Université de Limoges
ED 653 : SCIENCES ET INGENIERIE - XLIM
Faculté des Sciences et Techniques

Thèse
pour l'obtention le grade de Docteur de l'Université de Limoges

dans la spécialité

Mathématiques Appliquées et Applications des Mathématiques

Optimization and machine learning algorithms applied to the phase control of an array of laser beams

présentée et soutenue publiquement par
Maksym Shpakovych

le 2 décembre 2022

devant le jury composé de

Remerciements

Pendant ces trois années, j'ai eu l'occasion unique de toucher au monde de la recherche en mathématiques appliquées aux problèmes de la physique et d'améliorer considérablement mes compétences professionnelles. Cela n'aurait pas été possible sans mes directeurs de recherche, les professeurs Paul Armand en mathématiques appliquées et Vincent Kermene, directeur de recherche au département de photonique, qui sont tous deux des professionnels incontestables dans leur domaine.

Le professeur Paul Armand est, sans exagération, un expert de haut niveau. C'est la seule personne que j'ai rencontrée dans ma vie qui peut présenter ses idées d'une manière aussi méthodique, claire et simple et, ce faisant, transmettre la richesse des connaissances qu'il possède à des étudiants comme moi. Dire que je lui suis reconnaissant, c'est ne rien dire. Les qualités personnelles de cet homme sont difficiles à surestimer.

Vincent Kermene, directeur de recherche, est sans aucun doute un professionnel hautement qualifié, avec qui il a été très intéressant et instructif de travailler. Au cours de cette période, mon amour pour la physique et pour l'apprentissage du fonctionnement de notre monde s'est non seulement développé, mais j'ai également acquis de nombreuses connaissances en traitant avec lui. Les qualités personnelles de Vincent, telles que l'ouverture et la convivialité, ont certainement facilité mon adaptation à l'équipe, me permettant d'interagir efficacement avec ses membres et d'obtenir des résultats importants.

Je suis également reconnaissant au reste de l'équipe, à savoir Alain Barthélémy, directeur de recherche émérite, Alexandre Boju, Professeur Agnès Desfarges-Berthelemot, et Geoffrey Maulion, qui appartiennent au département de photonique. Ce fut un plaisir de travailler avec eux pendant cette période. Sans aucun doute, chacun d'entre eux a contribué aux objectifs définis et c'est grâce au travail d'équipe que nous avons pu atteindre nos objectifs.

Cette thèse a été réalisée avec le soutien financier du Comité d'Animation Scientifique et Interdisciplinaire (CASI) du laboratoire XLIM et du Labex Σ-LIM.

Contents

Chapter 1

Introduction

This work studies algorithms for the dynamic phase control of an array of laser beams [31]. The main interest from the physical point of view is to obtain a synthetic beam of high brightness (high power and high beam quality). There exist many physical problems where a beam of high power is required. The first application comes from the military goal to efficiently hit potentially dangerous objects such as self-guided missiles or drones over big distances and is developing in the context of the Tactical Advanced Laser Optical System project (TALOS). The main requirement is that the propagated light must have a high destructive effect, which can be transmitted over a long distance without tangible energy losses. Another application uses a pressure of photons to move light-driven nano-spacecraft in the vacuum of space, which is equipped with a light sail (Breakthrough Starshot), where the same approach can be applied to clean the earth's orbital space from debris (CleanSpace). One more interesting application raises from the need to accelerate particles using a high power laser (ICAN), where the main user is Conseil Européen pour la Recherche Nucléaire (CERN). To achieve the defined task the

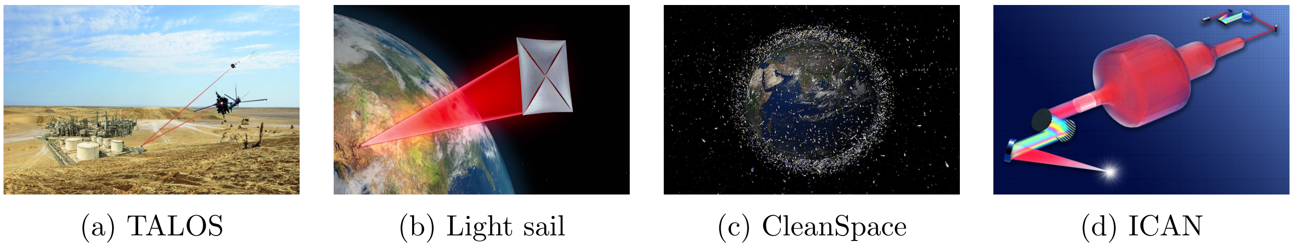

(a) TALOS (b) Light sail (c) CleanSpace (d) ICAN

Figure 1.1: Possible applications of a beam with a high brightness.

coherent beam combining [6] approach is applied. The physical fact that is extensively used here is the interference of light waves, which is controlled by their phases. Thus, clearly, if several parallel waves are in phase, in the far field they overlap and exhibit an intense central lobe. An example of such a process is depicted on Figure 1.2, where the near field on the left (an intensity pattern of the four output beams of the laser system) and the far field on the right (an intensity pattern after propagation) are presented. It can be seen that depending on beams phases a coherent in phase beams (high peak at the center) or incoherent, without stable phase relationships (smooth intensity distribution) intensity pattern can be obtained.

This problem can be formulated in terms of a mathematical model. Each j-th beam is a light wave with its own amplitude and phase. This information can be encoded into a complex

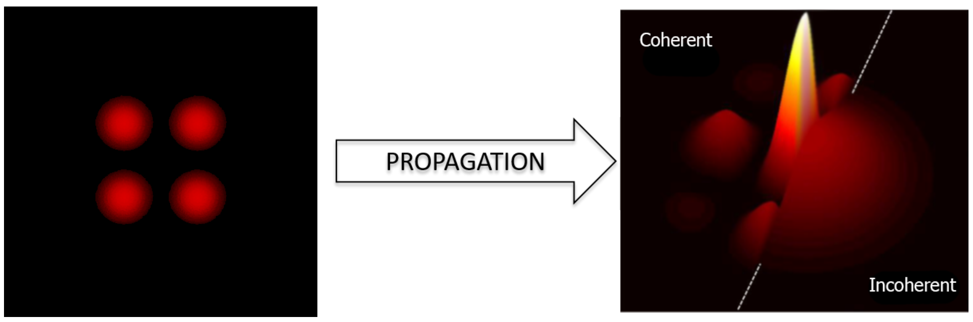


Figure 1.2: Coherent and Incoherent combining example: left, intensity pattern of 4 beams (near field), right, intensity pattern of the their far field when beams are coherent (in phase case) or incoherent (without stable phase relationships)

number $x_j \in \mathbb{C}$, which has an amplitude-phase representation $x_j = \alpha_j \cdot e^{i\varphi_j}$, where $\alpha_j \geq 0$ equals to a square root of beam intensity, and $\varphi_j \in [-\pi, \pi[$ represents its phase.

Then, the problem of maximization of brightness for an array of n laser beams can be formulated in terms of the following mathematical model

$$\left| \sum_{j=1}^{n} \alpha_j \cdot e^{i\varphi_k} \right| \to \max_{\{\varphi_1,\ldots,\varphi_n\}} .$$

Clearly, it reaches maximum when $\varphi_1 = \cdots = \varphi_n$, which coincides with physical interpretation. However, despite the simplicity of the formulation, there are several difficulties, which increase the complexity of the problem.

The first difficulty is that direct phase measurements $\varphi_1, \ldots, \varphi_n$ are not possible, which means that these values are not observable directly. The only available possibility is phase modulations by given values $\delta\varphi_1, \ldots, \delta\varphi_n$. One option to receive information about the phases of an array of laser beams is to measure the intensities of the transmitted laser rays through an optical diffuser. The transformation returns the phase-less information which can be used for a vector of phase corrections $\varphi \in [-\pi, \pi[^n$ computation. This idea is a basis for an opto-numerical algorithm developed in [31]. In practice, the algorithm is applied continuously and never stops. The reason comes from the fact that the phases of the beam array change continuously at high frequency up to some kilohertz, which is the case due to the intrinsic noise of the independent amplifiers and because of the sensitivity of the laser system to environmental perturbations such as pressure, temperature or acoustics. Thus, the optical part of the algorithm is done by means of noisy intensity measurements from a photonic system plus phase modulations, and the numerical part is an algorithm, which computes $\varphi \in [-\pi, \pi[^n$.

The second difficulty is that we have a need to correct unknown phases $\varphi_1, \ldots, \varphi_n$ not just to a constant value to obtain a co-phased far field pattern, but to an arbitrary vector of targets $\varphi_1^t, \ldots, \varphi_n^t$. This vector describes the atmospheric perturbations which frequently changes and must be taken into account to maintain a high power density far away from the laser system. Thus, it is extremely important for the numerical algorithm to be adaptive and to not require extra computational operations to change the target.

The third difficulty is a presence of a high level of noise in the experimental process. There are at least two possible sources which are the noise during measurements and phase corrections.

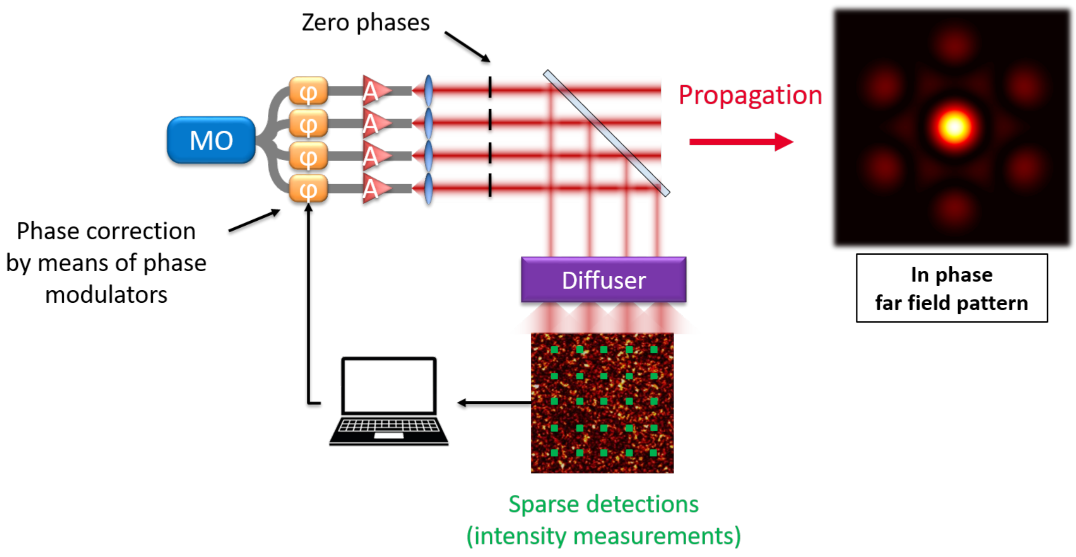

Zero phases
Propagation
MO
φ
A
Phase correction by means of phase modulators
Diffuser
In phase far field pattern
Sparse detections (intensity measurements)

Figure 1.3: Phase correction to a zero target (MO: master oscillator, φ: phase modulator, A: fiber amplifier)

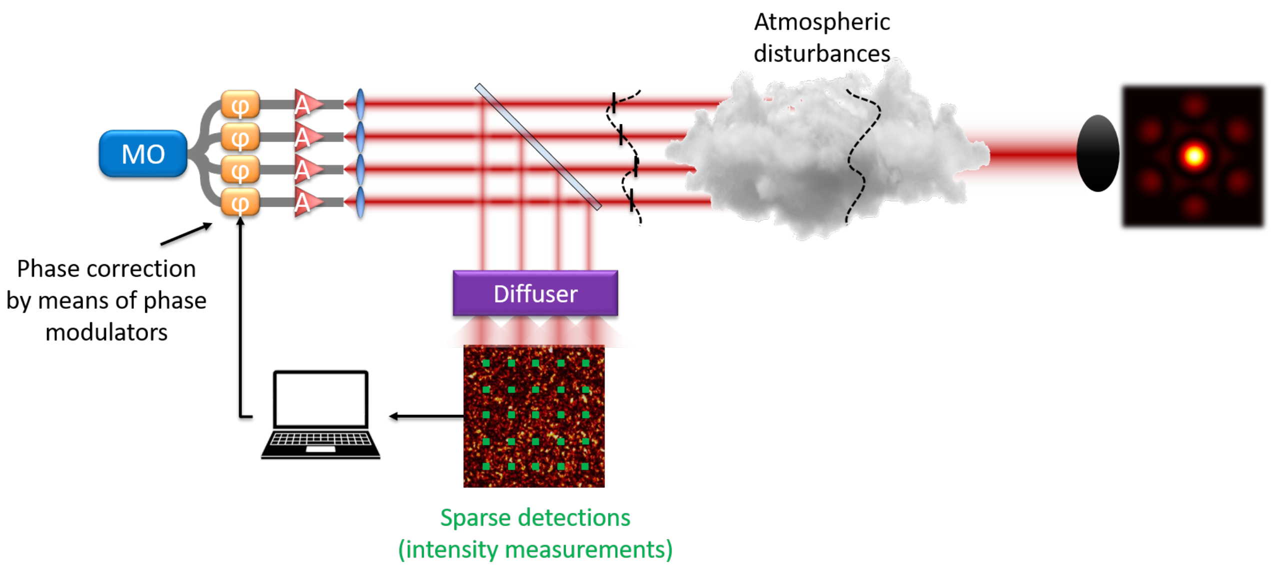

Atmospheric disturbances
MO
φ
A
Phase correction by means of phase modulators
Diffuser
Sparse detections (intensity measurements)

Figure 1.4: Phase correction to an atmospheric compensation target.

Thus, it is extremely important that the algorithm, which computes a phase correction vector $\delta\varphi$, is highly robust.

The crucial point, which is the main topic of this work, is to select from existing numerical algorithms or to develop a new one, that can perform a phase reconstruction satisfying the following criteria:

- Relevancy: the ability of the algorithm to compute a sequence of corrections capable to hit the target signal with high precision.
- Rapidity: the ability to do it in a short time (less than one millisecond).

- Robustness: the algorithm should be as insensitive as possible to a high level of experimental noises.
- Scalability: the algorithm must remain efficient with an increasing number of input sources.
- Adaptability: dynamic updates of the target signal must be taken into account. The correction process must be able to quickly adapt to a change in the target signal.

1.1 Notation

All vector inequalities are understood componentwise. Let x be a vector with complex components. The j-th component of x is denoted by x_j. The vectors $|x|$ and $\arg(x)$ are the vectors those components are the modulus and argument of each component of x. By convention we set $\arg(0) = 0$. When we write $|x| + c$ or $\arg(x) + c$, for $c \in \mathbb{C}$, the operation is understood componentwise, which means $|x_j| + c$ and $\arg(x_j) + c$ for all j. Given two vectors x and y in $\mathbb{C}^n$, their Euclidean scalar product is denoted by $\langle x, y\rangle = x^* y = \bar{x}^\top y$ and means that $\langle x, y\rangle = \sum_{j=1}^n \bar{x}_j y_j$, where $\bar{x}_j$ is a complex conjugate. The associated norm is $\|x\| = \langle x, x\rangle^{1/2}$. Let us consider a matrix $A \in \mathbb{C}^{m\times n}$, then a_j means the j-th row and $a_{\cdot j}$ denotes the j-th column. The component by component product of two vectors is denoted by $x \odot y = (x_1y_1, \dots, x_ny_n)^\top$ for $x, y \in \mathbb{C}^n$. The same sign is used if we multiply vector by matrix componentwise which means that for $A = (a_{\cdot 1}, \dots, a_{\cdot n}) \in \mathbb{C}^{m\times n}$ and $x \in \mathbb{C}^m$ we have that $A \odot x = (a_{\cdot 1} \odot x, \dots, a_{\cdot n} \odot x)$. Let us denote by $A(:)$ the operation of flattening a matrix into a vector such that $A(:) = (a_{11}, \dots, a_{1n}, a_{21}, \dots, a_{2n}, \dots, a_{m1}, \dots, a_{mn})$.

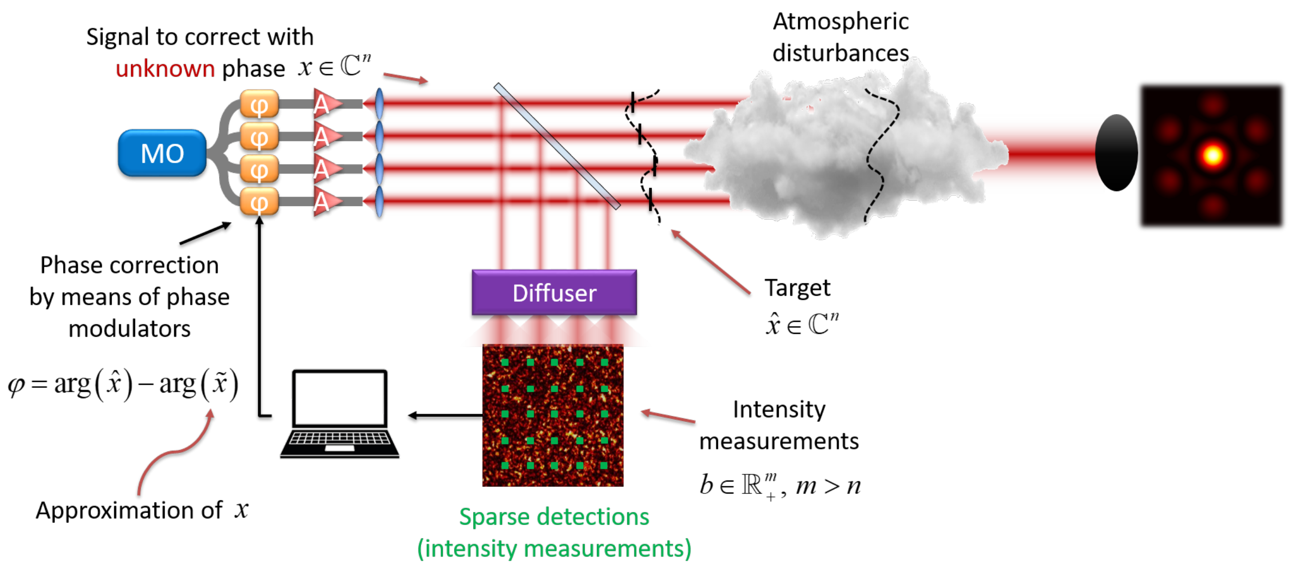

Signal to correct with
unknown phase $x \in \mathbb{C}^n$
Atmospheric disturbances
MO
φ
A
Phase correction by means of phase modulators
$\varphi = \arg(\hat{x}) - \arg(\tilde{x})$
Approximation of x
Diffuser
Target
$\hat{x} \in \mathbb{C}^n$
Intensity measurements
$b \in \mathbb{R}^m_+,\ m > n$
Sparse detections (intensity measurements)

Figure 1.5: Mathematical notations for physical values.

Let us also introduce the notations for physical values that are used in this work frequently. By $x \in \mathbb{C}^n$ we denote a set of beams the phases of which we want to correct. The target phases $\varphi^t = (\varphi^t_1, \dots, \varphi^t_n)$ are encoded into a complex vector $\widehat{x} \in \mathbb{C}^n$ such that $\widehat{x} = |x| \odot e^{i\varphi^t}$, where t

in superscript means "target". By $b \in \mathbb{R}_+^m$ we denote a set of intensity measurements of lasers that were sent through a diffuser. By $\widetilde{x} \in \mathbb{C}^n$ we denote an approximation of x by means of some numerical algorithm that uses intensities b for this goal.

1.2 Phase retrieval problem

This work studies the phase retrieval algorithms, that can be split into two families:

- The *model-based* family of algorithms uses a mathematical description of the physical process of diffusion, scattering, and measurements for an array of laser beams. A mathematical model together with measured intensities is then used for phase recovery.
- The *model-less* family of algorithms tries to approximate the physical transformation by means of a neural network (parametric function) from some family, where a finite set of measurements and phases is required to learn the dependence from scratch.

The conceptual difference between these two families is that the first one uses a model with inputs to produce an output vector, whereas the second case is about building a model that approximates the desired mapping and then can be used for phase recovery. Pictorially, the difference is described on Figure 1.6.

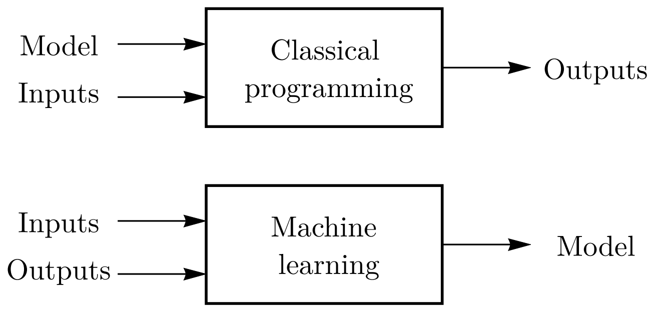


Figure 1.6: Machine learning paradigm.

These two families are considered in current work as the options for a *phase retrieval algorithm* block depicted on Figure 1.7 to compare and select the best one.

Let us consider the first family of phase retrieval algorithms. The main idea is to model a signal scattering by a linear map $x \to \{\langle \bar{a}_j, x\rangle\}_{j=1}^m$, where $A = (a_1, \dots, a_m)^\top \in \mathbb{C}^{m\times n}$ is called a *transfer matrix* or *transmission matrix*, n is a number of signals, and m is a number of detectors, and to model signal measurements by a square of modulus $x \to |x|^2$. Let $b^2 \in \mathbb{R}_+^m$ be a vector of intensity measurements returned by physical detectors. Then, the problem of signals reconstruction with a given vector b and known matrix A is called a *phase retrieval* (see, e.g., [22, 32]) and writes as

$$\text{Find } x \in \mathbb{C}^n \text{ such that } |\langle \bar{a}_j, x\rangle| = b_j, \text{ for } j \in \{1, \dots, m\}, \tag{1.1}$$

or equivalently

$$\text{Find } x \in \mathbb{C}^n \text{ such that } |Ax| = b. \tag{1.2}$$

Note, that a multiplication x by an arbitrary complex number with a unit amplitude does not change the measurements

$$|\langle \bar{a}_j, x \cdot e^{i\theta} \rangle| = |\langle \bar{a}_j, x \rangle| \text{ for all } \theta \in [-\pi, \pi[,$$

which means that a signal x can be reconstructed up to a constant shift in phase.

To use this model a transmission matrix $A \in \mathbb{C}^{m\times n}$ is required. Physically, it depends on the scattering device which is used in the experimental setup. A simple lens corresponds to a Fourier transform of the signal which can be described as the matrix $A \in \mathbb{C}^{m\times n}$. In this case, knowing the characteristics of a lens the transmission matrix can be easily built. In contrast, if a scattering device is represented by a diffuser, as for the opto-numerical algorithm [31], then each j-th row of matrix is computed by means of a phase retrieval algorithm

$$\text{Find } a_j \in \mathbb{C}^n \text{ such that } |\langle \bar{x}_k, a_j \rangle| = b_{kj} \text{ for all } j \in \{1, \dots, m\},\ k \in \{1, \dots, N\},$$

or equivalently

$$\text{Find } A \in \mathbb{C}^{m\times n} \text{ such that } |XA^\top| = B,$$

where sets of signals $X \in \mathbb{C}^{N\times n}$ and measurements $B \in \mathbb{R}_+^{N\times m}$ obtained experimentally are required. To build a transmission matrix A the number of measurements m must be specified in order to have a possibility to solve (1.2). There are results [1, 11], where it is proved that when $m \geq 4n - 4$ then the problem (1.2) always has a unique solution up to a constant phase for any *generic* A. Following [2], the term "generic" can be thought of as the property which is satisfied with probability 1 by measurement vectors $\{a_i\}_{i=1}^m \subseteq \mathbb{C}^n$ drawn from continuous distributions. In other words, if columns of matrix A were drawn from a continuous probability distribution then mapping $x \to |Ax|$ is injective and thus a solution of (1.2) is unique up to a constant phase. Despite the fact of uniqueness, it is still a difficult problem to retrieve vector $x \in \mathbb{C}^n$ by its measurements $b \in \mathbb{R}_+^m$.

For this aim, a huge variety of numerical algorithms were developed. The main idea is to reformulate (1.2) in terms of a minimization problem with some useful properties for a specific algorithm. For instance, the reformulation

$$\begin{array}{ll} \min_{(x,y)\in\mathbb{C}^n\times\mathbb{C}^m} & \frac{1}{2}\|Ax - y\|^2 \\ \text{s.t.} & |y| = b. \end{array} \tag{1.3}$$

can be solved in terms of iterative projections between the range of a transmission matrix $\text{range}(A) = \{y \in \mathbb{C}^m : y = Ax, \forall x \in \mathbb{C}^n\}$ and the set $\mathcal{M}_b = \{y \in \mathbb{C}^m : |y| = b\}$. This algorithm is known as *alternating minimization* or *alternating projection* algorithm [25, 23]. It is also sometimes referred as the Gerchberg–Saxton algorithm [45], while the original Gerchberg–Saxton algorithm [14] is when the matrix A corresponds to a discrete Fourier transform.

This formulation can be modified by introducing variables u and θ which replaces y in (1.3) by $u \odot e^{i\theta}$ and writes as

$$\begin{array}{ll} \min_{(x,u,\theta)\in\mathbb{C}^n\times\mathbb{R}_+^m\times\mathbb{R}^m} & \frac{1}{2}\|u - b\|^2 \\ \text{s.t.} & Ax = u \odot e^{i\theta}. \end{array} \tag{1.4}$$

This reformulation is used in [20] to solve (1.2) by means of *Alternating Directions Method of Multipliers* [28, §4.4] which is also a part of projection algorithms family. In contrast to the alternating projections algorithm, ADMM requires a way of computing a regularization parameter ρ, which is an additional difficulty.

There is also the reformulation which is useful for the descent direction family of algorithms.

$$\min_{x\in\mathbb{C}^n} \quad \frac{1}{2}\||Ax|^2 - b^2\|^2 \tag{1.5}$$

Using a square of left and right parts of (1.2), the function to minimize in (1.5) is differentiable in terms of Wirtinger calculus (Section 2.3). This property gives an ability to apply the descent algorithms like *gradient descent* [7], *Gauss-Newton* [13] or other.

There is the family of algorithms that is based on the reformulation of (1.2) in terms of a relaxed semi-definite programming [8, 45]. One possible formulation

$$\begin{array}{ll} \min_{X\in\mathbb{C}^{n\times n}} & \operatorname{trace}(X) \\ \text{s.t.} & \operatorname{trace}(a_j a_j^* X) = b^2 \\ & X \succeq 0 \end{array} \tag{1.6}$$

where $X = xx^*$ when exact recovery occurs. Otherwise, it is reasonable to compute a normalized leading eigenvector of X to obtain a solution of (1.2). This relaxation is called as PhaseLift [8].

Another reformulation from this family writes as

$$\begin{array}{ll} \min_{U\in\mathbb{C}^{n\times n}} & \operatorname{trace}(UM) \\ \text{s.t.} & \operatorname{diag}(U) = 1_n \\ & U \succeq 0 \end{array} \tag{1.7}$$

where $M = \operatorname{diag}(b)(I - AA^\dagger)\operatorname{diag}(b)$. If $\operatorname{rank}(U) = 1$, then the solution of (1.2) is such that $U = xx^*$. Otherwise, the normalized leading eigenvector of U is used as an approximate solution of (1.2). This approach known as a MaxCut [45]. The number of parameters to optimize in (1.7) and (1.6) raise with a quadratic rate if n increases. Since an efficiency of a phase retrieval algorithm is the main criteria in this work, reformulations (1.6) and (1.7) are not considered to use.

