Compute the step evaluation parameter $\rho_k$ 14;
if $\rho_k \leq \rho_{\text{bad}}$ then
    $\Delta_{k+1} = \frac{1}{2}\Delta_k$
else if $\rho_k \geq \rho_{\text{good}}$ then
    $\Delta_{k+1} = \min\{2\Delta_k, \Delta_{\max}\}$
else
    $\Delta_{k+1} = \Delta_k$
if $\rho_k > \rho_{\text{bad}}$ then
    accept the step: $\theta_{k+1} = \theta_k + d_k$
else
    reject the step: $\theta_{k+1} = \theta_k$
```

Remark. *In the setting where there is no constraint, the sub-problem 13 can be easily solved without a specific linear programming solver, by simply taking for each i*

$$(d_k)_i = \begin{cases} -\Delta_k & \text{if } (\nabla f(\theta_k))_i > 0 \\ +\Delta_k & \text{otherwise} \end{cases}$$

Remark. *At each step, we put* $\nu_k = \max\{\lambda_k, \overline{\lambda_k}\}$ *where* λ_k *and* $\overline{\lambda_k}$ *are the Lagrange multipliers of the constraints 13c and 13d in the linear sub-problem.*

4.2.3 Application and results

Results Similarly to Section 4.1, we have not achieved to make this method work on our inverse PDE problem, but we have implemented it and tested it successfully on simpler non-inverse problems.

Conclusion

This internship has been an opportunity to go into optimization in depth and to discover methods I had never really studied, like trust-region methods. Although it has not led to very convincing methods for the noisy inverse PDE problem, it has permitted to highlight some serious difficulties and to identify some promising research trails. I have also produced Python / PyTorch codes – implementing PINNs for inverse problems and custom optimizers based on trust-region methods – that will be used in further works.

References

[BMR16] Stefania Bellavia, Benedetta Morini, and Elisa Riccietti. On an adaptive regularization for ill-posed nonlinear systems and its trust-region implementation. *Computational Optimization and Applications*, 64(1):1–30, 2016.

[NW99] Jorge Nocedal and Stephen J Wright. *Numerical optimization.* Springer, 1999.

[RBPK17] Samuel H Rudy, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Data-driven discovery of partial differential equations. *Science advances*, 3(4):e1602614, 2017.

[RBS19] Elisa Riccietti, Stefania Bellavia, and Stefano Sello. Sequential linear programming and particle swarm optimization for the optimization of energy districts. *Engineering Optimization*, 51(1):84–100, 2019.

[RPK19] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[SMB$^+$20] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020.