Let us consider the second family of phase retrieval algorithms. In this case, there are no assumptions on the form of connection between phases of beams and the measurements. The only information that could be given is a set of N couples $D = \{(x_i, b_i)\}_{i=1}^N$ where $x_i \in \mathbb{C}^n$ and $b_i \in \mathbb{R}_+^m$ are input signals and experimentally performed measurements. The goal is to use D as training data to obtain an artificial neural network (NN) that approximates measurements to signals mapping. One of the schemes covered in the published literature [17] relies on a phase recovery by a convolutional neural network, such as in the pioneering work on NN for adaptive optics [30]. The NN serves to map the intensity of an interference pattern of the beam array directly into the distribution of phase in the array. The simulations reported in [17] shows that the accuracy of the CNN-based phase control drops when the array increases from 7 to 19 beams. This is a limitation that was also highlighted in the field of wavefront sensing so that NNs were often used only as a preliminary step for initialization of an optimization routine [27]. The numerical experiments in [46] also confirms that a direct phase recovery appears to be a complicated task.

1.3 Phase correction problem

Schematically, the process of phase control for a set of n beams is described on Figure 1.7. There, one can see that the input beam array $x \in \mathbb{C}^n$, the phases of which are unknown but with known amplitudes, is passed through the phase modulation block where phases can be changed using a vector $\varphi \in [-\pi, \pi[^n$. Then, beams are split by a beamsplitter where a reflected part is sent to the output and a passed part is sent to a diffuser. This device mixes the input beams which produces an interference pattern which is called speckle. The intensities of this pattern are measured by means of photodetectors visualized by green dots on the speckle image and stored in the vector $b \in \mathbb{R}^m_+$, where $m > n$. These measurements together with the target signal $\widehat{x}$ are then given to the phase retrieval algorithm, the goal of which is to reconstruct approximately the measured signals x using b and $\widehat{x}$. The phases of a retrieved approximation $\widetilde{x} \in \mathbb{C}^n$ are then subtracted from the given phases of the target signal $\widehat{x}$, and sent to the phase modulation block. The meaning of phase correction vector $\varphi = \arg(\widehat{x}) - \arg(\widetilde{x})$ can be clearly

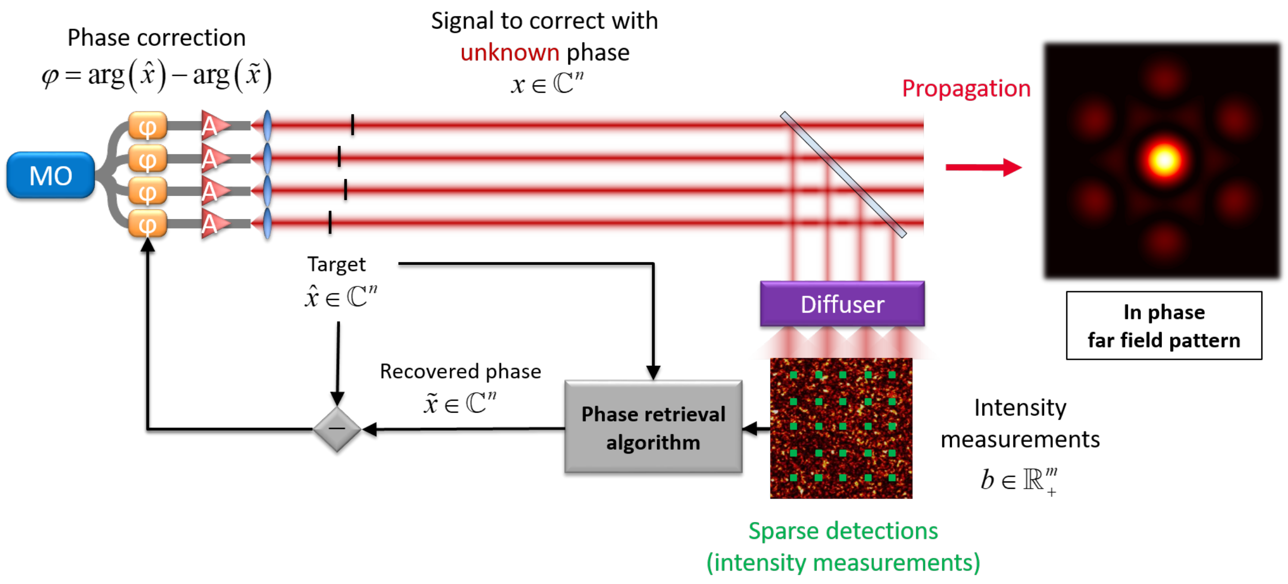


Figure 1.7: Opto-numerical algorithm [31].

seen in the case of one laser (Figure 1.8). The vector φ, which is applied to x, moves it closer to the target $\widehat{x}$. Thus, we have that

$$\begin{aligned}\arg(x') &= \arg(x) + \arg(\widehat{x}) - \arg(\widetilde{x})\\ &= \arg(x) + \varphi,\end{aligned}$$

from where it is clear that $\arg(x') = \arg(\widehat{x})$ if retrieved phases $\arg(\widetilde{x})$ coincide with $\arg(x)$.

However, one correction can be not enough to lock the phases to the prescribed values. That is why the process of measurements and corrections repeats several times. At this level, it is important to understand a difference between a *phase correction* and *phase retrieval* problems. As it was revealed above, by phase correction task we consider a problem to set beams' phases to the prescribed target values, where the initial beams phases are not observable. A *phase retrieval* problem aims to approximate x using its measurements b (see, e.g., [22, 32]). A particular application of phase retrieval algorithm in a phase correction loop requires target $\widehat{x}$ to be used as an initial point, but originally it is not required.

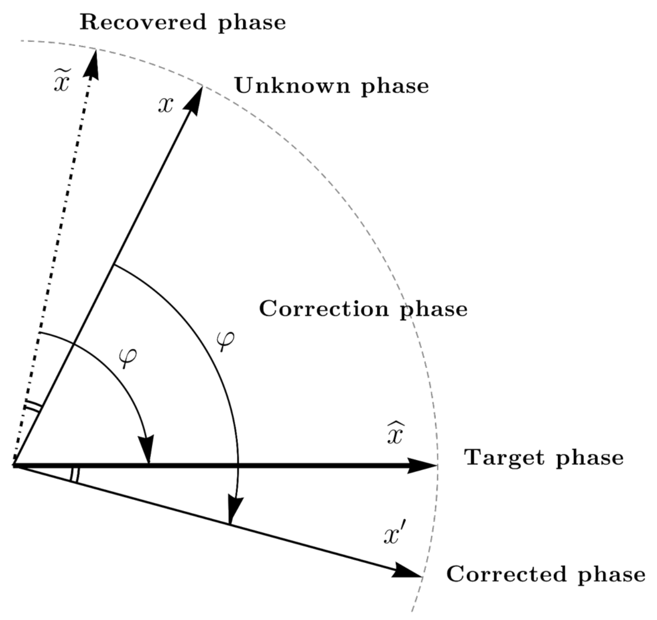


Figure 1.8: One dimensional phase correction visualization.

Chapter 2

Preliminaries

2.1 Noise and data generation

As it is discussed in Chapter 1, the experimental data contains a high level of noise. Thus, one of the main requirements for the algorithms that are considered in this work is robustness. To verify it numerically, the noise must be generated and added during numerical simulations, which gives the ability to estimate the robustness of the algorithm before using it in experiments. This section explains how noise is generated during numerical simulations.

In practice, the noise comes from intensity measurements by means of a camera or photodetectors. Experimental intensity measurements $b \in \mathbb{R}_+^m$ together with input signals $x \in \mathbb{C}^n$ are used to find parameters $a_j \in \mathbb{C}^n$ for $j \in \{1, \dots, m\}$ of the mathematical model (1.2), which leads to a presence of noise in it. To simulate that numerically it is proposed to use the following approach

$$A' = A + \sigma\varepsilon, \tag{2.1}$$

where σ controls the level of noise, $\varepsilon \in \mathbb{C}^{m\times n}$ be a random matrix where $\mathrm{Re}\{\varepsilon_{ij}\}, \mathrm{Im}\{\varepsilon_{ij}\} \sim \mathcal{N}(0,1)$, $A \in \mathbb{C}^{m\times n}$ be the generated transmission matrix in the same way as ε, which is used to compute a vector of measurements $b \in \mathbb{R}_+^m$, and A' is a noisy transmission matrix, which will be given to the algorithm. Thus the algorithm will try to find $x \in \mathbb{C}^n$ such that $|Ax| = b$ using A' and b. The same conditions are for initialization methods.

2.2 Metrics

In this section, we review several metrics than can be used to evaluate the distance between two signals represented by two vectors x and y in $\mathbb{C}^n$.

The first metric, used in [31][1], is useful to only compare the arguments of two vectors. It is defined by

$$\mathrm{q}(x,y) = \sqrt{1 - \left|\frac{\langle x, y\rangle}{\langle |x|, |y|\rangle}\right|^2}. \tag{2.2}$$

Proposition 1 *For all vectors x and y in $\mathbb{C}^n$, with all nonzero components, $q(x,y) = 0$ if and only if there exists $c \in \mathbb{R}$ such that* $\arg(x) = \arg(y) + c$.

[1]In [31] the phasing quality is defined as $1 - \mathrm{q}(x,y)^2$.

Proof. Let $(x,y) \in \mathbb{C}^n \times \mathbb{C}^n$ such that $x_k \neq 0$ and $y_k \neq 0$ for all $k = 1, \dots, n$. The assertion follows from the triangle inequality[2]. Indeed, we have

$$\begin{aligned} |\langle x,y\rangle| &= \left|\sum_{k=1}^{n} |x_k||y_k|e^{i(\arg(y_k)-\arg(x_k))}\right| \\ &\leq \sum_{k=1}^{n} |x_k||y_k| \\ &= \langle |x|, |y|\rangle, \end{aligned}$$

with an equality if and only if $\arg(y_k) = \arg(x_k) + c$, with $c = \arg(y_1) - \arg(x_1)$. □

The metric (2.2) is sensitive to changes in $|x|$ and $|y|$ (Example 1) which is redundant for our applications where just phase changes are interesting to consider.

Example 1 (q(x,y) sensitivity) *Let us consider $x = (1, \alpha \cdot e^{i\varphi})$ and $y = (1,1)$ for some $\alpha > 0$ and $\varphi \in [0, 2\pi]$. Then we obtain that*

$$\begin{aligned} q(x,y) &= \sqrt{1 - \left|\frac{\langle x,y\rangle}{\langle |x|,|y|\rangle}\right|^2} \\ &= \sqrt{1 - \frac{|1+\alpha\cdot e^{i\varphi}|^2}{|1+\alpha|^2}} \\ &= \sqrt{1 - \frac{(1+\alpha\cos\varphi)^2 + (-\alpha\sin\varphi)^2}{(1+\alpha)^2}} \\ &= \sqrt{1 - \frac{1+2\alpha\cos\varphi+\alpha^2}{(1+\alpha)^2}} \end{aligned}$$

Let us visualize this function with respect to $\alpha \in (0, 10]$ and $\varphi \in [0, 2\pi]$. It can be seen on

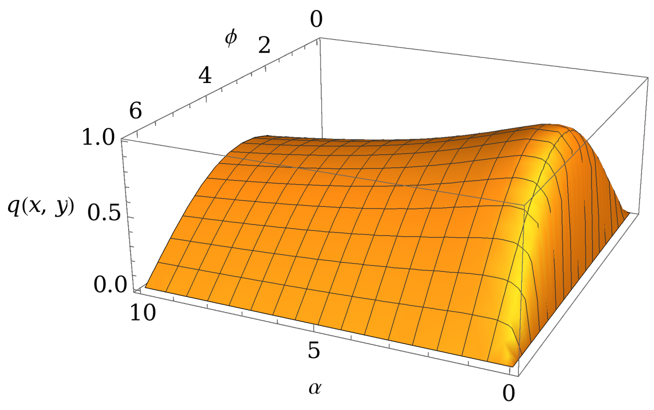


Figure 2.1: q(x,y)

Figure 2.1 that for any fixed $\varphi \in [0, 2\pi]$ the value of $q(x,y)$ is not constant when α is modified.

[2]Let $z_1, \dots, z_n$ be complex numbers. We have $|\sum_{k=1}^{n} z_k| \leq \sum_{k=1}^{n} |z_k|$. Moreover, if $z_k \neq 0$ for all k, then the previous inequality is an equality if and only if for all $k = 2, \dots, n$ z_k/z_1 is a positive real number.

It is proposed to use a normalized quality metric

$$\mathrm{q}_{\mathrm{norm}}(x,y) = 1 - n^{-2}|\langle e^{i\arg(x)}, e^{i\arg(y)}\rangle|^2. \tag{2.3}$$

Proposition 2 *For all vectors x and y in $\mathbb{C}^n$, with all nonzero components, $q_{norm}(x,y) = 0$ if and only if there exists $c \in \mathbb{R}$ such that $\arg(x) = \arg(y) + c$. Moreover, $q_{norm}(x,y)$ is such that $q_{norm}(x,y) = q\big(\frac{x}{|x|}, \frac{y}{|y|}\big)^2$.*

Proof. The first assertion is shown following the same approach as in Proposition 1 setting $|x| = |y| = 1_n$. The second assertion is shown by direct formula evaluation for x and y from $\mathbb{C}^n$

$$\begin{aligned}
\mathrm{q}\left(\frac{x}{|x|}, \frac{y}{|y|}\right)^2 &= 1 - \left|\frac{\left\langle \frac{x}{|x|}, \frac{y}{|y|}\right\rangle}{\left\langle \left|\frac{x}{|x|}\right|, \left|\frac{y}{|y|}\right|\right\rangle}\right|^2 \\
&= 1 - \left|\frac{\langle e^{i\arg(x)}, e^{i\arg(y)}\rangle}{\langle 1_n, 1_n\rangle}\right|^2 \\
&= 1 - n^{-2}|\langle e^{i\arg(x)}, e^{i\arg(y)}\rangle|^2
\end{aligned}$$

□

In [7], the distance between two complex vectors is defined as follows. For $(x,y) \in \mathbb{C}^n \times \mathbb{C}^n$ define

$$\mathrm{dist}(x,y) = \min_{\phi\in[0,2\pi]} \|x - e^{i\phi}y\|. \tag{2.4}$$

Proposition 3 *For all vectors x and y in $\mathbb{C}^n$, $\mathrm{dist}(x,y) = \|x - e^{-i\arg(\langle x,y\rangle)}y\|$. Moreover, $\mathrm{dist}(x,y) = 0$ if and only if $|x| = |y|$ and there exists $c \in \mathbb{R}$ such that $\arg(x) = \arg(y) + c$.*

Proof. Let $(x,y), \in \mathbb{C}^n \times \mathbb{C}^n$ and $\phi \in \mathbb{R}$. We have

$$\|x - e^{i\phi}y\|^2 = \|x\|^2 + \|y\|^2 - 2\,\mathrm{Re}(e^{i\phi}\langle x,y\rangle).$$

The result follows from the fact that $\mathrm{Re}(e^{i\phi}\langle x,y\rangle) \le |e^{i\phi}\langle x,y\rangle| = |\langle x,y\rangle|$, with an equality if and ony if $\phi = -\arg(\langle x,y\rangle)$. □

In some cases, it is also useful to use normalized distance defined as

$$\mathrm{dist}_{\mathrm{norm}}(x,y) = \frac{\mathrm{dist}(x,y)}{\max\{\|x\|, \|y\|\}}. \tag{2.5}$$

2.3 Wirtinger calculus

In this section, we explain Wirtinger calculus which is a way to compute derivatives of a real-valued function on a complex-valued domain $f : \mathbb{C} \to \mathbb{R}$. The application of Wirtinger calculus is used in this work to differentiate metrics that are defined in Section 2.2.

It is known from the analysis that complex variable function is differentiable if Cauchy-Riemann conditions are satisfied i.e. the function is holomorphic. However, the neural network model and loss can be non-holomorphic functions. In this case, Wirtinger calculus [19] can be applied. The idea of Wirtinger was that any differentiable mapping $f : \mathbb{R}^2 \to \mathbb{R}^2$ can be cast in the complex domain by introducing the conjugate coordinates and changing variables such that

$$\begin{pmatrix} z \\ \bar{z} \end{pmatrix} = \begin{pmatrix} 1 & i \\ 1 & -i \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}. \tag{2.6}$$

It means that any complex- or real-valued function

$$f(z) = f(x, y) = u(x, y) + iv(x, y),$$

of several variables can be written in the form $f(z, \bar{z})$, where f is holomorphic in $z = x + iy$ for fixed $\bar{z}$,and holomorphic in $\bar{z} = x - iy$ for fixed z. This holds as long as a the real-valued functions u and v are differentiable as functions of the real variables x and y [7]. As an example, consider

$$f(z) = (b - |a^* z|^2)^2 = (b - \bar{z}^\top a a^* z)^2 = f(z, \bar{z}),$$

with $z, a \in \mathbb{C}^n$, $b \in \mathbb{R}$. While $f(z)$ is not holomorphic in z, $f(z, \bar{z})$ is holomorphic in z with fixed $\bar{z}$ and vice versa. Then, the Wirtinger derivative can be defined as in Observation 1.

Observation 1 (Wirtinger derivative) *Let us consider a real-valued function $f(x, y) : \mathbb{R}^2 \to \mathbb{R}$. Let $f(z, \bar{z}) : \mathbb{C}^2 \to \mathbb{R}$ be the same function defined on a complex domain by changing variables as it is shown in (2.6) for $z \in \mathbb{C}$ and its conjugate $\bar{z}$, and $f(x, y)$ is differentiable with respect to x and y. Then derivatives on the complex domain can be expressed via derivatives in the real domain and write*

$$\frac{\partial f}{\partial z} := \left.\frac{\partial f(z, \bar{z})}{\partial z}\right|_{\bar{z}=const} = \frac{1}{2}\left(\frac{\partial f(x, y)}{\partial x} - i\frac{\partial f(x, y)}{\partial y}\right)$$
$$\frac{\partial f}{\partial \bar{z}} := \left.\frac{\partial f(z, \bar{z})}{\partial \bar{z}}\right|_{z=const} = \frac{1}{2}\left(\frac{\partial f(x, y)}{\partial x} + i\frac{\partial f(x, y)}{\partial y}\right).$$

Proof. From (2.6), we can explicitly express x and y in terms of z and $\bar{z}$ as

$$x = \frac{1}{2}(z + \bar{z}), \quad y = -\frac{i}{2}(z - \bar{z}).$$

Let us define $h_x(z, \bar{z}) = \frac{1}{2}(z + \bar{z})$ and $h_y(z, \bar{z}) = -\frac{i}{2}(z - \bar{z})$. Then, since

$$f(z, \bar{z}) = f(x, y) = f\big(h_x(z, \bar{z}), h_y(z, \bar{z})\big),$$

we obtain that

$$\begin{aligned}\frac{\partial f(z, \bar{z})}{\partial z} &= \frac{\partial f(x, y)}{\partial h_x}\frac{\partial h_x}{\partial z} + \frac{\partial f(x, y)}{\partial h_y}\frac{\partial h_y}{\partial z} \\ &= \frac{1}{2}\left(\frac{\partial f(x, y)}{\partial x} - i\frac{\partial f(x, y)}{\partial y}\right).\end{aligned}$$

Derivative with respect to $\bar{z}$ can be obtained in the same way. □

The idea, presented in Observation 1, can be generalized to a multivariate function in order to compute a Wirtinger gradient (Observation 2). For this aim, let us write the following useful identities, which are applied to $f(z,\bar{z})$ for $z \in \mathbb{C}$.

$$\overline{\left(\frac{\partial f}{\partial z}\right)} = \frac{\partial \bar{f}}{\partial \bar{z}}, \text{ when } f \text{ is real } \overline{\left(\frac{\partial f}{\partial z}\right)} = \frac{\partial f}{\partial \bar{z}} \tag{Conjugation rule}$$

$$\overline{\left(\frac{\partial f}{\partial \bar{z}}\right)} = \frac{\partial \bar{f}}{\partial z}, \text{ when } f \text{ is real } \overline{\left(\frac{\partial f}{\partial \bar{z}}\right)} = \frac{\partial f}{\partial z} \tag{Conjugation rule}$$

$$df = \frac{\partial f}{\partial z}dz + \frac{\partial f}{\partial \bar{z}}d\bar{z} \tag{Differential rule}$$

$$\frac{\partial (h \circ g)}{\partial z} = \frac{\partial h}{\partial g}\frac{\partial g}{\partial z} + \frac{\partial h}{\partial \bar{g}}\frac{\partial \bar{g}}{\partial z} \tag{Chain rule}$$

$$\frac{\partial (h \circ g)}{\partial \bar{z}} = \frac{\partial h}{\partial g}\frac{\partial g}{\partial \bar{z}} + \frac{\partial h}{\partial \bar{g}}\frac{\partial \bar{g}}{\partial \bar{z}} \tag{Chain rule}$$

Observation 2 (Wirtinger gradient [16]) *Let $z = (z_1, \dots, z_n)^\top \in \mathbb{C}^n$ be an n-dimensional column vector, where $z_j = x_j + iy_j$ for $j \in \{1, \dots, n\}$. Let $f : \mathbb{C}^n \to \mathbb{R}$ be a real-valued function of complex variable $z \in \mathbb{C}^n$ Then, Wirtinger gradient writes as*

$$\nabla f(z) = 2\left(\frac{\partial f}{\partial z}, \frac{\partial f}{\partial \bar{z}}\right)^*,$$

where * operation means consequent conjugation and transposition, and $\frac{\partial f}{\partial z} = \left(\frac{\partial f}{\partial z_1}, \dots, \frac{\partial f}{\partial z_n}\right)^\top$, $\frac{\partial f}{\partial \bar{z}} = \left(\frac{\partial f}{\partial \bar{z}_1}, \dots, \frac{\partial f}{\partial \bar{z}_n}\right)^\top$ for $\frac{\partial f}{\partial z_j} = \frac{1}{2}\left(\frac{\partial f}{\partial x_j} - i\frac{\partial f}{\partial y_j}\right)$, $\frac{\partial f}{\partial \bar{z}_j} = \frac{1}{2}\left(\frac{\partial f}{\partial x_j} + i\frac{\partial f}{\partial y_j}\right)$.

Proof. Let us consider a coordinate transformation (2.6), which we augment to n-dimensional case as

$$c = \begin{pmatrix} z \\ \bar{z} \end{pmatrix} = \begin{pmatrix} I & iI \\ I & -iI \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = Mr, \tag{2.7}$$

where I here is an identity matrix n by n, $x, y \in \mathbb{R}^n$. Note here that $M^{-1} = \frac{1}{2}M^*$. Since the coordinate transformation is a linear transformation it follows that for $f : \mathbb{C}^n \to \mathbb{R}$ that can be represented in different coordinates $f(z) = f(z,\bar{z}) = f(c) = f(r) = f(x,y)$ we have that

$$\frac{\partial f}{\partial r} = \left(\frac{\partial c}{\partial r}\right)^\top \frac{\partial f}{\partial c} = M^\top \frac{\partial f}{\partial c}.$$

Note, that since f is real-valued then we have that $\frac{\partial f}{\partial r} = \overline{\left(\frac{\partial f}{\partial r}\right)} = M^* \frac{\partial f}{\partial \bar{c}}$, which will be used to define gradient in c. Let us define the displacement in r as $\Delta r = -\alpha \frac{\partial f}{\partial r}$ for some $\alpha > 0$, which coincides with a displacement in a gradient descent method. Thus, we are interested in how it writes in terms of complex coordinates c. Thus we obtain that

$$\Delta c = M\Delta r = -\alpha M \frac{\partial f}{\partial r} = -\alpha M M^* \frac{\partial f}{\partial \bar{c}} = -2\alpha \frac{\partial f}{\partial \bar{c}} \tag{2.8}$$

Because $\bar{c} = (z^*, z^\top)^\top$, (2.8) can be written as

$$\Delta c = \begin{pmatrix} \Delta z \\ \Delta \bar{z} \end{pmatrix} = -2\alpha \begin{pmatrix} \dfrac{\partial f}{\partial \bar{z}} \\ \dfrac{\partial f}{\partial z} \end{pmatrix}. \tag{2.9}$$

The proof follows. □

Note, that from Observation 2 we have that the gradient of function of n complex variables is of dimension $2n$. Clearly, it is because the augmented system of coordinates $(z, \bar{z})$ is used. Thus, a displacement in the variable z only writes as $-2\alpha\frac{\partial f}{\partial \bar{z}}$ and then gradient is

$$\nabla_z f(z) = 2\frac{\partial f}{\partial \bar{z}}. \tag{2.10}$$

2.4 Tensorflow library extension

Wirtinger calculus can also be applied to compute gradients of complex-valued neural networks. This possibility is already implemented [3] in Tensorflow library, which is a popular framework to build, train, and use neural networks. The gradient of a complex-valued function here is computed as

$$\nabla_z f(z) = \overline{\frac{\partial f}{\partial z} + \frac{\partial \bar{f}}{\partial z}}, \tag{2.11}$$

which coincides with (2.10) if f is a real-valued function. Despite the fact that Tensorflow supports complex derivatives, there is no support for complex-valued layers. There exist several implementations on GitHub, which extend Tensorflow with complex-valued layers. However, not all of them are maintained by their creators, which produces difficulties in the version control of dependent libraries. For this goal, we created our own implementation (https://gitlab.xlim.fr/shpakovych/cvnn) that extensively uses a polymorphic paradigm in object-oriented programming, which gives an ability to reduce significantly the size of the library in comparison with other implementations.

2.5 Profiling method

In this section, we formulate the profiling method to be able to compare the performance of the algorithms, which are considered in this work. The main interest is to measure the required time to solve a problem by a selected algorithm. To achieve this goal we must specify both the conditions under which we conclude that the problem is solved and the way to measure time.

Definition 1 (Solved problem) *The problem is considered to be solved if the value of a distance metric between the k-th approximation of a solution and a solution itself is smaller than a selected tolerance for the last 10% percent of iterations, where the total number of iterations was fixed before.*

Example 2 (Solved problem) *Let the total number of iterations is set to* 100*. Let the tolerance value is set to* 0.001*. Let the distance metric is set to* $\mathrm{dist}_{\mathrm{norm}}$*. Let* $x^{(k)} \in \mathbb{C}^n$ *be the* k*-th approximation of a solution computed by some algorithm, and let* $x \in \mathbb{C}^n$ *be a solution. Then, the problem is considered to be solved if for all* $k > 90$ *we have that* $\mathrm{dist}_{\mathrm{norm}}(x^{(k)}, x) < 0.001$*.*

Definition 2 (Time measurement) *The time of one iteration of an algorithm is measured by counting the number of performed basic operations and multiplying them by their corresponding time of evaluation.*

Definition 3 (Time of basic operations) *The list of basic operations and their corresponding time of evaluation, which is CPU dependent and fixed in this work, is presented in Table 2.1. These values were obtained by measuring the time of performing the selected operations for random numbers 1 million times and then taking the average. For this goal, a program in C language was written.*

Operation	Time in seconds
$+$	$0.1089426 \cdot 10^{-8}$
$-$	$0.0834372 \cdot 10^{-8}$
$\times$	$0.1665280 \cdot 10^{-8}$
$/$	$0.3297135 \cdot 10^{-8}$
$\lvert \cdot \rvert$	$0.2500649 \cdot 10^{-8}$
$\sqrt{\cdot}$	$0.1788236 \cdot 10^{-8}$
$\sin(\cdot)$	$1.4113816 \cdot 10^{-8}$
$\cos(\cdot)$	$1.1528648 \cdot 10^{-8}$
$\tan(\cdot)$	$2.5544987 \cdot 10^{-8}$
$\arctan(\cdot)$	$2.2179606 \cdot 10^{-8}$
$\exp(\cdot)$	$1.3695341 \cdot 10^{-8}$

Table 2.1: Time to perform basic operations.

Example 3 (Time of basic operations) *Let us count the time of evaluation* $\sin(x)$ *for* $x \in \mathbb{R}^n$*. Following the Table 2.1, we obtain* $1.4113816 \cdot 10^{-8} n$ *seconds.*

Definition 4 (Profiling method) *The idea is to build a statistical curve that reveals which part of the problems can be solved by a selected time. The same strategy can be applied for the required number of algorithm iterations to achieve a selected tolerance.*

Example 4 (Profiling method) *For instance, from Figure 2.2 we can conclude that Alternating projection method with random initialization point can solve approximately 50% of all problems in 4 ms and requires maximum 200 iterations to solve 55% of all problems.*

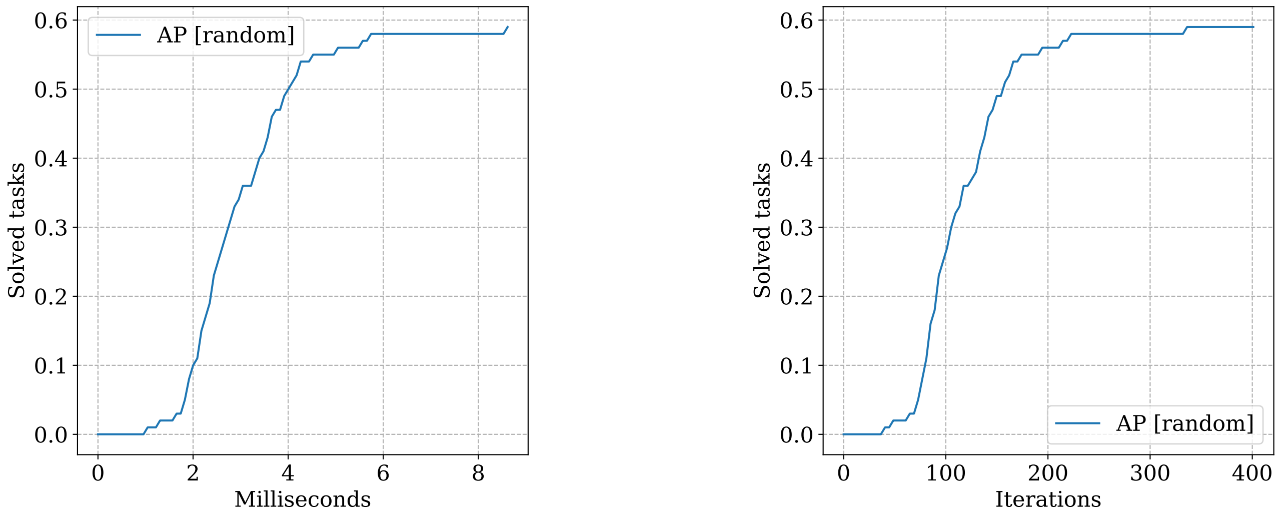


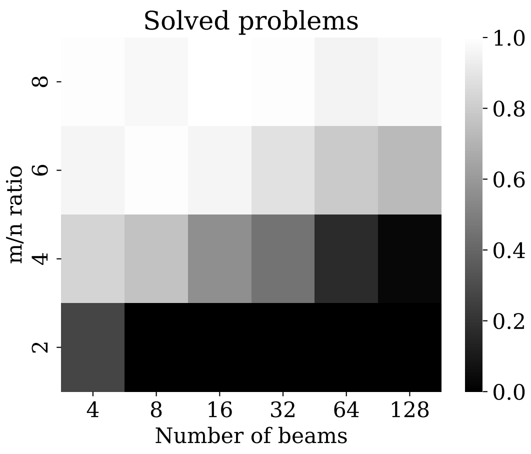
Figure 2.2: Profiles example.

2.6 Heat map method

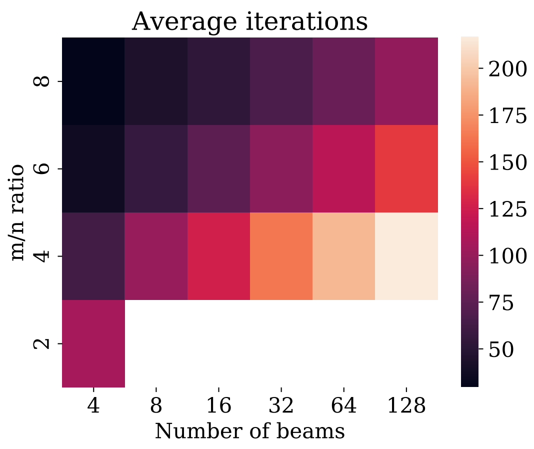
In this section, we introduce the method to explore the capabilities of the selected algorithm. In this work, we consider two kinds of problems to solve: phase retrieval and phase correction. The size of these problems is determined by the number of beams n and the number of intensity measurements m. Also, we consider two kinds of algorithms: optimization methods and neural network methods.

In the context of optimization methods, it is reasonable to check how many problems can be solved for different relations of n and m, and how many iterations it requires. Thus, we generate N problems for different n and m, try to solve them by means of an optimization method with a defined maximal number of iterations and required stopping tolerance, and count how many problems were solved, where the meaning of the solved problem is given in Definition 1. Let us consider that M problems were solved for some fixed n and m. Then, we get the M/N fraction which characterizes the solved part of all problems. Also, we compute how many iterations we require on average to solve the problem of size n by m. Then, it is convenient to visualize these results in terms of heat maps as it is presented on Figure 2.3.

In the context of neural network methods, we are more interested in a potential capability to solve problems of size defined by n and m. That is why instead of the fraction of solved problems we plot an average minimum value of the selected metric in a gray scale. The neural network method does not require iterations to solve a phase retrieval problem. That is why, instead of an average number of iterations, we plot an average relative time required to train the model of size defined by n and m (Figure 2.4).

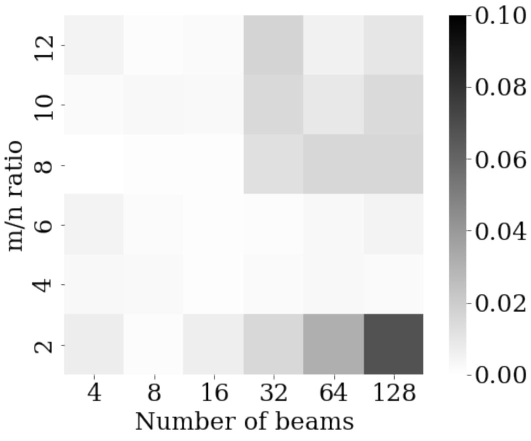


(a)

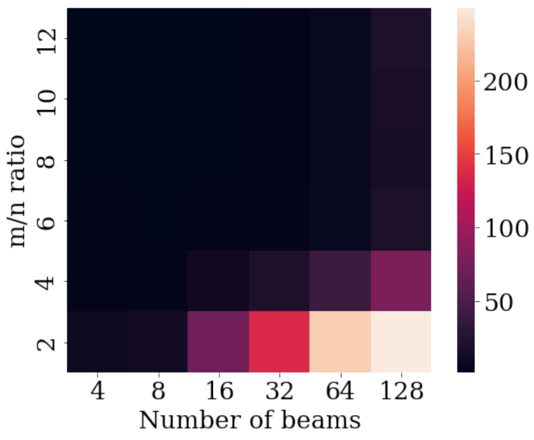


(b)

Figure 2.3: (a) Heat maps of the fraction of solved problems in grey scale and (b) its required average number of iterations.



(a)



(b)

Figure 2.4: (a) Heat maps of the minimal achievable mean q_{norm} in grey scale and (b) its required relative training time. The relative time on (b) is computed by dividing a learning time in seconds for each n and m/n by the minimal time to obtain GPU invariant information. The minimal time required by the GPU used in for this experiments was 1.3 s.

Chapter 3

Machine learning framework

Let X be a space of objects, Y be a space of responses, and let $y^* : X \to Y$ be an unknown function that exactly maps each object to its response. Let the evaluations $y_i = y^*(x_i)$ are known on a given subset $\{x_1, \dots, x_N\} \subset X$ of size N. Let $D = \{(x_i, y_i)\}_{i=1}^N$ be a set of couples which we call training sample.

The goal is to find $w \in \mathcal{W}(\mathcal{F})$ such that $a(\cdot, w) \approx y^*(\cdot)$ for

$$a : X \times \mathcal{W}(\mathcal{F}) \to Y, \tag{3.1}$$

where a is taken from some parametric family of functions $a \in \{a(x, w) : X \times \mathcal{W}(\mathcal{F}) \to Y\}$, $\mathcal{W}(\mathcal{F})$ is a space of weights over a field $\mathcal{F}$ (either $\mathbb{R}$ or $\mathbb{C}$). Each parametric family defines the form of a.

The quality of approximation y^* by a depends on the parametric family from where a was taken and on the size of training sample N. Since y^* is unknown, we can measure the quality just on the training sample or on some subset of it. Usually, people split the training sample on the train and test disjoint subsets $D_{\text{train}} = D_{i=1}^{N_1}$, $D_{\text{test}} = D_{i=N_1}^{N}$ where $N_1 = N - k$ to avoid evaluation of model on the information that was used to find the parameters $w^* \in \mathcal{W}(\mathcal{F})$. The quality approximation on a given subset can be measured by loss or cost function $\mathcal{L} : Y \times Y \to \mathbb{R}$.

If a loss function is a differentiable or at least piecewise differentiable then the task of finding parameters $w^* \in \mathcal{W}(\mathcal{F})$ can be formulated as the unconstrained optimization problem

$$w^* \in \text{argmin}_{w \in \mathcal{W}(\mathcal{F})} \mathcal{L}(a(\cdot, w), D_{\text{train}}). \tag{3.2}$$

There is a big variety of optimization methods that can be used to solve unconstrained problems. However, this set reduces with an increase of a training sample size and a dimension of parametric space $\mathcal{W}(\mathcal{F})$. For example, if it is not possible to store a training sample in the memory or the time of computing gradients is too long, then only stochastic methods like stochastic gradient descent [18] can be applied. These methods have two big advantages that are extensively used in machine learning. First, they can use a descent directions computed just on a subset of a training sample to perform an optimization step. Second, the steplength of a descent direction is computed without linesearch methods which increases an optimization speed dramatically.

When the parameters w^* are found, we obtain $a^*(\cdot) = a(\cdot, w^*)$ function that approximates $y^*(\cdot)$ in sense of loss function on the given training set. The test subset of training sample then is used to check the generalizing ability of the approximation a.

Example 5 *Let us consider the regression problem. For instance, we need to know the price y of a flat using the information about a number of rooms x_1 and a distance to the transport line x_2. And suppose that we know these characteristics for N flats. It means that we have training sample of size N in form $D = \{([x_{1,i}, x_{2,i}], y_i)\}_{i=1}^N$ where $X = \{[x_{1,i}, x_{2,i}]\}_{i=1}^N$ is the set of objects and $Y = \{y_i\}_{i=1}^N$ is the set of responses.*

Let we also have an assumption that there is a linear dependence between x_1, x_2 and y. It means that we can choose $a : X \times \mathcal{W}(\mathcal{F}) \to Y$ from the parametric family of linear functions

$$a(x, w) = w_0 + w_1 x_1 + w_2 x_2,$$

where $x = [x_1, x_2] \in \mathbb{R}^2$, $w = [w_0, w_1, w_2] \in \mathcal{W}(\mathbb{R}) = \mathbb{R}^3$. Then we define the loss function for this task. The obvious choice is the squared l_2 norm normalized by the number of objects N

$$\mathcal{L}(\hat{y}, y) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)_2^2,$$

where $\hat{y}_i = w_0 + w_1 x_{1,i} + w_2 x_{2,i}$ is the predicted i-th price, and y_i is the original i-th price from training sample.

Then, we split training sample on the train and test disjoint subsets D_{train} and D_{test} and define X_{train}, Y_{train} to be objects and responses from D_{train}, and X_{test} and Y_{test} to be objects and responses from D_{test}. Then the problem of finding parameters w can be formulated as the following optimization problem

$$w^* \in argmin_{w \in \mathcal{W}(\mathbb{R})} \frac{1}{|D_{train}|} \sum_{i=1}^{|D_{train}|} \big(a([X_{train}]_i, w) - [Y_{train}]_i\big)^2.$$

After finding the optimal parameters we can evaluate this model on the test set

$$score = \frac{1}{|D_{test}|} \sum_{i=1}^{|D_{test}|} \big(a([X_{test}]_i, w) - [Y_{test}]_i\big)^2,$$

and finish.

In the example 5, the step-by-step process of building and training the simple linear model with a squared error loss function is presented. The linear parametric family of functions which is used in 5 is appropriate for this simple problem, however its complexity increases with the complexity a problem.

Typically, a parametric family is a composition of linear blocks with nonlinear transformations between them. A common name for the elements that are used for building a parametric family is layer. There are two types of layers: trainable and non-trainable. By trainable we mean that it contains parameters to be optimized during learning. In example 5 it was the vector $w = [w_0, w_1, w_2]$. In the opposite case, there are no parameters to optimize i.e. some data transformation like dropout regularization (see Section 3.3).

3.1 Layers

In this section, we consider an example of trainable layer which is called fully connected or dense layer. The fully connected layer is an affine function $\mathrm{fc} : \mathcal{F}^n \times \mathcal{F}^{m\times n} \times \mathcal{F}^m \to \mathcal{F}^m$ that writes as

$$\mathrm{fc}(x, W, b) = Wx + b, \tag{3.3}$$

where $x \in \mathcal{F}^n$ is an input data and $W \in \mathcal{F}^{m\times n}$, $b \in \mathcal{F}^m$ are parameters to optimize. Sometimes, we will skip parameters W and b from the function arguments list and write $\mathrm{fc}(x) = Wx + b$ to simplify notation. Note, that if $m = 1$, $n = 2$ and $\mathcal{F} = \mathbb{R}$, fully connected layer coincides with the function $a(x, w)$ from Example 5 where $w_0 = b$.

The optimization requires gradients with respect to trainable variables. In the case where $\mathcal{F} = \mathbb{R}$ we have

$$\nabla_W \mathrm{fc}(x, W, b) = [x, x, \ldots, x]^\top \in \mathbb{R}^{m\times n}, \quad \nabla_b \mathrm{fc}(x, W, b) = 1_{\mathbb{R}^m}, \tag{3.4}$$

where $1_{\mathbb{R}^m}$ is a vector of ones of dimension m. In the case where $\mathcal{F} = \mathbb{C}$ we have

$$\nabla_W \mathrm{fc}(x, W, b) = [x, x, \ldots, x]^* \in \mathbb{C}^{m\times n}, \quad \nabla_b \mathrm{fc}(x, W, b) = 1_{\mathbb{C}^m}, \tag{3.5}$$

where $1_{\mathbb{C}^m}$ is a vector of $1+0i$ elements of dimension m. The complex differentiation is explained in Section 2.3.

Let us give a simple example of a binary classification problem that can be solved by one layer model. The only thing that we add which was not discussed before is a sigmoid activation function. It scales the output of a fully connected layer into the range from 0 to 1.

Example 6 *Let $\xi_1 \sim \mathcal{N}(-2, I_2)$ and $\xi_2 \sim \mathcal{N}(2, I_2)$. Here I_2 is an identity matrix 2×2. Let us sample ξ_1 100 times and collect it into X_1. Then we do the same for ξ_2 and X_2. Thus $X = [X_1, X_2] \in \mathbb{R}^{200\times 2}$ is a set of 200 objects where each element $x \in X$ was sampled either from $\mathcal{N}(-2, I_2)$ or from $\mathcal{N}(2, I_2)$ distributions. This process creates an artificial data from two classes. Then, let us prepare the responses. Let $Y_1 \in 0_{\mathbb{R}^{100}}$ and $Y_2 \in 1_{\mathbb{R}^{100}}$ which means that*

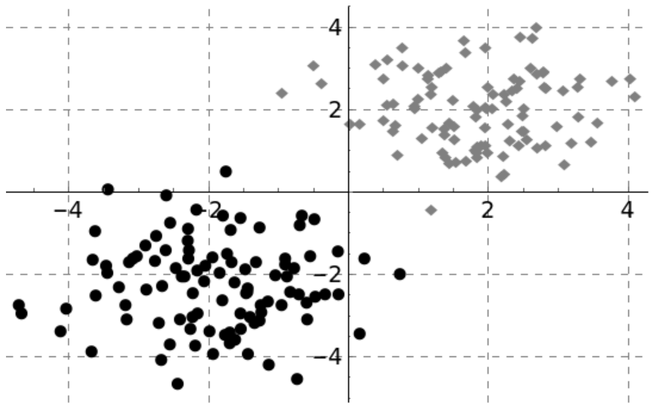


Figure 3.1: Generated objects X.

we give a label 0 for objects x that were sampled from $\mathcal{N}(-2, 1)$ and 1 for x that were sampled from $\mathcal{N}(2, 1)$. Thus $Y = [Y_1, Y_2]$ is the set of responses. At this moment we created the training *sample $D = (X, Y)$ of size 200. Let us define the parametric family of functions where we search the best one. Our model contains one fully connected layer $fc : \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}$*

$$fc(x, w, b) = w_1 x_1 + w_2 x_2 + b, \tag{3.6}$$

where $x \in \mathbb{R}^2$ is the input, $w \in \mathbb{R}^2$ and $b \in \mathbb{R}$ are the parameters to optimize. Then, since our responses are ether 0 or 1 we will scale the output of $fc(x, w, b)$ into the range $[0, 1]$ using a sigmoid function.

$$\sigma(x) = \frac{1}{1 + \exp(-x)}. \tag{3.7}$$

Thus the parametric family writes as

$$a(x, w, b) = \frac{1}{1 + \exp(-(w_1 x_1 + w_2 x_2 + b))}. \tag{3.8}$$

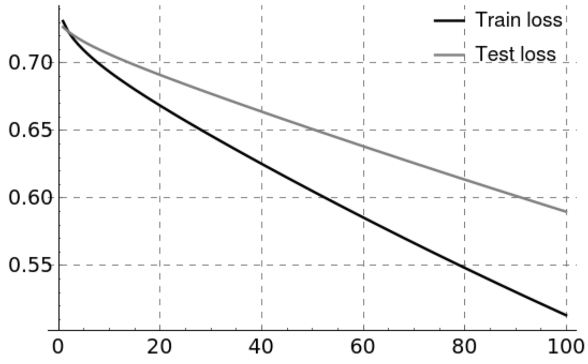
Then we need to define the loss function. For a task of binary classification where responses are either 0 or 1 there is well-known loss function that is called log loss of binary cross-entropy that came from the information theory and writes as

$$\mathcal{L}(\hat{y}, y) = -\frac{1}{N} \sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i). \tag{3.9}$$

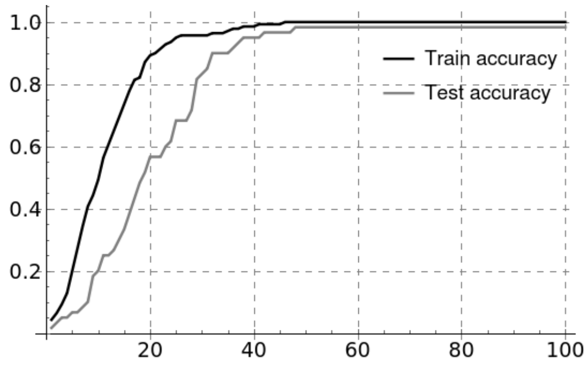
Now, we split training sample on the train D_{train} and test D_{test} disjoint subsets and define X_{train}, Y_{train} are the objects and responses from D_{train}, the same for X_{test} and Y_{test}. Then the problem of finding parameters $(w, b)^*$ can be formulated as the following optimization problem

$$(w, b)^* \in argmin_{(w,b) \in \mathbb{R}^3} \mathcal{L}(a(X_{train}, w, b), Y_{train}). \tag{3.10}$$

However, log loss is difficult to interpret in terms of quality of the approximation. In this case, we need to compute some additional metrics like accuracy, which is simply a fraction of the correct predictions to all samples in a set. The learning process is visualized in Figure 3.2 with log loss and accuracy metric on the vertical axis and the number of gradient steps on the horizontal axis. After finding the optimal parameters we can evaluate this model on the test set



(a) Log loss.



(b) Accuracy.

Figure 3.2: Learning process.

$$score = \mathcal{L}(a(X_{test}, w, b), Y_{test}) \tag{3.11}$$

Also, there is an interpretation of optimal parameters $(w, b)^*$, equation $w_1^* x_1 + w_2^* x_2 + b^* = 0$ defines a separation hyperplane between objects of two classes (Figure 3.3).

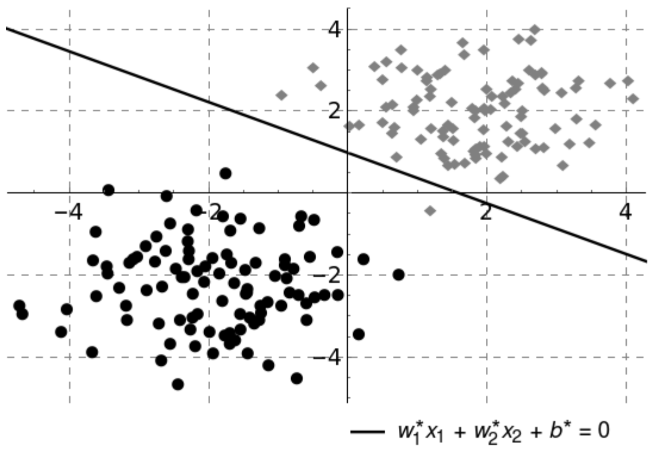


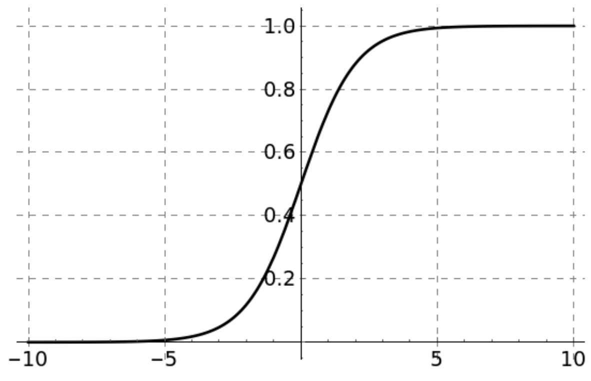
Figure 3.3: Separation hyperplane.

3.2 Activation functions

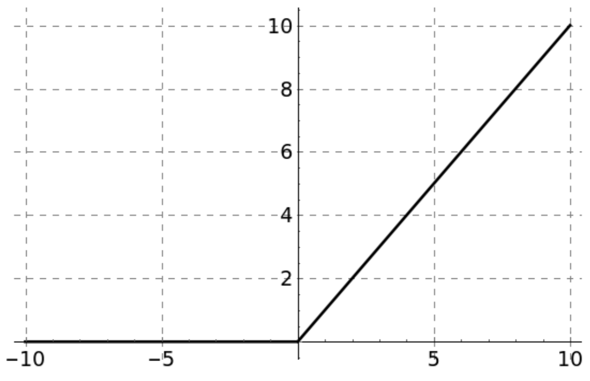
There is a wide range of special functions that can be applied for model building. Usually, they are added between layers that allow learning nonlinear hidden dependencies in data. Even though the number of activation functions exists, some core functions are commonly used. One of then is a sigmoid function that was used in Example 6. It scales the input data into the range from 0 to 1 by the formula

$$\sigma(x) = \frac{1}{1 + \exp(-x)},$$

and the plot is depicted on Figure 3.4a. Usually, this layer is applied as a last one and mainly



(a) Sigmoid function.



(b) ReLU function.

Figure 3.4: The most common activation functions.

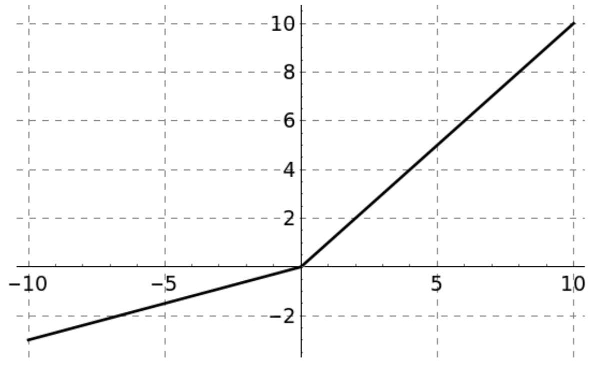
for classification tasks as it was shown in Example 6. Another important function is ReLU (rectified linear unit). It is usually applied in the multi-layer models and used as a nonlinear transformation between layers. It writes as

$$\text{relu}(x) = \max(0, x),$$

with the corresponding plot on Figure 3.4b.

There are also some modifications of ReLU that can be useful for better convergence. They are LeakyReLU and ELU (exponential linear unit).

$$\text{LeakyRelu}(x, \alpha) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases}, \quad \text{ELU}(x, \alpha) = \begin{cases} \alpha * (\exp(x) - 1) & x < 0 \\ x & x \geq 0 \end{cases}$$


(a) Leaky ReLU function ($\alpha = 0.3$).

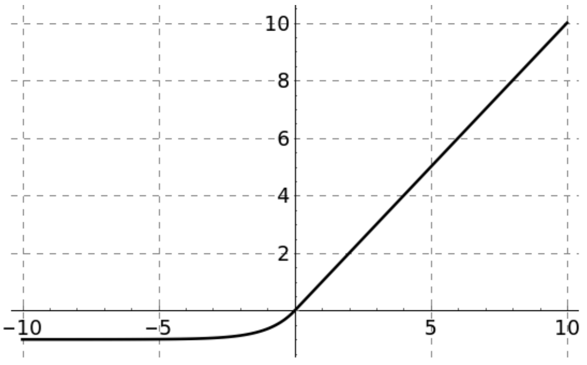

(b) ELU function ($\alpha = 1.0$).

Figure 3.5: ReLU function modifications.

3.3 Regularization

Regularization is a layer for reducing the overfitting (or memorization) phenomenon during learning. The indicator of overfitting is a different behavior of loss value during the learning process on a train and test sets. The case when loss decreases on a train set and increases on a test is called overfitting. The classical regularizers are l_1 or l_2 norm of the trainable parameters, which is added as an additional term to a loss function to prevents high variance for model parameters that leads to its better generalization.

However, for neural networks, there is a more efficient state-of-the-art regularizer which is called dropout [42]. The idea is randomly set to zero some parts of neurons during training. This encourages each neuron to be independently useful and does not rely on the output of other neurons. More precisely, with some predefined probability $p \in [0, 1]$ the elements of vector y (which is for example $y = \text{fc}(x)$) are set to zero. Other elements are scaled up by coefficient $\frac{1}{1-p}$ so that the expected value is preserved.

3.4 Loss function

The loss function or cost function is a function that maps input values $\mathcal{I} \subset \mathbb{R}^n$ onto a real number which can be interpreted as a cost.

$$\mathcal{L} : \mathcal{I} \to \mathbb{R}. \tag{3.12}$$

This kind of functions is used to measure a quality of approximation of an unknown dependence between inputs and outputs by some model a. Each kind of tasks requires a special loss function. For instance, mean squared error measure is useful for regression task

$$\mathrm{MSE}(\hat{y}, y) = \|\hat{y} - y\|_2^2, \tag{3.13}$$

where $\hat{y} \in \mathbb{R}^n$ is predicted values by a model, $y \in \mathbb{R}^n$ is the original values that were taken from the training sample. However, this function starts being less useful for classification tasks for m classes where a cross-entropy loss plays an important role and writes as

$$\mathcal{H}(\hat{y}, y) = -\frac{1}{n} \sum_{k=1}^{n} \sum_{i=1}^{m} y_{k,i} \log(\hat{y}_{k,i}), \tag{3.14}$$

where $y, \hat{y} \in [0, 1]^{n \times m}$ are predicted and true classes where $y_k \in \mathbb{R}^m$ is a one-hot vector (full of zeros except one entry with 1) and $\hat{y}_k \in \mathbb{R}^m$ is a probability vector ($\hat{y}_k \geq 0$ for $k \in \{1, \dots, m\}$ and $\sum_{i=1}^{m} \hat{y}_{k,i} = 1$).

3.5 Gradients computing

The step of computing gradient is crucial for understanding the learning process. As was mentioned at the beginning of this chapter, by learning we consider a process of minimization of a loss function with respect to trainable parameters. In most of the cases, machine learning uses the first-order derivatives for optimization. The main motivation is a computational speed which is a crucial part in machine learning. Usually, the model is represented by the composition of the functions which leads us to the result in analysis about differentiation of a composite function that is known as a chain rule.

$$\frac{d(f_1 \circ \cdots \circ f_n)(x)}{dx} = \frac{df_1}{df_2} \cdot \cdots \cdot \frac{df_n}{dx}.$$

This result is a core for model differentiation in a machine learning world which is used to define a Back-propagation algorithm, which is just the another name for algorithmic differentiation [29] method that was developed to track an inner data interaction and efficiently compute derivatives. We will not give a detailed explanation of this area in current work but just consider that gradients at given point $x_0 \in \mathcal{F}^n$ for a composite functions can be computed fast for millions of trainable parameters.

3.6 Supervised learning

When the training sample is collected, the model is built, the loss function is defined we are ready to train or learn the model. Recall, that by learning, we mean the process of solving an optimization problem of the form

$$w^* \in \mathrm{argmin}_{w \in \mathcal{W}(\mathcal{F})} \frac{1}{N} \sum_{j=1}^{N} \mathcal{L}(a(x_j, w), y_j), \tag{3.15}$$

where $\mathcal{W}(\mathcal{F})$ is a space of trainable parameters over field $\mathcal{F}$ ($\mathbb{R}$ or $\mathbb{C}$), N is the size of training sample $\{(x_i, y_i)\}_{i=1}^{N}$, x_i and y_i for $i \in \{1, \dots, N\}$ are objects and responses, $a(x, w)$ is the function from some parametric family. Then, learning algorithm 1 can be applied.

Algorithm 1: Supervised learning algorithm

```
1. Initialization:
    (a) Let $w_0 \in \mathcal{W}(\mathcal{F})$ be the starting value for trainable parameters.
    (b) Let $a : X \times \mathcal{W}(\mathcal{F}) \to Y$ be a model to learn.
    (c) Let $\mathcal{L} : Y \times Y \to \mathbb{R}$ is a loss function.
    (d) Let $X_{train} \subset X$, $Y_{train} \subset Y$ are train objects and responses.
    (e) Let $\mathcal{U} : \mathcal{W}(\mathcal{F}) \times \mathcal{W}(\mathcal{F}) \to \mathcal{W}(\mathcal{F})$ is a functions that computes parameters update
        with respect to a given descent direction $\Delta w$ and state $w$.
   Set $k = 0$.
2. Gradients computing: $\Delta w_k = \nabla_w \mathcal{L}(a(X_{train}, w), Y_{train})|_{w=w_k}$
3. Gradients applying: $w_{k+1} = \mathcal{U}(w_k, \Delta w_k)$
4. Return loop: Set $k = k + 1$ and go to 2.
```

3.7 Quasi-Reinforcement learning

In this section, we consider a special kind of learning, which is called Reinforcement learning or simply RL. To be more precise, we need discuss a modification of this process that has been developed in this work. However, to have a full picture in mind, we need to start from the original algorithm.

High-level definition reveals that the reinforcement learning is devoted to learn an agent how to do an action $a_t \in A$ at the current state $s_t \in S$ to maximize reward $r_t \in R$ of the environment $\mathcal{E} : A \to S \times R$. Let us give a simple example of the reinforcement learning problem where all of these parts are presented.

Example 7 *Let as consider a stick balancing problem. A stick is attached by an un-actuated joint to a cart, which moves along a frictionless track. This is our environment $\mathcal{E} : A \to S \times R$. The characteristics of the stick at time t are 4 numbers: linear and angular position and velocity. They are the state and thus $s_t \in S = \mathbb{R}^4$. The system is controlled by applying a force of $+1$ or -1 to the cart. The ± 1 force is the action $a_t \in A = \{-1, 1\}$. A reward $r_t \in R = \{+1\}$ is provided for every timestep that the stick remains upright. The goal is to create agent : $\mathbb{R}^4 \to \{-1, 1\}$ that keep the stick from falling over and thus maximize the total reward $r = \sum_{t=0}^{T} r_t$ where T is a maximal number of steps per simulation.*

Reinforcement learning approach proposes to find the agent from the parametric family of neural network functions. The main difficulty is that we can not formulate it as supervised problem since we do not know the correct action a_t at state s_t and thus the reward is not differentiable because it computes by environment which is considered as a black box. This fact leads to the different approaches of learning agent which are all based on the repetitive collection of a huge number of triplets $\{(a_t, s_t, r_t)\}_{t=0}^{T}$ that are used to update weights of an agent. However, in this section, we are not concentrated on the explanation of the ways of updating weights using the triplets, but on the problem when correct actions are known and can be returned from the environment $\mathcal{E} : A \times A \to S \times A$, and, as a consequence, the reward can be calculated in the same way as a loss. The motivation is that the phase correction problem can be formulated in this framework. Thus we call it quasi-reinforcement learning to point on the fact that it is not classical case. The algorithm of quasi-reinforcement learning can be formulated as the Algorithm 2. Note, that it requires a possibility to easily generate states an correct actions at step 2 to perform learning. The application of this approach to the phase correction problem is presented in Chapter 6.

Algorithm 2: QRL algorithm

```
1. Initialization:
    (a) Let $w^{(0)} \in \mathcal{W}(\mathcal{F})$ be the starting value for trainable parameters.
    (b) Let $Agent : S \times \mathcal{W}(\mathcal{F}) \to A$ be a model to learn.
    (c) Let $\mathcal{R} : A \times A \to \mathbb{R}$ is a reward function.
    (d) Let $\mathcal{E} : A \times A \to S \times A$ is an environment function.
    (e) Let $\mathcal{U} : \mathcal{W}(\mathcal{F}) \times \mathcal{W}(\mathcal{F}) \to \mathcal{W}(\mathcal{F})$ computes parameters update concerning a given
        descent direction $\Delta w$ and state $w$.
    (f) $T \in \mathbb{N}\backslash\{0\}$ is the maximal number of steps and $N \in \mathbb{N}\backslash\{0\}$ is the size of training
        sample.
    **Set** $k = 0$.
2. Training samples generation: To generate initial states and correct actions
        $S_0 = \{s_0^{(i)}\}_{i=0}^N, \quad A_0 = \{a_0^{(i)}\}_{i=0}^N.$
3. Environment discovering: **Set** $t = 0$, $w_t^{(k)} = w^{(k)}$.
    (a) To predict actions $\hat{A}_t = Agent(S_t, w_t^{(k)})$.
    (b) To compute gradients $\Delta w_t^{(k)} = \nabla_w \mathcal{R}(\hat{A}_t, A_t)|_{w=w_t^{(k)}}$
    (c) To apply gradients $w_{t+1}^{(k)} = \mathcal{U}(w_t^{(k)}, \Delta w_t^{(k)})$
    (d) To send actions to the environment $S_{t+1}, A_{t+1} = \mathcal{E}(\hat{A}_t, A_t)$
    (e) **Set** $t = t + 1$. **If** $t < T$ **then** $w^{(k)} = w_t^{(k)}$ **and go to** (3a).
4. Return loop: **Set** $k = k + 1$ **and go to** 2.
```

Chapter 4

Phase retrieval algorithms

In this chapter we present a set of phase retrieval algorithms that can be used as a numerical part of the opto-numerical algorithm [31]. The first two sections Section 4.2 and Section 4.3 describe the descent direction optimization algorithms: gradient descent and Gauss-Newton. It is known that the gradient descent iteration is cheep if a line search strategy selected properly. It makes this algorithm a good candidate for the application in a phase correction loop. Also, despite the fact that the Gauss-Newton method requires just the first order derivatives, the calculation of a Jacobian and solving the system is a computationally expensive part. However, it is known that this algorithm requires far less iterations to achieve a selected tolerance. This is the reason to check this algorithm also. Then, Section 4.5 and Section 4.6 describe the optimization algorithms from the family of the projection methods: the Alternating Projections (AP) and the Alternating Direction Method of Multipliers (ADMM). It is known that AP and ADMM require a low number of operations for one update, which also makes them good candidates for current applications. Another point that is considered in this chapter and discovered in Section 4.7 is the initialization strategies [13, 21] which aim to find a starting point in the neighborhood of a solution. It is shown in this work that a good initial point can increase a speed of the phase correction algorithm significantly. The last section presents the numerical results which show the capabilities of the considered algorithms.

4.1 Research path

The goal that we tried to achieve in this section is to try different optimization algorithms to solve a phase retrieval problem and to find the best one with respect to the requirements defined in the introduction section. As was mentioned before, there are many reformulations of the original problem which leads to different phase retrieval algorithms. So the goal was also to find a proper formulation.

The most interesting result we achieved was with the ADMM optimization algorithm. We started from the publication [20] where the authors implement the ADMM algorithm to solve a phase retrieval problem. The crucial thing that was observed is that if we decrease ρ we increase the number of problems that the algorithm can solve. However, there was no possibility to set $\rho = 0$ because of the division operation. That is why we tried to adapt update formulas in order to remove ρ completely. In this way, we discovered the Algorithm 8. Then, we searched for the minimization problem which corresponds to these update formulas and found (4.21).

The next step was to ensure that the algorithm is robust. Unfortunately, even with a small level of noise the algorithm does not converge (see Figure 4.8b). This observation can be easily explained since the system (1.2) is overdetermined, which implies that, almost surely, there does not exist $y \in \text{range}(A)$ and $z \in \mathcal{M}_b$ such that $y = z$. As a result there is no limit point for the sequence $\lambda^{(k+1)} = \lambda^{(k)} + y^{(k+1)} - z^{(k+1)}$ (see Proposition 8).

To overcome this difficulty we introduced the relaxed minimization problem (4.22), which aims to replace the strong linear constraint by $y - z = \xi$. This gives us the optimization algorithm Algorithm 9. However, then we need to deal with a regularization parameter ρ. The formulas for the adaptive update of ρ can be found in (4.25) and (4.26). Here, the idea is simple. When we are far from a solution, ρ must be small to guide the sequences $z^{(k)}$ and $y^{(k)}$ to the "perfect solution" even if it does not exist. As soon as the distance between $z^{(k)}$ and $y^{(k)}$ is small, then we set $\rho = 1$, which means that we are using the alternating projection algorithm.

At the end of Section 4.6, we found the algorithm that satisfies all the requirements and, at the same time, converges faster than all other algorithms considered in this chapter.

4.2 Gradient Descent Method

In this section we describe the gradient descent method for solving the optimization problem

$$\min_{x\in\mathbb{C}^n} \quad f(x) := \tfrac{1}{2m}\||Ax|^2 - b^2\|^2. \tag{4.1}$$

It can be easily seen that the solution of (4.1) coincide with the solution of (1.2). Using the Wirtinger calculus (2.10) it is possible to compute the gradient, which in a matrix form writes as

$$\nabla_x f(x) = \frac{1}{m} A^*\left[(|Ax|^2 - b^2) \odot Ax\right],$$

and can be used in the gradient descent step $x^{(k+1)} = x^{(k)} - \alpha \nabla_x f(x^{(k)})$ where $\alpha \in (0, 1]$ is a step length to be computed by one of the line search algorithms and $x^{(0)} \in \mathbb{C}^n$ is an initial point for the algorithm. Then, the gradient descent method can be formulated as Algorithm 3.

Algorithm 3: Gradient descent algorithm.

```
    Input: Initial point $x^{(0)} \in \mathbb{C}^n$
    Output: Approximate solution $x \in \mathbb{C}^n$ of (1.2).
1   Set $k = 0$;
2   Loop
3       $p^{(k)} = -\nabla_x f(x^{(k)})$;
4       compute a step length $\alpha^{(k)}$ by some line search method;
5       $x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)}$;
6       if stopping test is satisfied then
7           break
8       $k = k + 1$;
9   return: $x = x^{(k+1)}$
```

The most computationally expensive part is the process of finding the step length $\alpha^{(k)}$. Thus, it is extremely important to select a line search strategy appropriately in order to maintain efficiency of the Algorithm 3. One of the simplest line search approach is a backtracking algorithm [26], which aims to find α that satisfies the following condition

$$f(x) > f(x - \alpha \nabla_x f(x)) + c_1 \alpha \|\nabla_x f(x)\|^2, \tag{4.2}$$

where $c_1 \in (0, 1)$. The idea is simple: starting from $\alpha = 1$, we decrease it with some rate until (4.2) is satisfied. This approach is presented in Algorithm 4.

Algorithm 4: Back-tracking line search [26].

```
   Input: Initial step length $\alpha^{(0)} \in (0,1)$, current point $x \in \mathbb{C}^n$, gradient $\nabla_x f(x)$
          evaluated in $x$, decrease rate $\rho \in (0,1)$, constant $c_1 \in (0,1)$
   Output: Step length $\alpha$ that satisfies (4.2).
1  $k = 0$;
2  while $f(x) < f(x - \alpha^{(k)} \nabla_x f(x)) + c_1 \alpha^{(k)} \|\nabla_x f(x)\|^2$ do
3      $\alpha^{(k+1)} = \rho \alpha^{(k)}$;
4      $k = k + 1$;
5  return: $\alpha = \alpha^{(k)}$
```

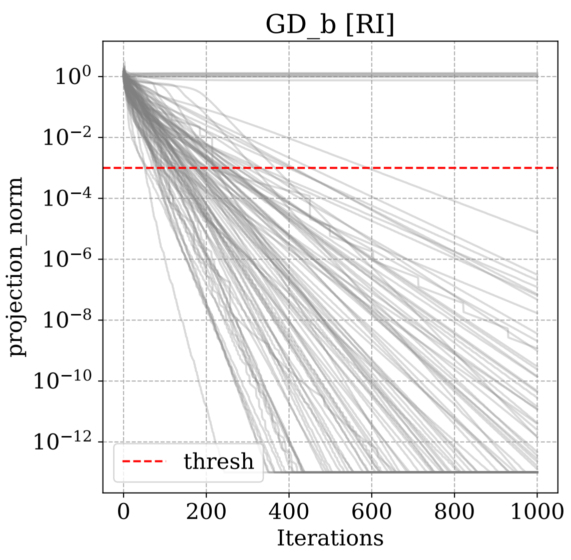
Let us provide numerical simulations to observe how Algorithm 3 can solve phase retrieval problem (1.2) for $n = 8$ and $m = 4n$ with Algorithm 4 in step 4. The results of numerical simulations are presented on Figure 4.1, where the metric $\text{dist}_{\text{norm}}$ between approximation $x^{(k)}$ and solution is drawn by gray lines for 100 problems. The main interest is to see the behavior of the algorithm with and without the presence of noise. The thing to pay attention is how the value of metric behaves when there is no possibility to improve an approximation due to the presence of noise in the model. For this goal, different solving tolerance is used: 0.001 for Figure 4.1a and 0.2 for Figure 4.1b. The second value was selected empirically from the behavior of the algorithm. It can be seen on Figure 4.1b that there is a limit in $\text{dist}_{\text{norm}}$ for all of the problems because of the presence of noise in the mathematical model. We observe no oscillations when the metric value reaches its limit, which indicates a sufficient robustness to the noise from this point of view. Also, in the case of no noise, Algorithm 3 with Algorithm 4 in step 4 can solve 95% of all problems and with noise $\sigma = 0.1$ it can solve 81% of all problems. Recall, that the problem is considered to be solved with respect to a Definition 1. Thus, there is difference in 14% of solved problems between noisy model and the model with no noise. Also, the starting point was selected randomly but the same for each experiment: with and without noise.

Note that in Algorithm 4 we require evaluation of f at each k-th step in order to check the stopping condition. It involves matrix multiplication at each step, which is extremely expensive for the current application. That is why, let us consider a Two-Point Step Size line search by Jonathan Barzilai and Jonathan Borwein [4]. This approach is based on the usage of secant equation for hessian approximation of the form αI where I is an identity matrix. According to [4] the step length α can be computed from the following two optimization problems

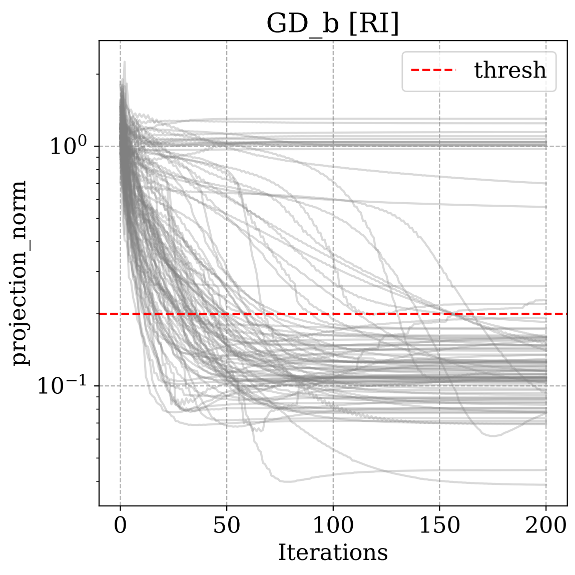
$$\min_{\alpha \in \mathbb{R}} \|s^{(k)} - \alpha y^{(k)}\|^2, \qquad \min_{\alpha \in \mathbb{R}} \|\alpha s^{(k)} - y^{(k)}\|^2, \tag{4.3}$$

where $s^{(k)} = x^{(k)} - x^{(k+1)}$ and $y^{(k)} = \nabla_x f(x^{(k)}) - \nabla_x f(x^{(k-1)})$. The solutions can be easily obtained and written as

$$\alpha = \frac{2\langle s^{(k)}, s^{(k)} \rangle}{\langle s^{(k)}, y^{(k)} \rangle + \langle y^{(k)}, s^{(k)} \rangle}, \tag{4.4}$$



(a)



(b)

Figure 4.1: (a) Metric $\text{dist}_{\text{norm}}$ between approximation $x^{(k)}$ computed by Algorithm 3 with line search Algorithm 4 and a solution for $n = 8$, $m = 32$, and $\sigma = 0$, and (b) for $\sigma = 0.1$.

and

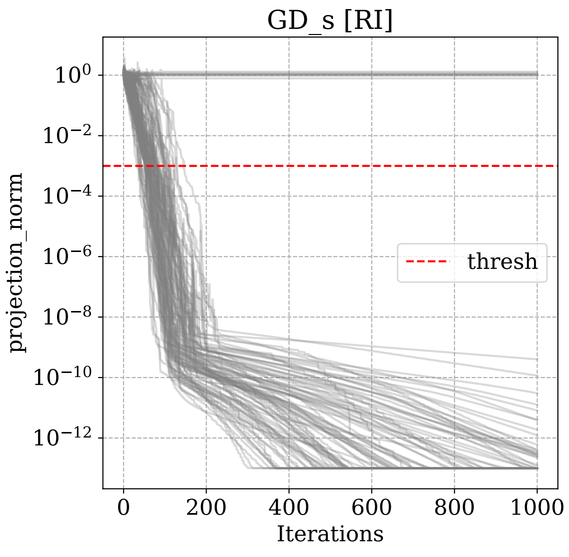
$$\alpha = \frac{\langle s^{(k)}, y^{(k)} \rangle + \langle y^{(k)}, s^{(k)} \rangle}{2\langle y^{(k)}, y^{(k)} \rangle}, \tag{4.5}$$

respectively. However, this step length can be applied just when $\langle s^{(k)}, y^{(k)} \rangle + \langle y^{(k)}, s^{(k)} \rangle > 0$ which is an analogue to the curvature condition generalized for the case of complex variable. The case of negative curvature is possible since the function to optimize is not convex and if it is detected then backtracking line search is used. Then, we can formalize gradient descent method with the secant based line search in the Algorithm 5.

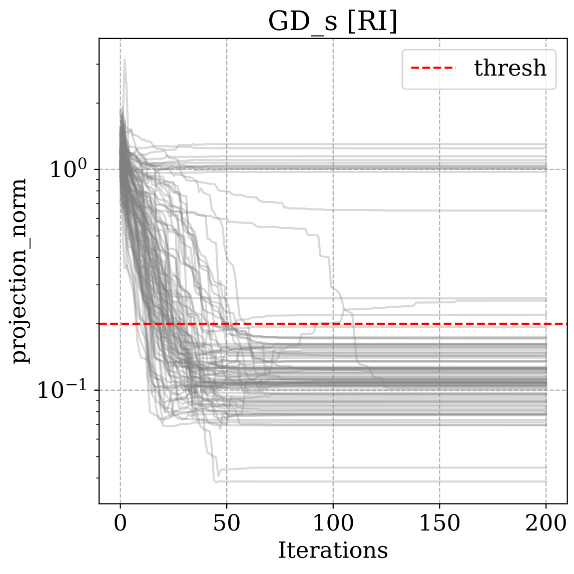
Algorithm 5: Gradient descent algorithm with secant based line search.

```
Input: Initial point $x_0 \in \mathbb{C}^n$, decrease rate $\rho \in (0,1)$, constant $c_1 \in (0,1)$.
Output: Approximate solution $x \in \mathbb{C}^n$ of (1.2).
1  $k = 0$;
2  Loop
3      $g_k = \nabla_x f(x_k)$;
4      if $k = 0$ or $\langle s^{(k)}, y^{(k)} \rangle + \langle y^{(k)}, s^{(k)} \rangle < 0$ then
5          compute $\alpha_k$ using Algorithm 4 ;        // use Armojo if negative curvature
6      else
7          compute $\alpha_k$ using (4.4) or (4.5)
8      $x^{(k+1)} = x^{(k)} - \alpha^{(k)} \nabla_x f(x^{(k)})$;
9      if stopping test is satisfied then
10         break
11     $k = k + 1$;
12 return: $x = x^{(k+1)}$
```

Let us provide numerical simulations to observe how Algorithm 5 can solve phase retrieval problem (1.2) for $n = 8$ and $m = 4n$ The results of numerical simulations are presented on Figure 4.2, where the meaning of curves are the same as for Figure 4.1. From Figure 4.2 it



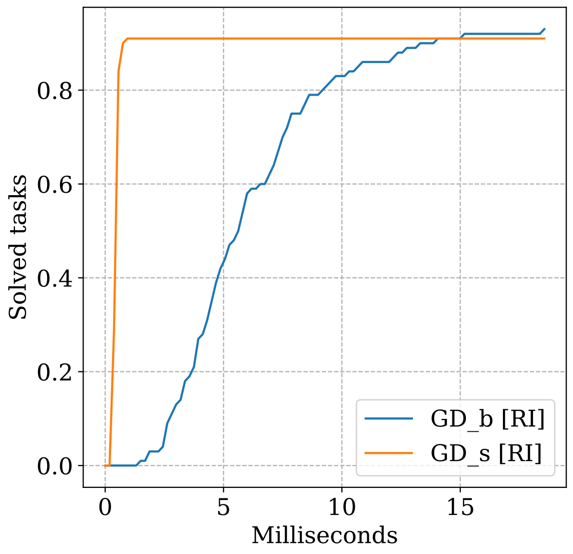
(a)



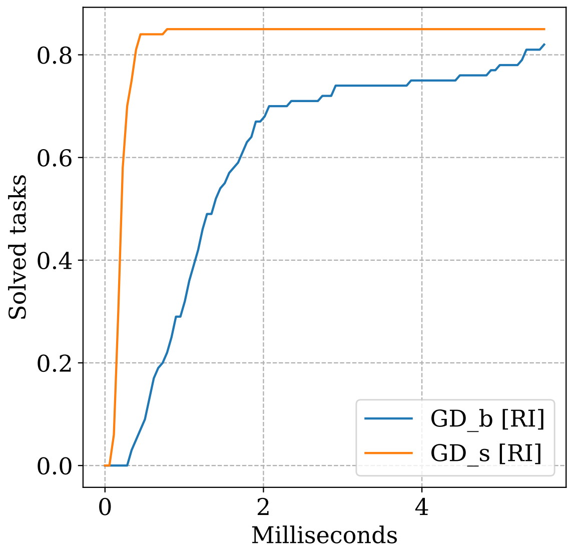
(b)

Figure 4.2: (a) Metric $\mathrm{dist}_{\mathrm{norm}}$ between approximation $x^{(k)}$ computed by Algorithm 5 and a solution for $n = 8$, $m = 32$, and $\sigma = 0$, and (b) for $\sigma = 0.1$.

is clear that Algorithm 5 performs faster than Algorithm 3 with Algorithm 4 in step 4. The number of problems that can be solved coincides. Also, we observe no oscillations when it reaches the limit in improvement. To conclude about required time to solve problems by these two algorithm let us build time profiles as it is explained in Section 2.5. It can be seen that in both cases, without noise and with noise, line search based on a secant equation is extremely faster than standard backtracking line search, which is also revealed on Figure 4.3.



(a)



(b)

Figure 4.3: (a) Profile for solved problems by selected time for 100 problems of size $n = 8$, $m = 32$, and $\sigma = 0$, and (b) for $\sigma = 0.1$.

4.3 Gauss-Newton Method

In this section we describe the Gauss-Newton method for solving the optimization problem

$$\min_{x \in \mathbb{C}^n} \quad f(x) := \tfrac{1}{2m} \| |Ax|^2 - b^2 \|^2, \tag{4.6}$$

which is clearly equivalent to (1.2). Following the general scheme of Gauss-Newton method [26], we denote a residual function $r : \mathbb{C}^n \to \mathbb{R}^m_+$ as

$$r(x) = |Ax|^2 - b^2, \tag{4.7}$$

and its jacobian matrix in terms of Wirtinger calculus (Observation 2) writes as

$$\nabla r(x) = \begin{pmatrix} A \odot \bar{A}\bar{x} & \bar{A} \odot Ax \end{pmatrix} \subset \mathbb{C}^{m \times 2n}. \tag{4.8}$$

Then, the descent direction $p \in \mathbb{C}^{2n}$ is a solution of the system

$$\nabla r(x)^* \nabla r(x) p = -\nabla r(x)^* r(x). \tag{4.9}$$

However, it can be shown (Observation 3) that the matrix $\nabla r(x)^* \nabla r(x)$ is always not a full rank matrix. Thus, the regularization is required which simply adds δI to $\nabla r(x)^* \nabla r(x)$, where $\delta > 0$. Typical value of δ depends on the data type that is used to represent numbers. For instance, if real and imaginary parts of complex numbers in $\nabla r(x)^* \nabla r(x)$ are represented as float numbers where each takes 32 bits in the memory, then δ must be greater than 10^{-8} to provide a regularization effect.

Observation 3 *Let $m > 2n$, then the rank of the jacobian matrix (4.8) is always less than $2n$ for any $x \in \mathbb{C}^n$.*

Proof. To show that it is enough to find a nonzero vector $v \in \mathbb{C}^n$ such that $\nabla r(x)v = 0$ for any $x \in \mathbb{C}^n$. Let $v = (x^\top, -\bar{x}^\top)^\top$ then

$$\begin{aligned}\begin{pmatrix} A \odot \bar{A}\bar{x} & \bar{A} \odot Ax \end{pmatrix} \begin{pmatrix} x \\ -\bar{x} \end{pmatrix} &= (A \odot \bar{A}\bar{x})x - (\bar{A} \odot Ax)\bar{x} \\ &= Ax \odot \bar{A}\bar{x} - \bar{A}\bar{x} \odot Ax \\ &= |Ax|^2 - |Ax|^2 = 0.\end{aligned}$$

The proof follows. □

Then, system (4.9) is replaced by

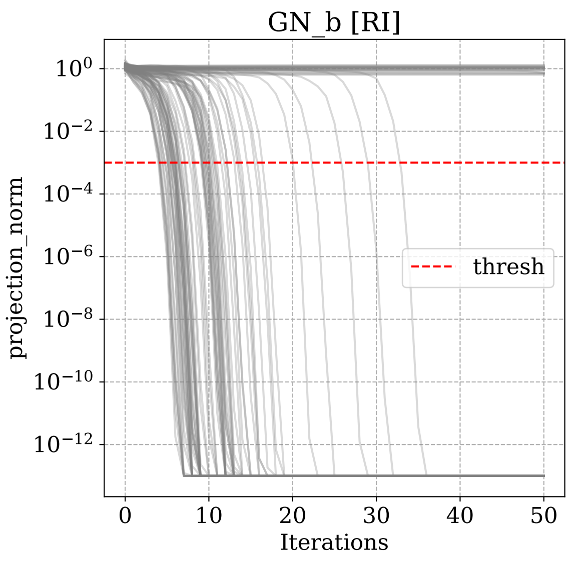
$$\big(\nabla r(x)^* \nabla r(x) + \delta I\big)p = -\nabla r(x)^* r(x), \tag{4.10}$$

where $\delta > 0$ and I is an identity matrix of size $2n$ by $2n$.

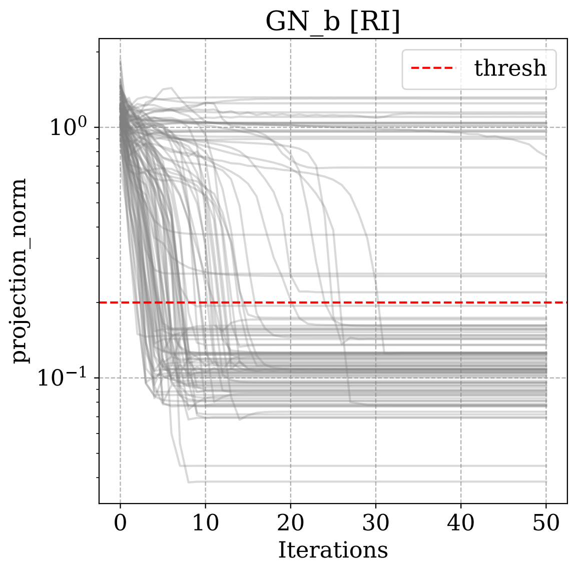
Algorithm 6: Gauss-Newton algorithm.

```
  Input: Initial point $x_0 \in \mathbb{C}^n$, regularization parameter $\delta > 0$.
  Output: Approximate solution $x \in \mathbb{C}^n$ of.
1 $x = x_0$;
2 Loop
3     find descent direction $p$ by solving (4.10);
4     compute step length $\alpha$ for descent direction $-p$ using a line search algorithm;
5     $x = x + \alpha p$;
6     if stopping test is satisfied then
7         break
8 return: $x$
```

Note, that in step 4 of the Algorithm 6 either backtracking or secant line search can be used. Let us provide numerical simulations to compare these two possibilities and to test the robustness property. The results are presented on Figure 4.4 and Figure 4.5, where the meaning of curves are the same as for Figure 4.1. The system (4.10) is solved by means of Cholesky decomposition. Recall, that the main interest is to see the behavior of the algorithm with and without the presence of noise. The thing to pay attention is how the value of metric behaves when there is no possibility to improve an approximation due to the presence of noise in the model.

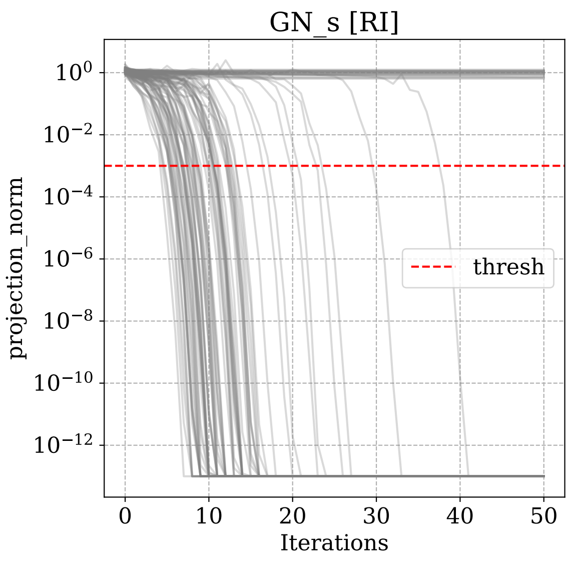


(a)

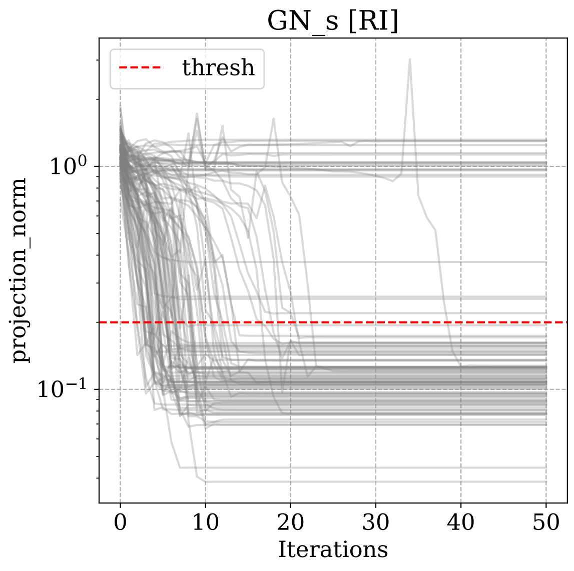


(b)

Figure 4.4: (a) Metric $\mathrm{dist}_{\mathrm{norm}}$ between approximation $x^{(k)}$ computed by Algorithm 6 with line search Algorithm 4 and a solution for $n = 8$, $m = 32$, and $\sigma = 0$, and (b) for $\sigma = 0.1$.



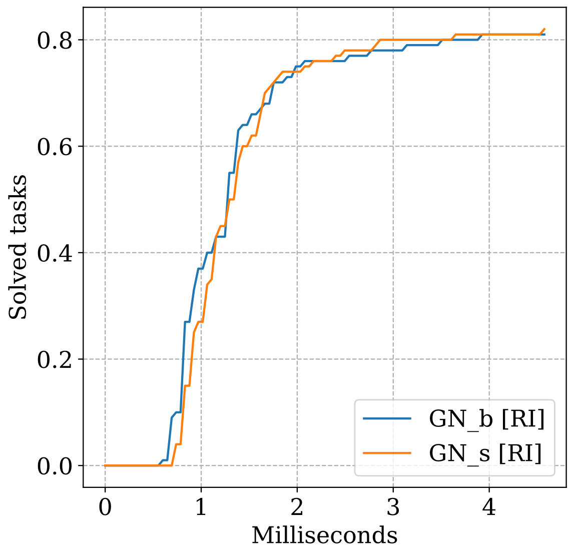
(a)



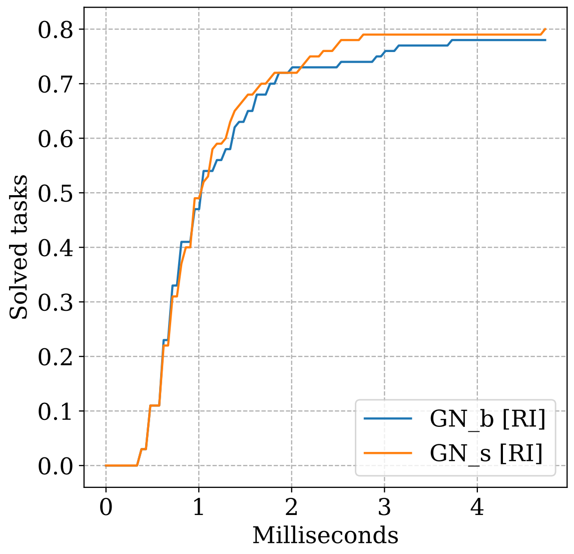
(b)

Figure 4.5: (a) Metric $\mathrm{dist}_{\mathrm{norm}}$ between approximation $x^{(k)}$ computed by Algorithm 6 with line search (4.4) and a solution for $n = 8$, $m = 32$, and $\sigma = 0$, and (b) for $\sigma = 0.1$.

From Figure 4.4 and Figure 4.5 we can conclude that both line search strategies are equivalent since the same behavior and the same number of solved problems ($\approx 80\%$) are observed. In addition, no oscillations are found when the algorithm reaches a limit of approximation in the noisy case. Also, from Figure 4.6 we can conclude that both algorithms perform with the same computational speed.



(a)



(b)

Figure 4.6: (a) Profile for solved problems by selected time for 100 problems of size $n = 8$, $m = 32$, and $\sigma = 0$, and (b) for $\sigma = 0.1$. GN_b and GN_s means Gauss-Newton with backtracking and secant line search respectively.

4.4 Semidefinite Programming Method

In this section, we consider the convex relaxation of (1.2), which is called PhaseLift and reformulates the original system of nonlinear equations as a semidefinite programming problem (Candes et al. [8]) The reformulation is based on a fact that the intensity measurements in (1.2) can be written as

$$|a_j^* x|^2 = \operatorname{tr}(x^* a_j a_j^* x) = \operatorname{tr}(a_j a_j^* x x^*) = \operatorname{tr}(A_j X), \tag{4.11}$$

where A_j is a rank-one matrix $a_j a_j^*$. Let $\mathcal{A}(X) = \{\operatorname{tr}(A_j X) : j \in \{1, \dots, m\}\}$, then the problem (1.2) can be written as

$$\begin{array}{ll} \text{find} & X \\ \text{s.t.} & \mathcal{A}(X) = b \\ & X \succeq 0 \\ & \operatorname{rank}(X) = 1, \end{array} \tag{4.12}$$

or equivalently

$$\begin{array}{ll} \min_{X \in \mathbb{C}^{n\times n}} & \operatorname{rank}(X) \\ \text{s.t.} & \mathcal{A}(X) = b \\ & X \succeq 0, \end{array} \tag{4.13}$$

which appears to be a rank minimization problem. The problem (4.13) is NP hard, and thus it was proposed in [8] to use the trace norm as a convex surrogate [5, 24] for the rank functional, which gives the following SDP problem

$$\begin{array}{ll} \min_{X \in \mathbb{C}^{n\times n}} & \operatorname{tr}(X) \\ \text{s.t.} & \mathcal{A}(X) = b \\ & X \succeq 0. \end{array} \tag{4.14}$$

To recover an original solution x the leading eigenvector of X can be used. Recall, that one of the main requirements to the phase retrieval algorithm defined in Chapter 1 is an efficiency. The number of parameters in reformulation (4.14) is n^2, where n is the number of laser beams. Since in practice, the number of beams can be up to 100, this method will not be considered further in this work due to obvious inconsistency with the efficiency requirement.

4.5 Alternating Projections Method

In this section, we present the numerical solution of (1.2) by means of the alternating projection algorithm. Let us first reformulate equation (1.2) under the equivalent form

$$Ax = y \quad |y| = b, \tag{4.15}$$

where the unknown is the couple $(x, y) \in \mathbb{C}^n \times \mathbb{C}^m$. Obviously, x is a solution of (1.2) if and only if $(x, b \odot e^{i\arg(Ax)})$ is a solution of (4.15).

The system of equations (4.15) can be reformulated as a minimization of a quadratic convex function subject to a nonconvex set of equality constraints:

$$\begin{array}{ll} \min_{(x,y)\in\mathbb{C}^n\times\mathbb{C}^m} & \frac{1}{2}\|Ax - y\|^2 \\ \text{s.t.} & |y| = b. \end{array} \tag{4.16}$$

We have the following elementary properties.

Proposition 4 *Let x be fixed vector in (4.16), then the optimization problem is separable with respect to y_j for $j \in \{1, \ldots, m\}$.*

Proof. Let α_j denotes the j-th component of Ax, and b_j denotes the j-th component of b. Then (4.16) writes as

$$\begin{array}{ll} \min_{y\in\mathbb{C}^m} & \frac{1}{2}\sum_{j=1}^m |\alpha_j - y_j|^2 \\ \text{s.t.} & |y| = b. \end{array} \tag{4.17}$$

Using the fact that the constrains are defined for each variable y_j separately we can conclude that the minimization of a sum in (4.17) is equivalent to m minimization problems of the form

$$\begin{array}{ll} \min_{y_j\in\mathbb{C}} & |\alpha_j - y_j|^2 \\ \text{s.t.} & |y_j| = b_j, \end{array} \tag{4.18}$$

for $j \in \{1, \ldots, m\}$ □

Proposition 5 *$x \in \mathbb{C}^n$ is a solution of (1.2) if and only if $(x, b \odot e^{i\arg(Ax)})$ is an optimal solution of (4.16). In addition, when $y \in \mathbb{C}^m$ is fixed, (4.16) is a linear least squares problem, those optimal solution is $x = A^\dagger y$. When $x \in \mathbb{C}^n$ is fixed, the optimal solution of (4.16) is the orthogonal projection of Ax on the set of constraints and is given by $y = b \odot e^{i\arg(Ax)}$.*

Proof. The first assertion follows from the equivalence between (1.2) and (4.15).

When y is fixed in (4.16), the constraints disappears and thus the problem is reduced to a linear least squares problem.

When x is fixed in (4.16), following the Proposition 4, the problem is separable and is reduced to the solution of m one dimensional optimization problems of the form (4.18) From the triangle inequality, we know that for all $\theta \in \mathbb{R}$, $|\alpha_j - b_j e^{i\theta}| \geq |\,|\alpha_j| - b_j|$, with an equality if and only if $\theta = \arg(\alpha_j)$ or $\alpha_j = 0$ or $b_j = 0$. The optimal y follows. □

Proposition 6 *For all $(x, y) \in \mathbb{C}^n \times \mathbb{C}^n$, we have $\||x| - |y|\| \leq \|x - y\|$, with an equality if and only if for all $k \in \{1, \dots, n\}$, $x^{(k)} = 0$ or $y^{(k)} = 0$ or $\arg(x^{(k)}) = \arg(y^{(k)})$.*

Proof. Let $(x, y) \in \mathbb{C}^n \times \mathbb{C}^n$. Using the fact that $\|x\| = \||x|\|$, we have

$$\||x| - |y|\|^2 = \|x\|^2 - 2\langle |x|, |y|\rangle + \|y\|^2$$

and

$$\|x - y\|^2 = \|x\|^2 - 2\operatorname{Re}(\langle x, y\rangle) + \|y\|^2$$

We have $\operatorname{Re}(\langle x, y\rangle) = \sum_{k=1}^n |x^{(k)}||y^{(k)}| \cos(\arg(y^{(k)}) - \arg(x^{(k)}))$

$$\begin{aligned}\operatorname{Re}(\langle x, y\rangle) &= \sum_{k=1}^n |x^{(k)}||y^{(k)}| \cos(\arg(y^{(k)}) - \arg(x^{(k)})) \\ &\leq \sum_{k=1}^n |x^{(k)}||y^{(k)}| \\ &= \langle |x|, |y|\rangle,\end{aligned}$$

with an equality if and only if $\sum_{k=1}^n |x^{(k)}||y^{(k)}|(1 - \cos(\arg(y^{(k)}) - \arg(x^{(k)}))) = 0$. Because each element of the sum is nonnegative, this occurs if and only only one of the modulus or the cosine is zero for all k. □

Unfortunately, the problem (4.16) is not convex. To handle this difficulty, a well-known approach, suggested by Proposition 5, is to alternatively minimize in x and in y. This leads to Algorithm 7. This algorithm is known as *alternating minimization* or *alternating projection* algorithm (see, e.g., [25, 23]). It is also sometimes referred as the Gerchberg–Saxton algorithm (see, e.g., [45]), while the original Gerchberg–Saxton algorithm [14] is when the matrix A corresponds to a discrete Fourier transform (DFT) and the couple of iterates is defined by means of the recurrence formulas $y^{(k+1)} = b \odot e^{i \arg(Ax^{(k)})}$ and $x^{(k+1)} = a \odot e^{i \arg(A^\dagger y^{(k+1)})}$, where a is the known vector of amplitudes in the initial space, see e.g., [32]. Algorithm 7 shares a

Algorithm 7: Alternating projection algorithm.

```
Input: Initial point $x^{(0)} \in \mathbb{C}^n$
Output: Approximate solution $x \in \mathbb{C}^n$ of (1.2).
1 $k = 0$;
2 Loop
3     $y^{(k+1)} = b \odot \exp(i \arg(Ax^{(k)}))$;
4     $x^{(k+1)} = A^\dagger y^{(k+1)}$;
5     $k = k + 1$;
6     if stopping test is satisfied then
7         break
8 return: $x = x^{(k)}$.
```

common property with the original Gerchberg–Saxton algorithm, which is the reduction of the norm of the residual of (1.2) [12]. Unfortunately, the norm of the residual in Proposition 7 is nonincreasing, which means that Algorithm 7 is not guaranteed to converge to a global minimum of (4.16).

Proposition 7 *At each iteration $k \geq 0$ of Algorithm 7, we have*

$$\|Ax^{(k+1)} - y^{(k+1)}\| \leq \|Ax^{(k)} - y^{(k)}\| \quad \textit{and} \quad \||Ax^{(k+1)}| - b\| \leq \||Ax^{(k)}| - b\|.$$

Proof. Let $k \in \mathbb{N}$ be the iteration number. By using the fact that $x^{(k+1)}$ is a minimum of problem (4.16) with $y = y^{(k+1)}$ and the fact that $y^{(k+1)}$ is a minimum of problem (4.16) with $x = x^{(k)}$, we have

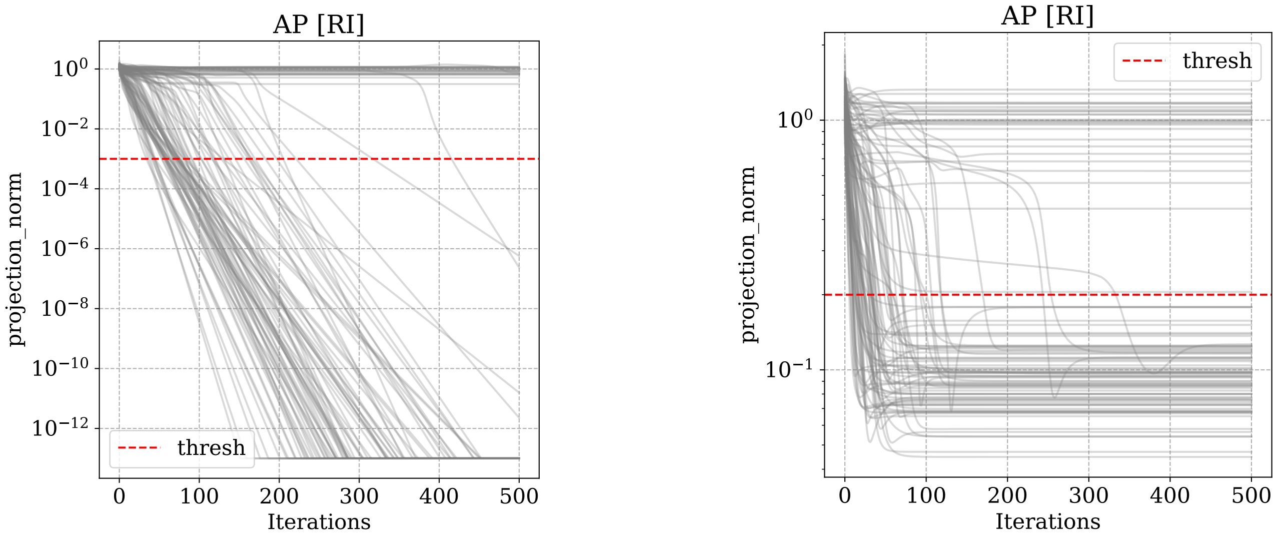
$$\begin{aligned} \|Ax^{(k+1)} - y^{(k+1)}\| &\leq \|Ax^{(k)} - y^{(k+1)}\| \\ &\leq \|Ax^{(k)} - y^{(k)}\|, \end{aligned} \tag{4.19}$$

which proves the first inequality. To prove the second inequality, use Proposition 6 and (4.19), then apply again Proposition 6 and the fact that $\arg(y^{(k+1)}) = \arg(Ax^{(k)})$, so that

$$\begin{aligned} \||Ax^{(k+1)}| - b\| &= \||Ax^{(k+1)}| - |y^{(k+1)}|\| \\ &\leq \|Ax^{(k+1)} - y^{(k+1)}\| \\ &\leq \|Ax^{(k)} - y^{(k+1)}\| \\ &= \||Ax^{(k)}| - |y^{(k+1)}|\| \\ &= \||Ax^{(k)}| - b\|. \end{aligned}$$

$\square$

Let us provide numerical simulations to test the robustness of the Algorithm 7. Recall, that the main interest is to see the behavior of the algorithm with and without the presence of noise. The thing to pay attention is how the value of metric behaves when there is no possibility to improve an approximation due to the presence of noise in the model. The results are presented on Figure 4.7, where the meaning of curves are the same as for Figure 4.1. From Figure 4.7 we



(a) (b)

Figure 4.7: (a) Metric $\text{dist}_{\text{norm}}$ between approximation $x^{(k)}$ computed by Algorithm 7 and a solution for $n = 8$, $m = 32$, and $\sigma = 0$, and (b) for $\sigma = 0.1$.

can conclude that no oscillations are found when the algorithm reaches a limit of approximation in the noisy case. Also, approximately the same number of problems can be solved in both cases: 78% when there is no noise and 74% with the presence of noise.

4.6 Alternating Directions Method of Multipliers

In this section, we present the numerical solution of (1.2) by means of the Alternating Directions Method of Multipliers. Let us first reformulate equation (1.2) under the equivalent form

$$y = z \quad y \in \operatorname{range}(A),\ z \in \mathcal{M}_b, \tag{4.20}$$

where $\mathcal{M}_b = \{z \in \mathbb{C}^m : |z| = b\}$ and the unknown is the couple $(y, z) \in \mathbb{C}^n \times \mathbb{C}^m$. Obviously, x is a solution of (1.2) if and only if (Ax, Ax) is a solution of (4.20).

The system of equations (4.20) can be reformulated as an artificial minimization problem of a zero function subject to set and equality constraints:

$$\begin{array}{ll} \min_{(y,z)\in\mathbb{C}^m\times\mathbb{C}^m} & 0 \\ \text{s.t.} & y - z = 0, \\ & y \in \operatorname{range}(A),\ z \in \mathcal{M}_b \end{array} \tag{4.21}$$

Let us define the update formulas for the Altenating Directions Method of Multipliers which is applied to (4.21).

Proposition 8 *Let us consider the minimization problem (4.21). Then the Altenating Directions Method of Multipliers update formulas are*

$$\begin{aligned} z^{(k+1)} &= b \odot \exp\left(i \arg(y^{(k)} + \lambda^{(k)})\right), \\ y^{(k+1)} &= AA^{\dagger}(z^{(k+1)} - \lambda^{(k)}), \\ \lambda^{(k+1)} &= \lambda^{(k)} + y^{(k+1)} - z^{(k+1)}, \end{aligned}$$

where $A^{\dagger} = (A^*A)^{-1}A^*$ is a pseudo-inverse of matrix $A \in \mathbb{C}^{m\times n}$.

Proof. The scaled augmented Lagrangian writes as

$$\begin{aligned} \mathcal{L}_\rho(y, z, \hat{\lambda}) &= \operatorname{Re}\{\hat{\lambda}^*(y - z)\} + \frac{\rho}{2}\|y - z\|^2 \\ &= \frac{\rho}{2}\|y - z + \lambda\|^2 - \frac{\rho}{2}\|\lambda\|^2, \end{aligned}$$

where $\lambda = \frac{\hat{\lambda}}{\rho}$ is a scaled dual variable. Then, we conclude that

1. The update formula for z with fixed $y = y^{(k)}$ and $\lambda = \lambda^{(k)}$ writes as

$$\begin{aligned} z^{(k+1)} &= \operatorname{argmin}_{z\in\mathcal{M}_b} \frac{\rho}{2}\|y^{(k)} - z + \lambda^{(k)}\|^2 \\ &= b \odot \exp\left(i \cdot \operatorname{argmin}_{\theta\in\mathbb{R}^m} \frac{\rho}{2}\|y^{(k)} - b \odot e^{i\theta} + \lambda^{(k)}\|\right) \\ &= b \odot \exp\left(i \cdot \arg(y^{(k)} + \lambda^{(k)})\right). \end{aligned}$$

2. The update formula for y with fixed $z = z^{(k+1)}$, $\lambda = \lambda^{(k)}$ writes as
$$\begin{aligned} y^{(k+1)} &= \operatorname{argmin}_{y \in \operatorname{range}(A)} \frac{\rho}{2} \|y - z^{(k+1)} + \lambda^{(k)}\|^2 \\ &= A\Big(\operatorname{argmin}_{x \in \mathbb{C}^n} \frac{\rho}{2} \|Ax - z^{(k+1)} + \lambda^{(k)}\|^2\Big) \\ &= AA^{\dagger}(z^{(k+1)} - \lambda^{(k)}). \end{aligned}$$

3. The update formula for λ with fixed $z = z^{(k+1)}$ and $y = y^{(k+1)}$ writes as
$$\lambda^{(k+1)} = \lambda^{(k)} + y^{(k+1)} - z^{(k+1)}.$$

□

The Observation 4 is used for simplification the update formula for $y^{(k+1)}$.

Observation 4 *Let us consider the update formulas in Proposition 8. If $A^*\lambda^{(0)} = 0$ then*
$$A^*\lambda^{(k)} = 0 \textit{ for any } k \in \mathbb{N}\backslash\{0\}.$$

Proof. Let us prove the result by induction reasoning. By assumption the result is true for $k = 0$. Suppose that $A^*x^{(k)} = 0$ for $k \in \mathbb{N}$. We have $\lambda^{(k+1)} = \lambda^{(k)} + AA^{\dagger}z^{(k+1)} - z^{(k+1)}$ we have that $A^*\lambda^{(k+1)} = A^*\lambda^{(k)} + A^*z^{(k+1)} - A^*z^{(k+1)}$ and thus $A^*\lambda^{(k+1)} = 0$. □

Algorithm 8: Alternating Directions Method of Multipliers.

```
   Input: Initial point $x^{(0)} \in \mathbb{C}^n$ and parameter $\lambda^{(0)} \in \mathbb{C}^m$
   Output: Approximate solution $x \in \mathbb{C}^n$ of (1.2).
1  $k = 0$;
2  $y_0 = Ax^{(0)}$;
3  Loop
4      $z^{(k+1)} = b \odot \exp(i \arg(y^{(k)} + \lambda^{(k)}))$;
5      $x^{(k+1)} = A^{\dagger}z^{(k+1)}$;
6      $y^{(k+1)} = Ax^{(k+1)}$;
7      if stopping test is satisfied then
8          break
9      $\lambda^{(k+1)} = \lambda^{(k)} + y^{(k+1)} - z^{(k+1)}$;
10     $k = k + 1$;
11 return: $x = x^{(k)}$.
```

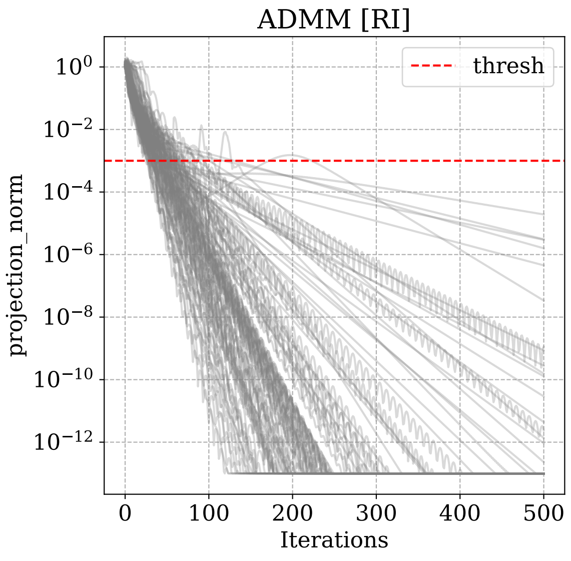
Proposition 9 *Let $(y, z, \lambda) \in \mathbb{C}^m \times \mathbb{C}^m \times \mathbb{C}^m$ be a limit point of a sequence defined by update formulas in Proposition 8 and $A^*\lambda^{(0)} = 0$, then (y, z) is a global solution of (4.21).*

Proof. Let $(y, z, \lambda) \in \mathbb{C}^m \times \mathbb{C}^m \times \mathbb{C}^m$ be a limit point of a sequence defined by update formulas in Proposition 8 exists and $A^*\lambda^{(0)} = 0$. Then, replacing y by $AA^{\dagger}z$ we have that
$$z = b \odot \exp\big(i \cdot \arg(AA^{\dagger}z + \lambda)\big), \quad \lambda = \lambda + (AA^{\dagger} - I)z.$$
We can immediately deduce that $z \in \mathcal{M}_b$ and $(AA^{\dagger} - I)z = 0$. The matrix $AA^{\dagger} - I$ is a projection matrix on the kernel of A^*. The projection is zero only if $z \in \operatorname{range}(A)$. □

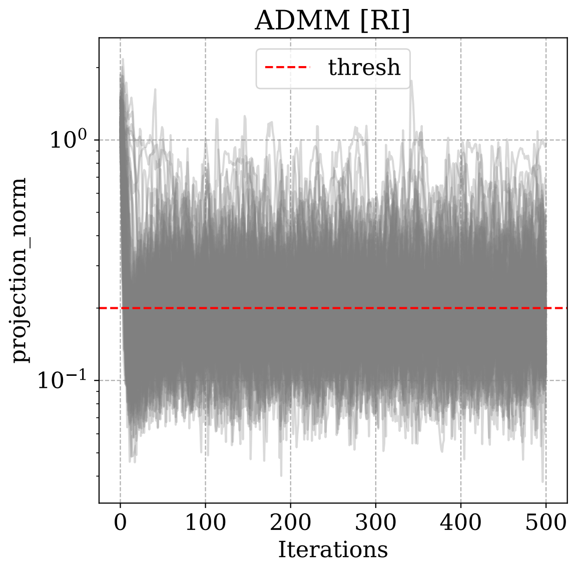
Remark 1 *By Proposition 9 and Proposition 8 we conclude that if the Alernating Directions Method of Multipliers converges, then it always returns a global solution of (4.21) and so (1.2). This is an important property which the alternating projections algorithm does not have [45].*

In practice, it is possible that there is no exact solution of (1.2) because of noisy measurements b and inaccurate transmission matrix A. In context of problem (4.21), it means that $\mathrm{range}(A) \cap \mathcal{M}_b = \emptyset$. Note, that this leads to unpleasant consequence for update formula of dual multipliers λ in Proposition 8. It is clear that if $\mathrm{range}(A) \cap \mathcal{M}_b = \emptyset$, then there are no $y \in \mathrm{range}(A)$ and $z \in \mathcal{M}_b$ such that $y - z = 0$ and thus no limit point for λ.

Let us provide numerical simulations to demonstrate this fact. On Figure 4.8 it can be clearly seen that when there is noise in the model with $\sigma = 0.1$, then Algorithm 8 does not converge. However, when no noise is added to the model, then all of 100 problems can be solved by means of Algorithm 8 despite the fact of extensive oscillations during convergence. This fact has been never observed for previous methods discussed in this chapter. Since the presence of



(a)



(b)

Figure 4.8: (a) Metric $\mathrm{dist}_{\mathrm{norm}}$ between approximation $x^{(k)}$ computed by Algorithm 8 and a solution for $n = 8$, $m = 32$, and $\sigma = 0$, and (b) for $\sigma = 0.1$.

noise is true for the applications that are considered in this work, it is necessary to modify the algorithm in order to work with noisy data.

Let us consider the relaxed version of (4.21) where we introduce a new variable $\xi \in \mathbb{C}^m$ to be able able to capture the case when $\mathrm{range}(A) \cap \mathcal{M}_b = \emptyset$.

$$\begin{array}{ll} \min_{(y,z,\xi) \in \mathbb{C}^m \times \mathbb{C}^m \times \mathbb{C}^m} & \frac{1}{2}\|\xi\|^2 \\ \text{s.t.} & y - z = \xi \\ & y \in \mathrm{range}(A),\ z \in \mathcal{M}_b \end{array} \tag{4.22}$$

Let us define the update formulas for the Altenating Directions Method of Multipliers which is applied to (4.22).

Proposition 10 *Let us consider the minimization problem (4.22). Then the Altenating Directions Method of Multipliers update formulas are*

$$\begin{aligned} z^{(k+1)} &= b \odot \exp\big(i \arg(y^{(k)} - \xi^{(k)} + \lambda^{(k)})\big), \\ y^{(k+1)} &= AA^{\dagger}(z^{(k+1)} + \xi^{(k)} - \lambda^{(k)}), \\ \xi^{(k+1)} &= \frac{\rho}{1+\rho}\Big(y^{(k+1)} - z^{(k+1)} + \lambda^{(k)}\Big), \\ \lambda^{(k+1)} &= \lambda^{(k)} + y^{(k+1)} - z^{(k+1)} - \xi^{(k+1)}, \end{aligned}$$

where $A^{\dagger} = (A^*A)^{-1}A^*$ is a pseudo-inverse of matrix $A \in \mathbb{C}^{m\times n}$.

Proof. The scaled augmented Lagrangian writes as

$$\begin{aligned} \mathcal{L}_\rho(y, z, \xi, \hat{\lambda}) &= \frac{1}{2}\|\xi\|^2 + \mathrm{Re}\{\hat{\lambda}^*(y - z - \xi)\} + \frac{\rho}{2}\|y - z - \xi\|^2 \\ &= \frac{1}{2}\|\xi\|^2 + \frac{\rho}{2}\|y - z - \xi + \lambda\|^2 - \frac{\rho}{2}\|\lambda\|^2, \end{aligned}$$

where $\lambda = \frac{\hat{\lambda}}{\rho}$ is a scaled dual variable. Then, we conclude that

1. The update formula for z with fixed $y = y^{(k)}$, $\xi = \xi^{(k)}$, and $\lambda = \lambda^{(k)}$ writes as

$$\begin{aligned} z^{(k+1)} &= \mathrm{argmin}_{z\in\mathcal{M}_b} \frac{\rho}{2}\|y^{(k)} - z - \xi^{(k)} + \lambda^{(k)}\|^2 \\ &= b \odot \exp\left(i \cdot \mathrm{argmin}_{\theta\in\mathbb{R}^m} \frac{\rho}{2}\|y^{(k)} - b\odot e^{i\theta} - \xi^{(k)} + \lambda^{(k)}\|\right) \\ &= b \odot \exp\big(i \cdot \arg(y^{(k)} - \xi^{(k)} + \lambda^{(k)})\big). \end{aligned}$$

2. The update formula for y with fixed $z = z^{(k+1)}$, $\xi = \xi^{(k)}$, and $\lambda = \lambda^{(k)}$ writes as

$$\begin{aligned} y^{(k+1)} &= \mathrm{argmin}_{y\in\mathrm{range}(A)} \frac{\rho}{2}\|y - z^{(k+1)} - \xi^{(k)} + \lambda^{(k)}\|^2 \\ &= A\Big(\mathrm{argmin}_{x\in\mathbb{C}^n} \frac{\rho}{2}\|Ax - z^{(k+1)} - \xi^{(k)} + \lambda^{(k)}\|^2\Big) \\ &= AA^{\dagger}(z^{(k+1)} + \xi^{(k)} - \lambda^{(k)}). \end{aligned}$$

3. The update formula for ξ with fixed $z = z^{(k+1)}$, $y = y^{(k+1)}$ and $\lambda = \lambda^{(k)}$ is obtained from

$$\partial_\xi \mathcal{L}_\rho(y, z, \xi, \lambda) = \xi - \rho(y - z - \xi + \lambda) = 0,$$

from where we have

$$\xi^{(k+1)} = \frac{\rho}{1+\rho}\Big(y^{(k+1)} - z^{(k+1)} + \lambda^{(k)}\Big).$$

4. The update formula for λ with fixed $z = z^{(k+1)}$, $y = y^{(k+1)}$, and $\xi = \xi^{(k+1)}$ writes as

$$\lambda^{(k+1)} = \lambda^{(k)} + y^{(k+1)} - z^{(k+1)} - \xi^{(k+1)}.$$

□

Observation 5 *Let us consider the update formulas in Proposition 10. If $\lambda^{(0)} \in \ker(A^*)$ then*

$$\lambda^{(k)}, \xi^{(k)} \in \ker(A^*) \text{ for all } k \in \mathbb{N}\backslash\{0\}.$$

Furthermore,

$$\rho\lambda^{(k)} = \xi^{(k)}, \text{ for all } k \in \mathbb{N}\backslash\{0\}.$$

Proof. First, note that

$$\begin{aligned}\lambda^{(k+1)} &= \lambda^{(k)} + y^{(k+1)} - z^{(k+1)} - \xi^{(k+1)} \\ &= \lambda^{(k)} + y^{(k+1)} - z^{(k+1)} - \frac{\rho}{1+\rho}\left(y^{(k+1)} - z^{(k+1)} + \lambda^{(k)}\right) \\ &= \frac{1}{1+\rho}\left(y^{(k+1)} - z^{(k+1)} + \lambda^{(k)}\right) \\ &= \frac{1}{\rho}\xi^{(k+1)}.\end{aligned}$$

Then, $\lambda^{(k+1)} = \lambda^{(k)} + AA^\dagger(z^{(k+1)} + \rho\lambda^{(k)} - \lambda^{(k)}) - z^{(k+1)} - \rho\lambda^{(k+1)}$ from where

$$\begin{aligned}\lambda^{(k+1)} &= \frac{1}{1+\rho}\left(\lambda^{(k)} + AA^\dagger\big(z^{(k+1)} + (\rho-1)\lambda^{(k)}\big) - z^{(k+1)}\right) \\ &= \frac{1}{1+\rho}\left(\lambda^{(k)} + (\rho-1)AA^\dagger\lambda^{(k)} + (AA^\dagger - I)z^{(k+1)}\right)\end{aligned}$$

Now, let $\lambda^{(k)} \in \ker(A^*)$, from previous equations it is clear that $\lambda^{(k+1)} \in \ker(A^*)$ since $AA^\dagger - I$ is a projection matrix on $\ker(A^*)$. And thus, from the first equation we obtain that $\xi^{(k+1)} \in \ker(A^*)$ also. □

The Observation 5 is used for simplification the update formula for $y^{(k+1)}$ and for elimination variable ξ.

The main question for Algorithm 9 is how to select ρ properly. Before going further, let us provide numerical simulations for $\rho = 0.5$. On Figure 4.9 it can be seen that the problem with convergence for noisy case is fixed, where the oscillations during the convergence for the problems without noise in the model still exists but with lower frequency. It is also important to note that not all problems are solved by Algorithm 9 when $\rho = 0.5$. It appears that $\rho = 0$ is useful when an approximation $x^{(k)}$ is far from the solution, but then it must be changed to some positive value.

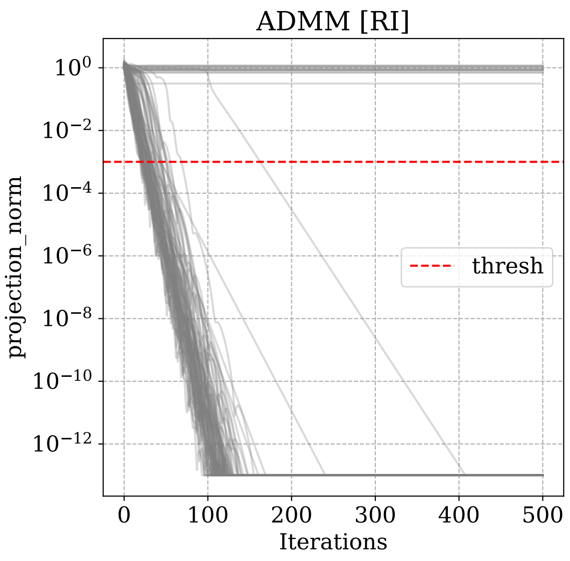
Let us provide the following observations which helps to understand what strategy for ρ is efficient.

Observation 6 *If $\rho = 1$, in Algorithm 9, then it reduces to Algorithm 7 starting from the second iteration.*

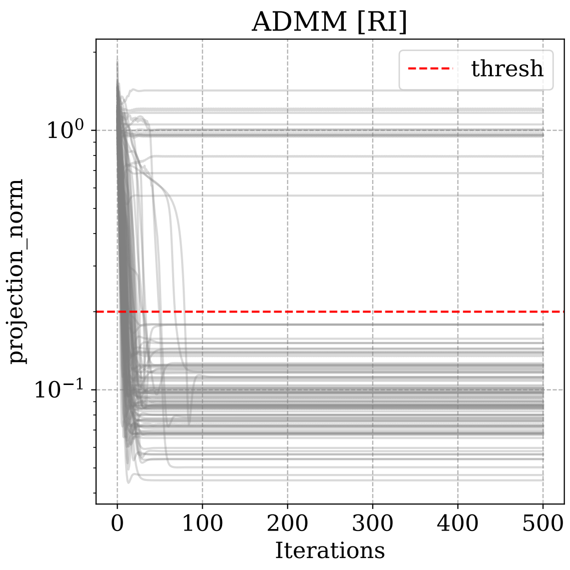
Proof. Follows from the second property of Observation 5. □

Algorithm 9: Relaxed Alternating Directions Method of Multipliers.

```
   Input: Initial point $x^{(0)} \in \mathbb{C}^n$ and parameters $\rho > 0$, $\lambda^{(0)}, \xi^{(0)} \in \mathbb{C}^m$.
   Output: Approximate solution $x \in \mathbb{C}^n$ of (1.2).
1  $k = 0$, $y_0 = Ax^{(0)}$;
2  Loop
3      $z^{(k+1)} = b \odot \exp\left(i \arg(y^{(k)} + (1-\rho)\lambda^{(k)})\right)$;
4      $x^{(k+1)} = A^\dagger z^{(k+1)}$;
5      $y^{(k+1)} = Ax^{(k+1)}$;
6      if stopping test is satisfied then
7          break
8      $\lambda^{(k+1)} = \frac{1}{1+\rho}\left(\lambda^{(k)} + y^{(k+1)} - z^{(k+1)}\right)$;
9      $k = k + 1$;
10 return: $x = x^{(k)}$.
```



(a)



(b)

Figure 4.9: (a) Metric $\mathrm{dist}_{\mathrm{norm}}$ between approximation $x^{(k)}$ computed by Algorithm 9 and a solution for $n = 8$, $m = 32$, $\rho = 0.5$, and $\sigma = 0$, and (b) for $\sigma = 0.1$.

Observation 7 *If $\rho = 0$, $\xi^{(0)} = 0$ in Algorithm 9 then it reduces to Algorithm 8.*

Proposition 11 *Let $(y, z, \xi, \lambda) \in \mathbb{C}^m \times \mathbb{C}^m \times \mathbb{C}^m \times \mathbb{C}^m$ be a limit point of a sequence defined by update formulas in Proposition 10 and $\lambda^{(0)} \in \ker(A^*)$, $\rho > 0$, then*

$$y - z \in \ker(A^*) \cap \{\alpha z, \alpha \in (-\rho, \infty)^m\}.$$

Proof. Let $(y, z, \xi, \lambda) \in \mathbb{C}^m \times \mathbb{C}^m \times \mathbb{C}^m \times \mathbb{C}^m$ be a limit point of a sequence defined by update formulas in Proposition 10 exists and $\lambda^{(0)} \in \ker(A^*)$, $\rho > 0$. Then, from Observation 5 we obtain that $\lambda = \frac{1}{1+\rho}\left(\lambda + (AA^\dagger - I)z\right)$, $y = AA^\dagger z$, and $\rho\lambda = \xi$ which gives that $y - z = \rho\lambda$.

Also, this can be equivalently written as $AA^\dagger z - z = \xi$. The last equation is used to obtain

$$\begin{aligned} z &= b \odot \exp\big(i \cdot \arg(AA^\dagger z - \xi + \lambda)\big) \\ &= b \odot \exp\big(i \cdot \arg(AA^\dagger z - AA^\dagger z + z + \lambda)\big) \\ &= b \odot \exp\big(i \cdot \arg(z + \lambda)\big), \end{aligned}$$

which gives that $\lambda \in \{\alpha z, \alpha \in (-1,\infty)^m\}$ and thus $y - z \in \operatorname{range}(A)^\perp \cap \{\alpha z, \alpha \in (-\rho,\infty)^m\}$. □

Observation 8 *Let us consider a recurrent form* $x^{(k+1)} = \frac{1}{2}(x_k + \Delta)$ *for some starting* $x^{(0)} \in \mathbb{R}^n$ *and fixed* $\Delta \in \mathbb{R}^n$. *Then* $\lim_{k\to\infty} x^{(k)} = \Delta$.

Proof. The proof follows from the explicit form $x^{(k+1)} = \dfrac{1}{2^{k+1}}\big(x^{(0)} + \Delta\big) + \Delta \displaystyle\sum_{i=1}^{k} \frac{1}{2^i}$. □

Proposition 12 *The Algorithm 9 always converges to a limit point* $(y, z, \xi, \lambda) \in \mathbb{C}^m \times \mathbb{C}^m \times \mathbb{C}^m \times \mathbb{C}^m$ *if* $\rho = 1$.

Proof. The result for $(y, z) \in \mathbb{C}^m \times \mathbb{C}^m$ follows from Observation 6 and Proposition 7. The result for $(\lambda, \xi) \in \mathbb{C}^m \times \mathbb{C}^m$ follows from Observation 8 where $\Delta = y - z$ and $x^{(k+1)} = \lambda^{(k+1)}$. □

The formulation (4.22) is useful when $\operatorname{range}(A) \cap \mathcal{M}_b = \emptyset$, which indicates the presence of noise in data. The Observation 6 and Observation 7 give us an idea about the link between a regularization parameter ρ and the ability of algorithm to converge. However, it is not clear how to choose ρ properly at each k-th iteration of Algorithm 9. In the following part, we try to answer this question.

Proposition 13 *Let* $\xi = AA^\dagger z - z$ *be such that* $\xi \in \{\alpha \odot z, \alpha \in (-\rho, +\infty)^m\}$ *for* $z \in \mathbb{C}^m$ *and* $\rho > 0$. *Then* $\|\xi - \alpha \odot z\| = 0$ *for*

$$\alpha = |z|^{-2} \odot \operatorname{Re}\{\overline{AA^\dagger z} \odot z\} - 1. \tag{4.23}$$

Proof. The optimal α can be found from the following optimization problem

$$\min_{\alpha \in \mathbb{R}^m} \quad f(\alpha) := \tfrac{1}{2}\|\xi - \alpha \odot z\|^2$$

The constrains $\alpha \in (-\rho, +\infty)^m$ are not taken into account since it is given that a solution exists and it is unique due to strict convexity of f. Then, from the first order optimality conditions we obtain

$$\begin{aligned} \nabla f(\alpha) &= \nabla\left(\frac{1}{2}\|\xi\|^2 - \operatorname{Re}\{\xi^*(\alpha \odot z)\} + \frac{1}{2}\|\alpha \odot z\|^2\right) \\ &= -\operatorname{Re}\{\bar{\xi} \odot z\} + \alpha \odot |z|^2 = 0. \end{aligned}$$

Replacing $\bar{\xi}$ by $\overline{AA^\dagger z - z}$ the proof follows. □

From Proposition 13 we can conclude that for any $z \in \mathbb{C}^m$ the optimal α writes as

$$\alpha = \max\{-\rho\, 1_m,\ |z|^{-2} \odot \operatorname{Re}\{\overline{AA^\dagger z} \odot z\} - 1\}. \tag{4.24}$$

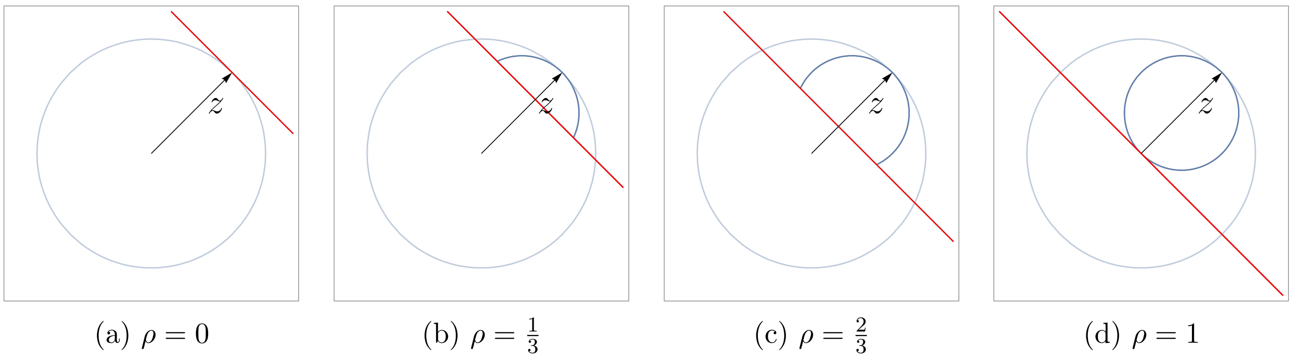


Figure 4.10: Set of possible projections $AA^{\dagger}z$ for the case of dimension 1 depending on ρ.

It can be seen that ρ changes the solution set for z. For example, if ρ is close to zero, then vector z and its projection on range(A) must be almost colinear to satisfy $\|AA^{\dagger}z - z - \alpha \odot z\| = 0$.

To have a clear picture in mind, let us visualize the set of possible projections for different ρ in the case of dimension 1. It can be seen on Figure 4.10 that when $\rho = 0$ the set of possible projections reduces to one point where the case for $\rho = 1$ indicates all possible projections in the set. The reason why we have such form of the set of possible projections can be seen on Figure 4.11 where the projection point is denoted by red color for many different linear spaces, which are visualized by gray dashed lines.

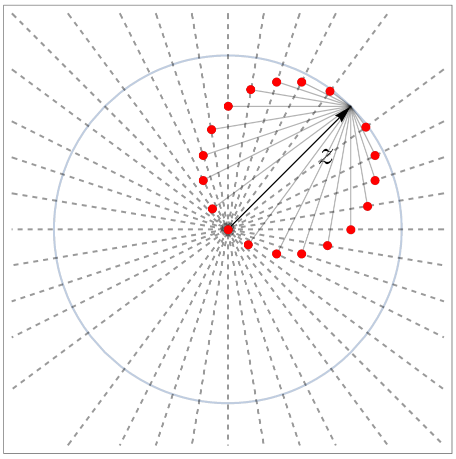


Figure 4.11: Set of possible projections $AA^{\dagger}z$ defined by $\rho = 1$ for the case of dimension 1.

This visualization brings an inside about how ρ affects the set of possible projections. However, at the same time, it gives an idea of how ρ can be properly selected during the iterations of the Algorithm 9. It is clear that at the begining of the iteration process it is reasonable to set $\rho = 0$ to guide iterations to the "perfect" point even if it does not exists. Closer algorithm converge to it – bigger ρ must be to be sure that the limit point exists. This kind of information can be obtained from (4.24). It is clear that to remove constrains defined by ρ, we need to have that $-\rho = \min\{\alpha_1, \ldots, \alpha_m,\}$ for $\alpha_i = |z_i|^{-2} \odot \text{Re}\{\overline{(AA^{\dagger}z)}_i \odot z_i\} - 1$ for $i \in \{1, \ldots, m\}$. Equivalently, we have that $\rho = \max\{-\alpha_1, \ldots, -\alpha_m,\}$. The value of ρ tells us how far we are from solution. Thus, following the logic explained above, we can use it to construct the formula for ρ in the following form

$$\rho = 1 - \min\{1, \max\{-\alpha_1, \dots, -\alpha_m\}\}. \tag{4.25}$$

The numerical study in Section 4.8 reveals the efficiency of this choice for ρ with a sufficient level of noise. However, when the noise dominates, the behavior near solution starts being unstable. For this goal, the empirical stabilization is proposed in the following form

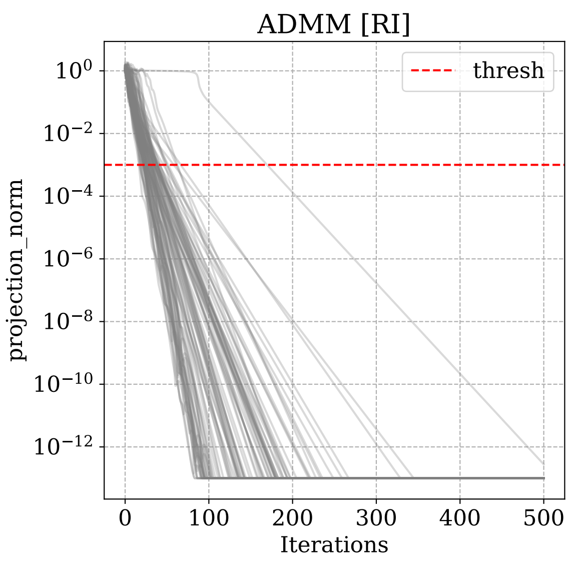
$$\text{If} \quad \text{dist}_{\text{norm}}(y, z) < \gamma \quad \text{then} \quad \rho = 1, \tag{4.26}$$

where $\gamma \in [0, 1]$ depends on the level of noise in the data. Thus, the final form of ADMM is presented in Algorithm 10 where all useful propositions and observations are applied.

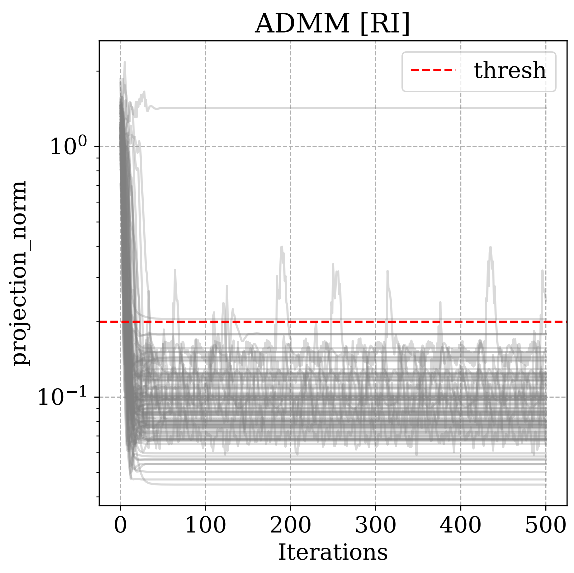
Algorithm 10: Adapted Relaxed Alternating Directions Method of Multipliers.

```
    Input: Initial point $x^{(0)} \in \mathbb{C}^n$, $\gamma \in [0,1]$.
    Output: Approximate solution $x \in \mathbb{C}^n$ of (1.2).
 1  $k = 0$, $\rho_0 = 0$, $\xi^{(0)} = \lambda^{(0)} = 0$, $y_0 = Ax^{(0)}$;
 2  Loop
 3      $z^{(k+1)} = b \odot \exp(i \arg(y^{(k)} + (1 - \rho^{(k)})\lambda^{(k)}))$;
 4      $x^{(k+1)} = A^\dagger z^{(k+1)}$;
 5      $y^{(k+1)} = Ax^{(k+1)}$;
 6      if stopping test is satisfied then
 7          break
 8      $\rho^{(k+1)} = 1 - \min\left\{1, \max\{|z^{(k+1)}|^{-2} \odot \mathrm{Re}\{\overline{y}^{(k+1)} \odot z^{(k+1)}\} - 1\}\right\}$;
 9      if $\text{dist}_{norm}(y^{(k+1)}, z^{(k+1)}) < \gamma$ then
10          $\rho^{(k+1)} = 1$
11      $\lambda^{(k+1)} = \frac{1}{1+\rho^{(k+1)}}\left(\lambda^{(k)} + y^{(k+1)} - z^{(k+1)}\right)$;
12      $k = k + 1$;
13  return: $x = x^{(k)}$.
```

Let us provide numerical simulations to verify the robustness and capability to solve all problems for Algorithm 10. On Figure 4.12 it can be observed that all problems almost without any oscillations can be solved by Algorithm 10 for the case when there is no noise in the model. However, there are still oscillations for the noisy case, when the algorithm reaches a limit of improvement of the approximation. That is why it is proposed to use γ parameter to switch to alternating projections when we are close to the solution. It was observed empirically that if $\gamma = 2\sigma$, then we can avoid the oscillations (Figure 4.13).

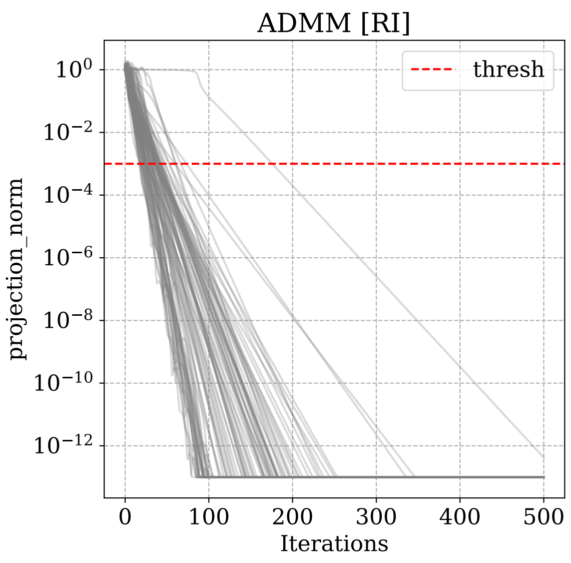


(a)

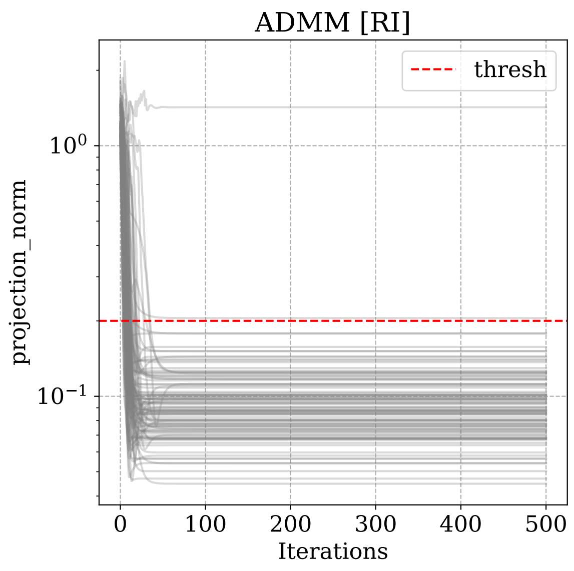


(b)

Figure 4.12: (a) Metric $\text{dist}_{\text{norm}}$ between approximation $x^{(k)}$ computed by Algorithm 10 and a solution for $n = 8$, $m = 32$, $\gamma = 0$, and $\sigma = 0$, and (b) for $\sigma = 0.1$.



(a)



(b)

Figure 4.13: (a) Metric $\text{dist}_{\text{norm}}$ between approximation $x^{(k)}$ computed by Algorithm 10 and a solution for $n = 8$, $m = 32$, $\gamma = 0.2$, and $\sigma = 0$, and (b) for $\sigma = 0.1$.

4.7 Initialization methods

For nonconvex problems it is highly important to select a proper initial guess to avoid local minimums during iterations. There are several methods [14, 12, 7, 13, 10, 9] which all are from the family of spectral algorithms because they are based on the computation of the largest eigenvalue of a matrix $Y = \frac{1}{m}\sum_{i=1}^{m} f(b_i)a_i a_i^*$, where $b_i \in \mathbb{R}^+$ is the i-th measurement and

$a_i \in \mathbb{C}^n$ is the i-th row of a transmission matrix $A \in \mathbb{C}^{m\times n}$ such that $b = |Ax|$, and function $f : \mathbb{R} \to \mathbb{R}$ differs for each method. The proper choice of f can reduce the required m to estimate x [13, 10, 9]. Since the goal of this work is to create a fast and robust algorithm to solve a phase correction problem, in this section we consider just two of initialization strategies to use in further numerical experiments instead of taking into account all of them. Following the published results in [7, 13], we decide to select Wirtinger flow (WF) initialization method [7] as a baseline algorithm, and Gao & Xu (GX) initialization algorithm [13] which is proved in the original paper to have better properties than WF.

Both WF and GX initialization algorithms requires computation of the leading eigenvector of matrix Y. The power method can efficiently solve this problem with a selected accuracy. Despite the fact that it can be easily found in literature, we put it in Algorithm 11 to make this document self-contained.

Algorithm 11: Power method.

```
  Input: Matrix $M \in \mathbb{C}^{n\times n}$, tolerance $\varepsilon > 0$.
  Output: The largest eigenvalue $\lambda \in \mathbb{C}$ and eigenvector $v \in \mathbb{C}^n$.
1 Set $v = \frac{1_n}{n}$, $\lambda = \frac{v^*Mv}{\|v\|^2}$.
2 Loop
3     Set $v = \frac{Mv}{\|Mv\|}$.
4     Set $\lambda' = \frac{v^*Mv}{\|v\|^2}$.
5     if $|\lambda' - \lambda| < \varepsilon$ then
6         break
7     Set $\lambda = \lambda'$.
8 Set $\lambda = \lambda'$;
9 return $\lambda$, $v$.
```

The goal of this section is not only to present the algorithms but also give the idea behind them. The principle of spectral methods is to use a transmission matrix $A \in \mathbb{C}^{m\times n}$ and a vector of measurements $b \in \mathbb{R}^m_+$ such that $b = |Ax|$ to approximate $x \in \mathbb{C}^n$. The Algorithm 12 is based on the fact if A is Gaussian then by the law of large numbers we have

$$\frac{1}{m}\sum_{i=1}^{m} b_i^2 a_i a_i^* \approx \|x\|^2 I + xx^*. \tag{4.27}$$

It can be seen that any leading eigenvector of $\|x\|^2 I + xx^*$ is of the form λx for $\lambda \in \mathbb{R}$. Thus for sufficiently large m, solution x can be recovered up to a sign and a global phase shift.

Lemma 1 *[43, Lemma 20]. Let* $a = \mathrm{Re}[a] + i\,\mathrm{Im}[a]$ *such that* $\mathrm{Re}[a], \mathrm{Im}[a] \sim \mathcal{N}\big(0, \frac{1}{\sqrt{2}}\big)$*, then for any fixed vector* $x \in \mathbb{C}^n$ *it holds that*

$$\mathbb{E}\big[|a^*x|^2 aa^*\big] = \|x\|^2 I + xx^*.$$

The normalization coefficient λ approximates $\|x\|$ but has no effect for a class of projection algorithms that is considered in this work. The reason comes from the update formula in Algorithm 7 that uses $\arg(Ax)$ expression, which is clearly invariant to multiplication by a real positive constant.

Algorithm 12: Wirtinger Flow: Initialization [7, Algorithm 1].

```
Input: Vector of measurements $b \in \mathbb{R}^m_+$, transmission matrix $A \in \mathbb{C}^{m \times n}$, Power
       method tolerance $\varepsilon > 0$.
Output: Initial guess $x^{(0)} \in \mathbb{C}^n$.
1 Set $Y = \frac{1}{m}\sum_{i=1}^{m} b_i^2 a_{i\cdot} a_{i\cdot}^*$, where $a_{i\cdot}$ is the $i$-th row of $A$.
2 Find the larges eigenvalue $v \in \mathbb{C}^n$ of $Y$ using Algorithm 11 with tolerance $\varepsilon$.
3 Set $\lambda = \sqrt{n\frac{\sum_{i=1}^m b_i^2}{\sum_{i=1}^m \|a_{i\cdot}\|^2}}$.
4 Set $x^{(0)} = \lambda v$.
```

The main result about WF initialization is presented in Theorem 8 which claims that $\mathcal{O}(n \log n)$ measurements are required to obtain an efficient initialization with high probability.

Theorem 8 *[7, Theorem 3.3] Let $x \in \mathbb{C}^n$ be an arbitrary vector and $b = |Ax| \in \mathbb{R}^m_+$ be m samples with $m \geq c_0 \cdot n \log n$, where c_0 is a sufficiently large numerical constant. Then the computed by Algorithm 12 estimate $x^{(0)}$ normalized to have squared Euclidean norm equal to $m^{-1}\sum_{i=1}^m b_i^2$,* [1] *obeys*

$$\operatorname{dist}(x^{(0)}, x) \leq \frac{1}{8}\|x\|,$$

with probability at least $1 - 10e^{-\gamma n} - 8/n^2$ (γ is a fixed positive numerical constant).

The idea of Gao & Xu initialization algorithm is similar to WF but with $f(x) = \frac{1}{2} - \exp(x/\lambda^2)$ which improves the quality of initial guess estimation. In the same way that we used in Lemma 1 let us show the motivation of such f.

Lemma 2 *[13, Lemma 5.2]. Let $a = \mathrm{Re}[a] + i\,\mathrm{Im}[a]$ such that $\mathrm{Re}[a], \mathrm{Im}[a] \sim \mathcal{N}\left(0, \frac{1}{2}\right)$, then for any fixed vector $x \in \mathbb{C}^n$ it holds that*

$$\mathbb{E}\left[\left(\frac{1}{2} - \exp\left(|a^* x|^2/\lambda^2\right)\right) aa^*\right] = \frac{xx^*}{4\|x\|^2}.$$

The main result about GX initialization is presented in Theorem 9 which claims that $\mathcal{O}(n)$ measurements are required to obtain an efficient initialization with high probability.

[1]The same results holds with the intialization from Algorithm 12 because $\sum_{i=1}^m \|a_i\|^n \approx mn$ with a standard deviation of about the square root of this quantity.

Algorithm 13: Gao & Xu initialization [13, Algorithm 1].

```
Input: Vector of measurements $b \in \mathbb{R}_+^m$, transmission matrix $A \in \mathbb{C}^{m \times n}$, Power
       method tolerance $\varepsilon > 0$.
Output: Initial guess $x^{(0)} \in \mathbb{C}^n$.
1 Set $\lambda^2 = \frac{\sum_{i=1}^m b_i}{m}$.
2 Set $Y = \frac{1}{m} \sum_{i=1}^m \left( \frac{1}{2} - \exp\left( b_i / \lambda^2 \right) \right) a_{i\cdot} a_{i\cdot}^*$, where $a_{i\cdot}$ is the $i$-th row of $A$.
3 Find the larges eigenvalue $v \in \mathbb{C}^n$ of $Y$ using Algorithm 11 with tolerance $\varepsilon$.
4 Set $x^{(0)} = \lambda v$.
```

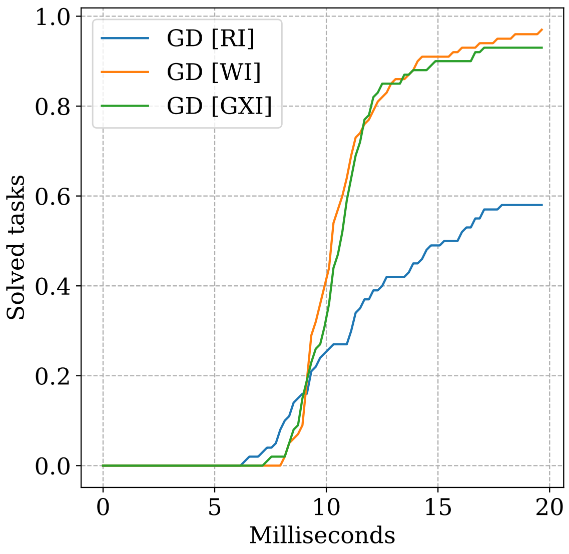
Theorem 9 *[13, Theorem 2.1] Let $x \in \mathbb{C}^n$ be an arbitrary vector and $b = |Ax| \in \mathbb{R}_+^m$ be m samples. Then for any $\theta > 0$ there exists constant C_θ such that for $m \geq C_\theta n$ the computed by Algorithm 13 estimate $x^{(0)}$ normalized to have squared Euclidean norm equal to $m^{-1} \sum_{i=1}^m b_i^2$, obeys*

$$\operatorname{dist}(x^{(0)}, x) \leq \sqrt{3\theta} \|x\|,$$

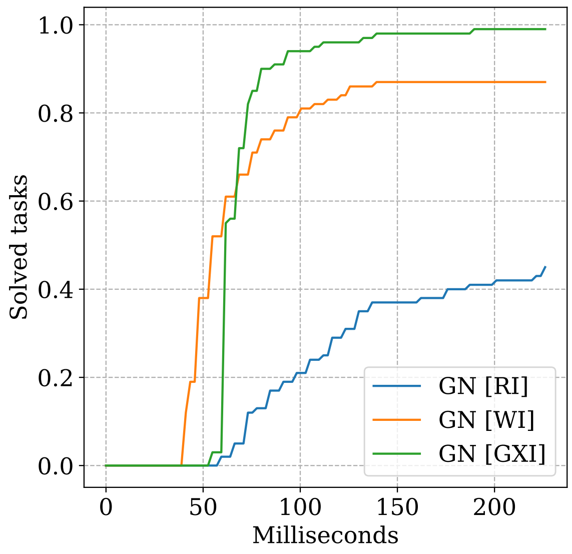
with probability at least $1 - 4e^{-c_\theta n}$ (c_θ is a fixed positive numerical constant).

Let us now provide a few time profiles for the phase retrieval methods that we discussed above which use different initialization strategies: random (RI), Wirtinger (WI) and Gao & Xu (GXI) initializers. It appears that they have different impact on the convergence of the considered phase retrieval algorithms. For example, on Figure 4.14a it can be seen that both strategies WI and GXI increases the number of solved problems in comparison with a random starting point. However, both of them have approximately the same effect on the speed of convergence for the gradient descent algorithm with a secant line search. The secant line search was selected since it is faster than backtracking line search, which can be seen on Figure 4.3. This picture is not the same for other methods. For instance, on Figure 4.14b it can be seen that GXI initialization allows to solve more problems than both WI and RI, where at the same time for half of the problems WI allows to converge faster. Alternating projections algorithm reveals that GXI initialization improves both the speed of convergence and the number of solved problems in comparison with Wirtinger and random initialization strategies. The most interesting behavior can be observed on Figure 4.14d. It reveals that the strategy which allows both to obtain the fastest convergence and to solve all problems is a random initialization strategy. Recall, that WI and GXI requires additional computational time to find starting point. More precisely, it needs to compute a matrix Y and to find its leading eigenvector by means of power method. It reveals that this effort is not relevant for ADMM (Algorithm 10) and it is more efficient from the computational speed point of view to generate randomly an initial point.

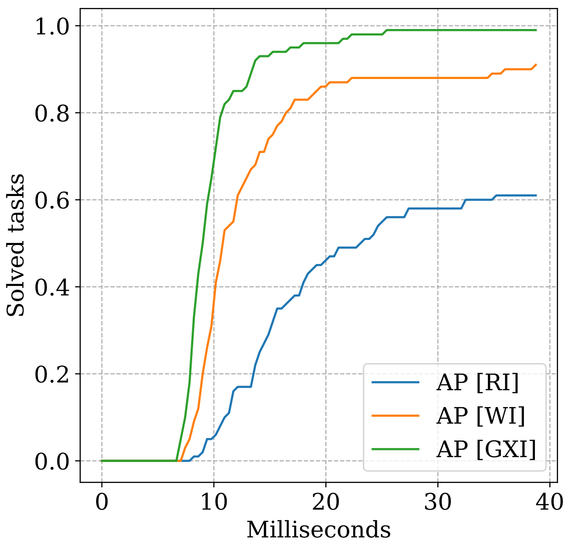
The experiments were done for $n = 32$, $m = 128$, without noise $\sigma = 0$, where 100 problems were generated for this aim. The metric $\operatorname{dist}_{\text{norm}}$ is used to measure distance between a solution and an approximation. Also, the stopping tolerance ε in Algorithm 11 was set to 0.001 in order to approximate the leading eigenvector.



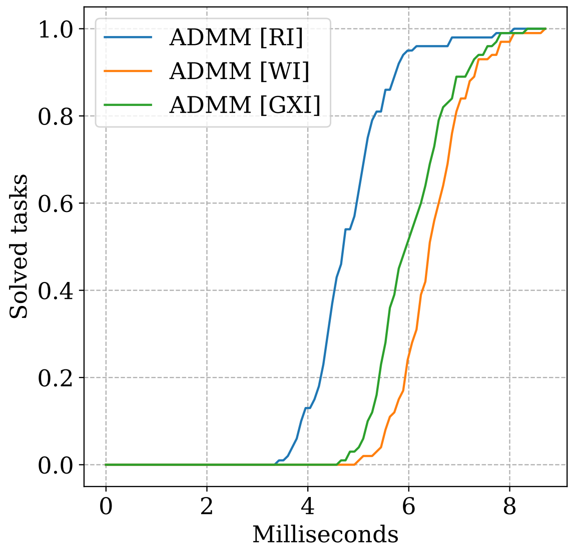
(a) Gradient descent (Algorithm 5)



(b) Gauss-Newton (Algorithm 6)



(c) Alternating projections (Algorithm 7)



(d) ADMM (Algorithm 10)

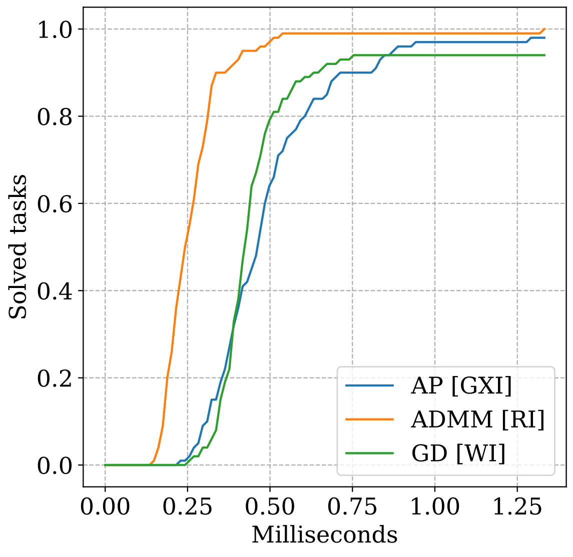
Figure 4.14: Time profiles for the phase retrieval algorithms with different initialization strategies: [RI] - random, [WI] - Wirtinger, [GXI] - Gao & Xu.

4.8 Numerical results

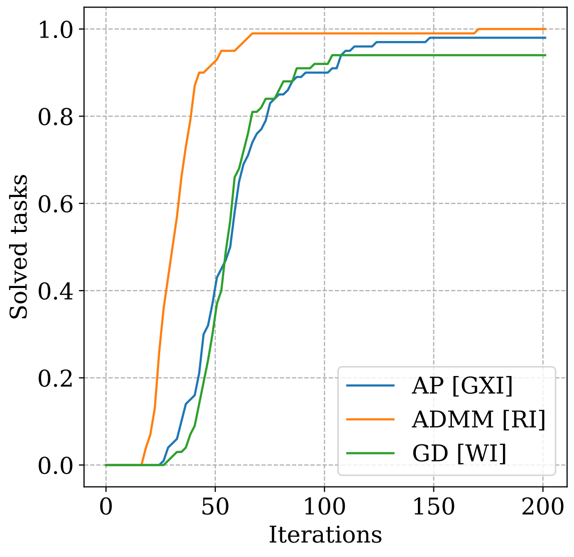
In this section, we present numerical results for different optimization algorithms to solve a set of phase retrieval problems in order to find the fastest and the most robust algorithm.

Following the previous research, we exclude Gauss-Newton method from a list of comparison. The reason can be clearly seen on Figure 4.14, where Gauss-Newton method can be up to 10 times slower than other algorithms. Also, we use different initialization strategies for different algorithms. On Figure 4.14 it can be clearly seen that the best initialization approach for gradient descent is Wirtinger flow, alternating projections algorithm performs faster with Gao & Xu initialization, and ADMM does not require any initialization algorithms and uses a random complex vector as a starting point.

We present time and iterations profiles for these three algorithms with corespondent initialization strategies, for different number of beams $n \in \{8, 32, 128\}$ and measurements $m = 4n$, for two levels of noise $\sigma = 0$ and $\sigma = 0.1$. The stopping tolerance of metric $\mathrm{dist}_{\mathrm{norm}}$ for $\sigma = 0$ is 0.001, for $\sigma = 0.1$ is 0.2.

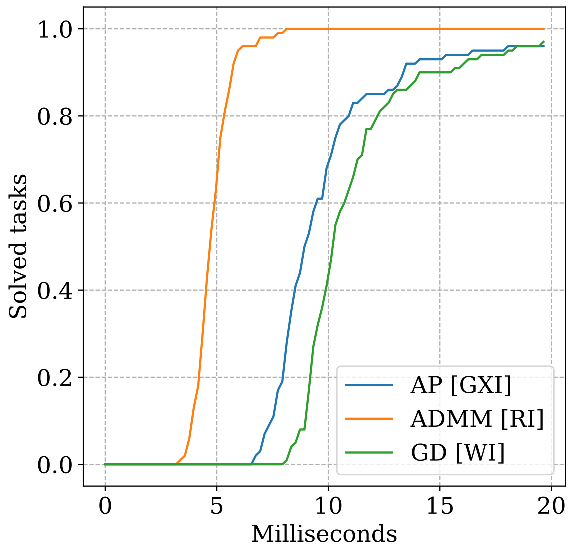


(a) Time profile

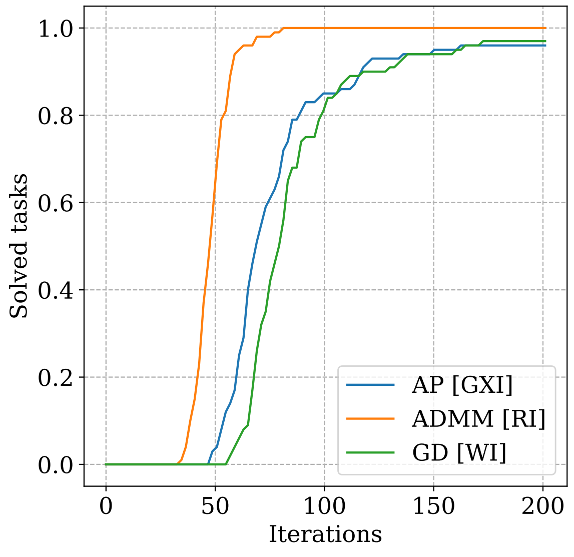


(b) Iteration profile

Figure 4.15: Profiles computed for $n = 8$, $m = 32$, $\sigma = 0$ and 100 problems in total.

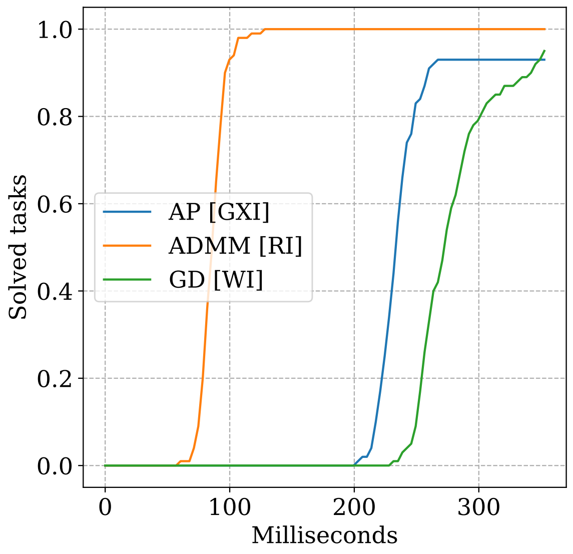


(a) Time profile

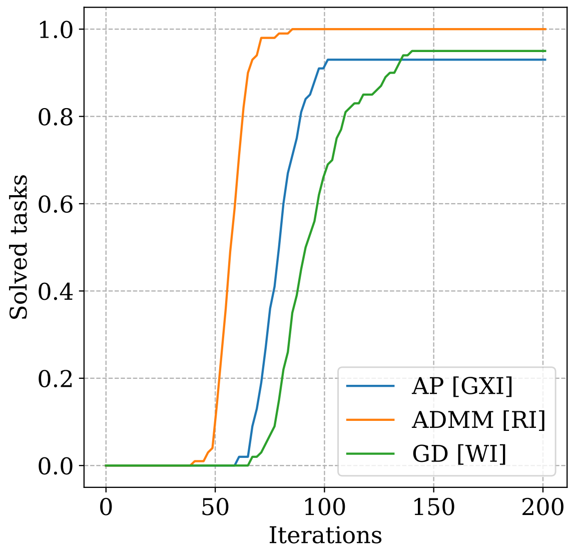


(b) Iteration profile

Figure 4.16: Profiles computed for $n = 32$, $m = 128$, $\sigma = 0$ and 100 problems in total.

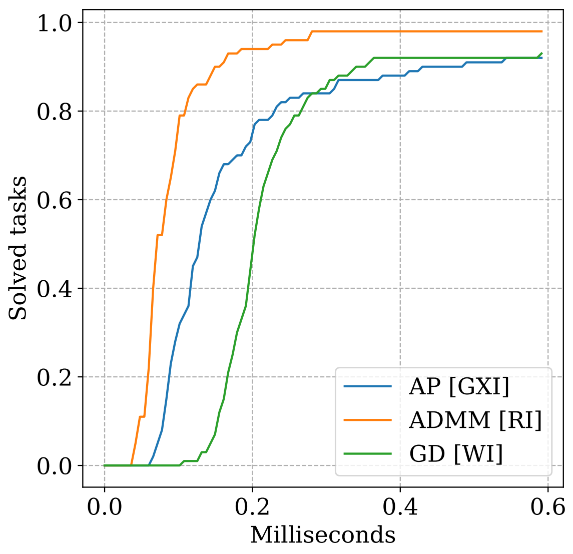


(a) Time profile

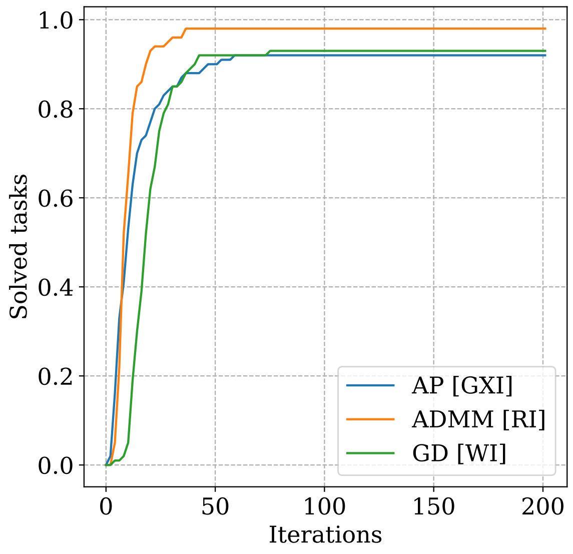


(b) Iteration profile

Figure 4.17: Profiles computed for $n = 128$, $m = 512$, $\sigma = 0$ and 100 problems in total.

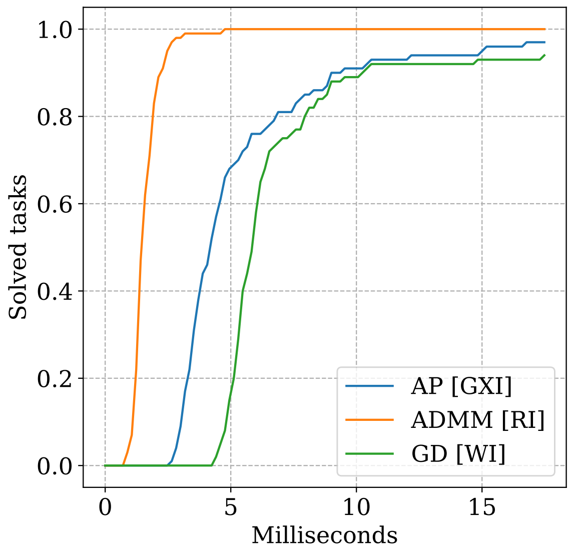


(a) Time profile

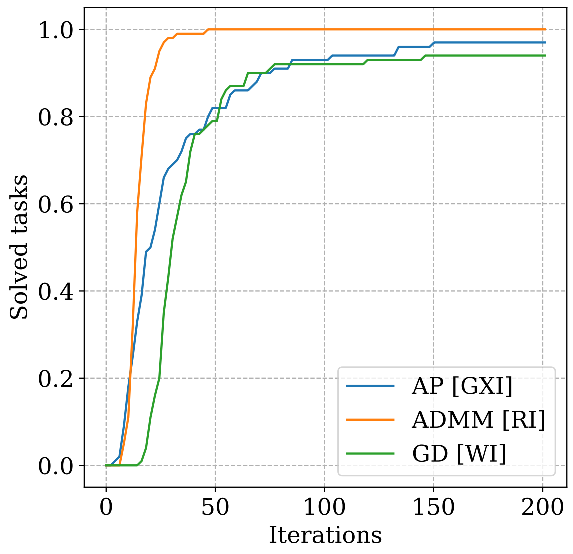


(b) Iteration profile

Figure 4.18: Profiles computed for $n = 8$, $m = 32$, $\sigma = 0.1$ and 100 problems in total.

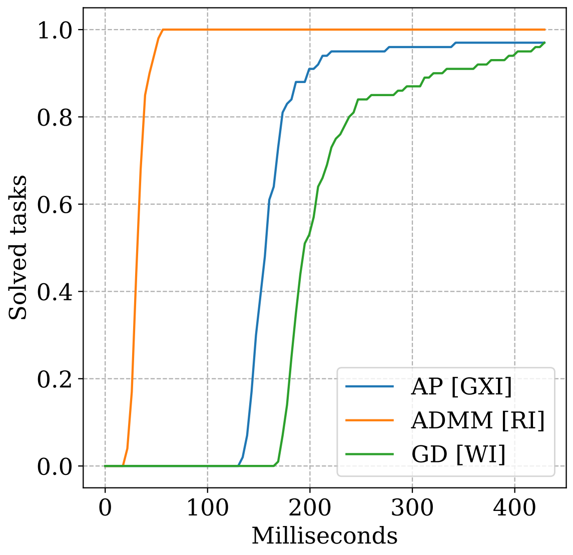


(a) Time profile

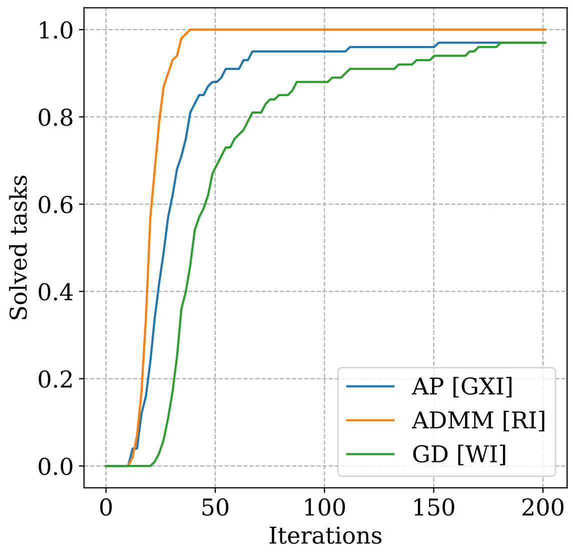


(b) Iteration profile

Figure 4.19: Profiles computed for $n = 32$, $m = 128$, $\sigma = 0.1$ and 100 problems in total.



(a) Time profile

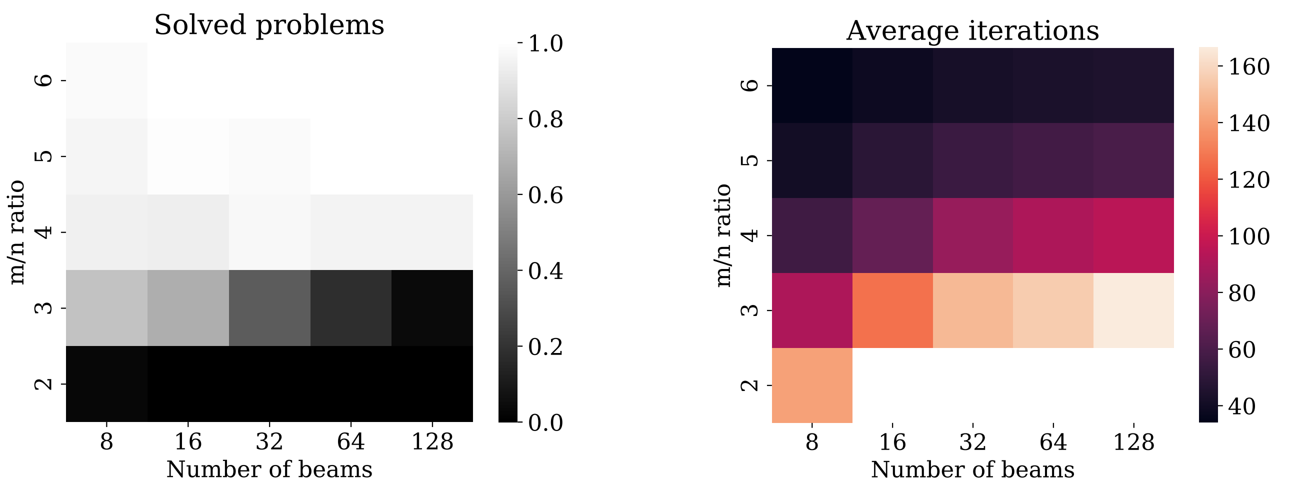


(b) Iteration profile

Figure 4.20: Profiles computed for $n = 128$, $m = 512$, $\sigma = 0.1$ and 100 problems in total.

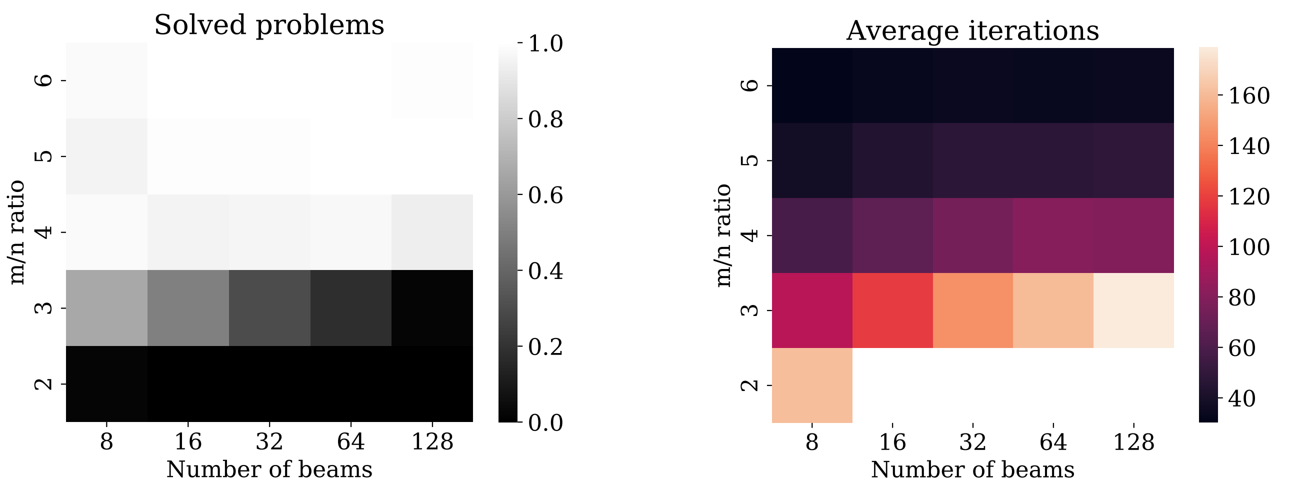
These profiles reveals that ADMM (Algorithm 10) in all of the cases performs faster than alternating projections (Algorithm 7) and gradient descent (Algorithm 5) algorithms. In addition, with increasing n this difference in speed increases.

Now, let us provide heat maps (explained in Section 2.6) for these algorithms to observe the solving capabilities for different n and m, with and without noise. It can be clearly seen (Figure 4.21 - Figure 4.26) that alternating projections and gradient descent algorithms requires $m \geq 4n$ measurements to solve a phase retrieval problem for both cases: $\sigma = 0$ and $\sigma = 0.1$. At the same time, ADMM algorithm can solve more that 50% of all problems even when $m = 3n$, which can be observed on Figure 4.23 and Figure 4.26.



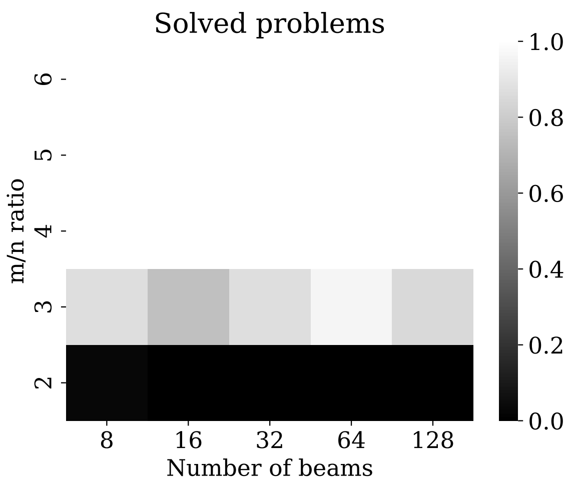
(a) Time profile (b) Iteration profile

Figure 4.21: Profiles computed for gradient descent algorithm (Algorithm 5) with $\sigma = 0$, 100 problems, and Wirtinger initialization (Algorithm 12)

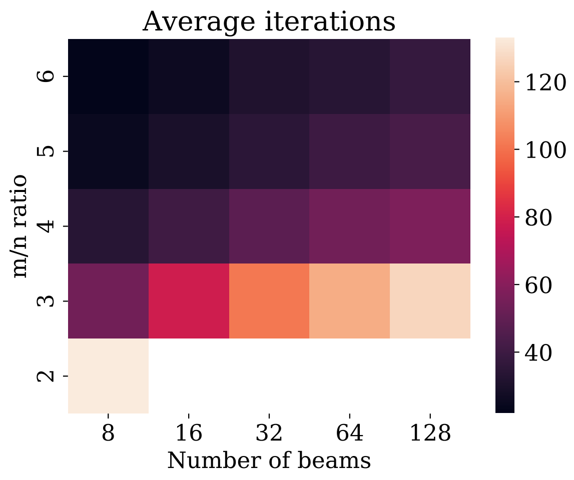


(a) Time profile (b) Iteration profile

Figure 4.22: Profiles computed for alternating projections algorithm (Algorithm 7) with $\sigma = 0$, 100 problems, and Gao & Xu initialization (Algorithm 13)

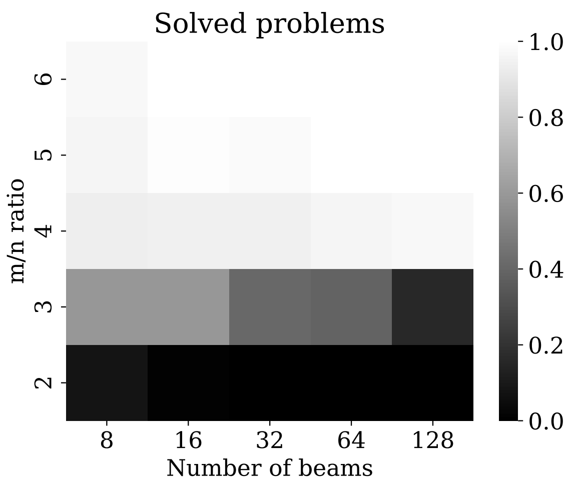


(a) Time profile

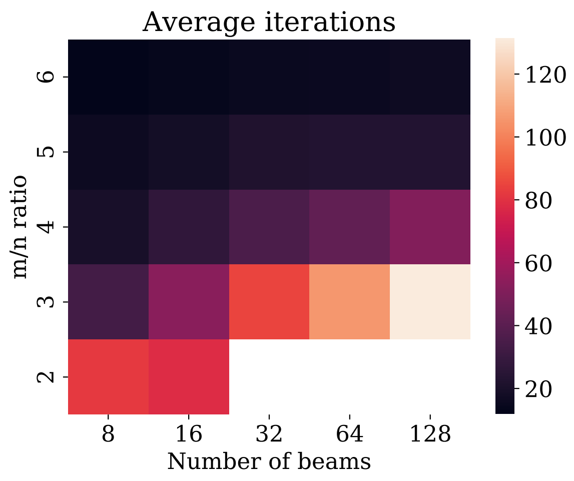


(b) Iteration profile

Figure 4.23: Profiles computed for ADMM algorithm (Algorithm 10) with $\sigma = 0$, 100 problems, and random initialization.

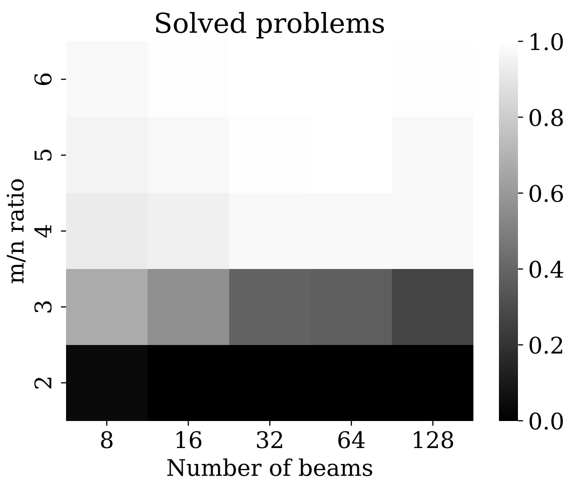


(a) Time profile

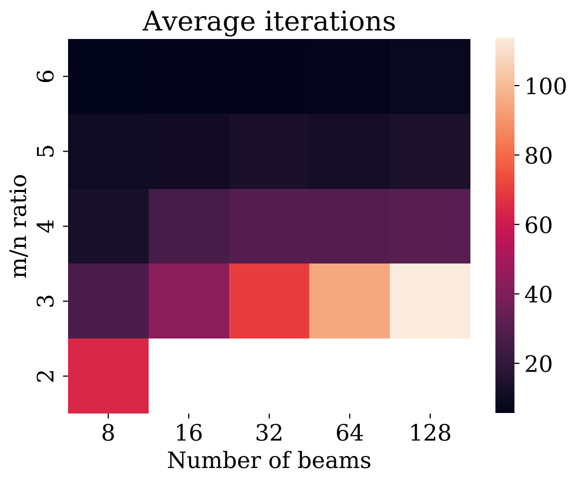


(b) Iteration profile

Figure 4.24: Profiles computed for gradient descent algorithm (Algorithm 5) with $\sigma = 0.1$, 100 problems, and Wirtinger initialization (Algorithm 12)

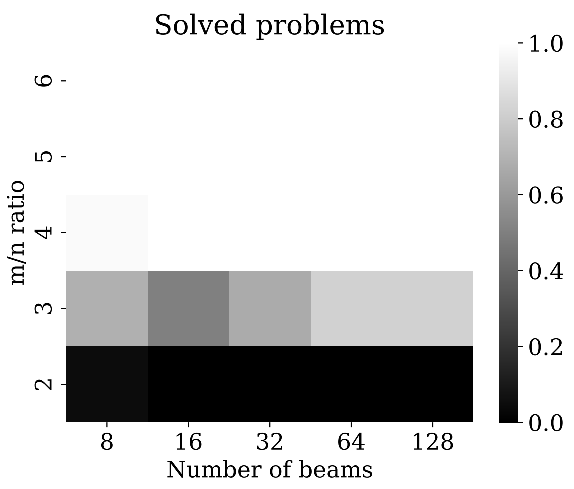


(a) Time profile

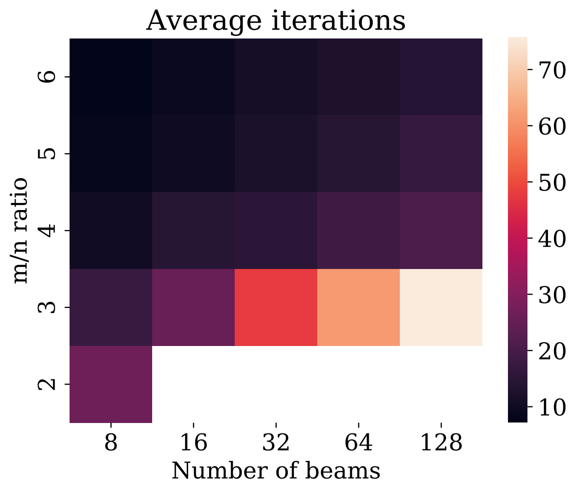


(b) Iteration profile

Figure 4.25: Profiles computed for alternating projections algorithm (Algorithm 7) with $\sigma = 0.1$, 100 problems, and Gao & Xu initialization (Algorithm 13)



(a) Time profile



(b) Iteration profile

Figure 4.26: Profiles computed for ADMM algorithm (Algorithm 10) with $\sigma = 0.1$, 100 problems, and random initialization.

Thus, we can conclude that ADMM (Algorithm 10) appears to be faster and more robust (with a properly selected value of γ) than other algorithms considered in this work, which potentially gives a real improvement of the phase correction algorithm considered in the next chapter.

Chapter 5

Phase corrections by means of optimization methods

In this chapter, we consider a phase correction problem to solve using a set of optimization algorithms, which aim to find a solution of (1.2). In Section 1.3, we briefly described the problem and now, let us give a formal definition.

Let us consider a vector $x^{(k)} \in \mathbb{C}^n$ representing n input lasers whose vector of phases $\arg(x^{(k)})$ is unknown, but with a known vector of amplitudes $|x^{(k)}|$. The output is given by a vector of m intensity measurements $b^{(k)} \in \mathbb{R}_+^m$ for $m > n$. The link between $x^{(k)}$ and $b^{(k)}$ can be modeled by means of a transmission matrix $A \in \mathbb{C}^{m \times n}$ such that $b^{(k)} = |Ax^{(k)}|$ where A represents a scattering device and operation $|\cdot|$ simulates the measurement process. Let $\widehat{x} \in \mathbb{C}^n$ be a given target signal. The goal is to apply a phase correction to the vector $x^{(k)}$ to obtain a new vector $x^{(k+1)}$ such that $|x^{(k+1)}| = |x^{(k)}|$ and $|Ax^{(k+1)}| = |A\widehat{x}|$. Suppose that $\widetilde{x}^{(k)}$ is a solution of the equation

$$|Ax| = b^{(k)}, \text{ where } b^{(k)} = |Ax^{(k)}|, \tag{5.1}$$

such that $\arg(\widetilde{x}^{(k)}) = \arg(x^{(k)})$ up to a constant. The current phase values are then modified by means of the phase modulation to produce a new vector $x^{(k+1)}$ such that

$$\arg(x^{(k+1)}) = \arg(x^{(k)}) - \arg(\widetilde{x}^{(k)}) + \arg(\widehat{x}).$$

Therefore, for some constant $c \in \mathbb{R}$, we will have

$$\arg(x^{(k+1)}) = \arg(\widehat{x}) + c.$$

The solution of (5.1) can be done by a phase retrieval algorithm (for instance, the alternating projection). But the computation of a nearly exact solution of this equation is not relevant in our context. The first reason is that the computational cost can be too large. The second reason is that the convergence to a global solution is not always guaranteed. The third reason comes from robustness. Due to noisy data, it is useless to compute a nearly exact solution. It is better to perform some few iterations of a phase retrieval algorithm to get an approximate solution of (5.1), to modify the input sources by phase modulation, then to perform new magnitude measurements and then to loop. This process is formulated in Algorithm 14. In practice, this loop is continuously applied and never stop because of phase deviations.

It is worth noting that the steps 2 and 4 are not numerical implementations and that the value of $\arg(x^{(k)})$ is never explicitly known. In Step 2, the vector $b^{(k)}$ follows from physical

Algorithm 14: Opto-numerical phase correction algorithm [31, Algorithm 1]

1. *Initialization*: Let $\widehat{x}$ be the known target signal. Let $x^{(0)}$ be the unknown vector of initial phases, such that $|x^{(0)}| = |\widehat{x}|$. Set $k = 0$.
2. *Magnitude measurement*: Let $b^{(k)} = |Ax^{(k)}|$.
3. *Phase retrieval inner algorithm*: Compute $\widetilde{x}^{(k)} \in \mathbb{C}^n$ as an approximate solution of
$$|Ax| = b^{(k)}. \tag{5.2}$$
4. *Phase modulation*: Modify the phases of input signal according to
$$x^{(k+1)} = x^{(k)} \odot \exp(i(\arg(\widehat{x}) - \arg(\widetilde{x}^{(k)}))). \tag{5.3}$$
5. *Return loop*: Set $k = k + 1$ and go to 2.

measurements by means of a scattering device with photodetectors. Step 4 corresponds to phase modulations of the input sources. For a detailed experimental setting and true experiments, see [31]. However, in our numerical experiments the phases are given by the arguments of a true vector $x^{(k)}$ and the magnitude measurements are simulated by calculating $|Ax^{(k)}|$.

5.1 Numerical results

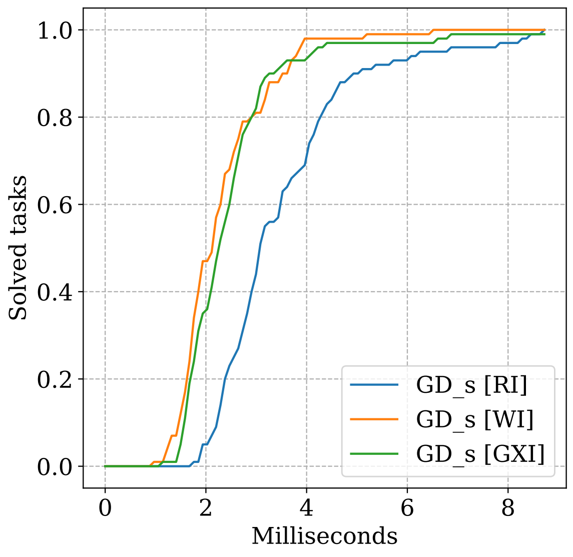
In this section, we present numerical results for different phase retrieval algorithms applied to the problem of a phase correction.

At first, let us select the best initialization strategy for starting point computation in the same way, as we did on Figure 4.14. The reason why the results on Figure 4.14 are not used directly in this section is that now we solve a phase correction problem, but not a phase retrieval, and the metric to measure distance between a solution and an approximation is $\mathrm{q_{norm}}$ instead of $\mathrm{dist_{norm}}$. Thus, from Figure 5.1 we conclude that:

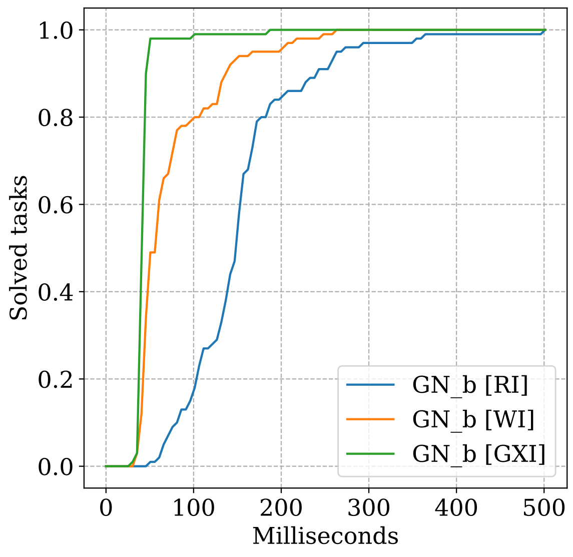
1. Gauss-Newton is extremely slower than other methods (Figure 5.1b) and will not be considered for further comparison.
2. GXI and WI initialization strategies are equivalent for gradient descent algorithm (Figure 5.1a).
3. The best initialization strategy for alternating projections algorithm is Gao & Xu initialization [13].
4. The best initialization strategy for ADMM is Gao & Xu initialization [13]. Note, that it is different from what we observe on Figure 4.14d. The reason comes from different metric. It reveals that GXI is better to use if $\mathrm{q_{norm}}$ is considered as a distance measure, and RI if $\mathrm{dist_{norm}}$.

The experiments were done for $n = 32$, $m = 128$, without noise $\sigma = 0$, where 100 problems were generated for this aim. The metric $\mathrm{q}_{\mathrm{norm}}$ is used to measure distance between a solution and an approximation.

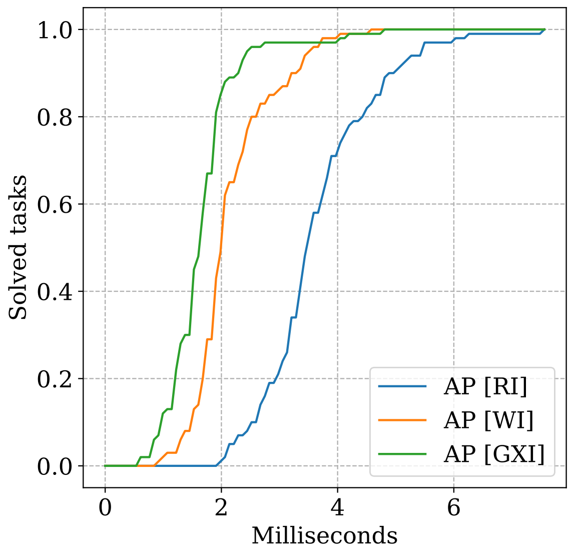
Then, from Figure 5.2 - Figure 5.7 we conclude that ADMM algorithm, which is used in step 3 of Algorithm 14 has the highest speed in solving a phase correction problem in both cases: with noise $\sigma = 0.1$ and without it ($\sigma = 0$).



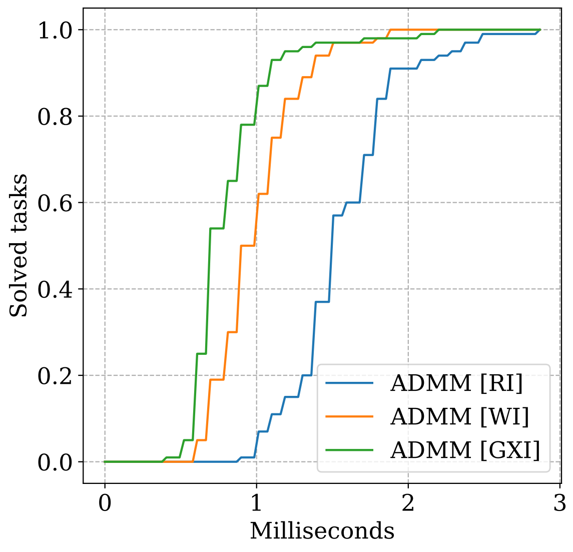
(a) Gradient descent (Algorithm 5)



(b) Gauss-Newton (Algorithm 6)

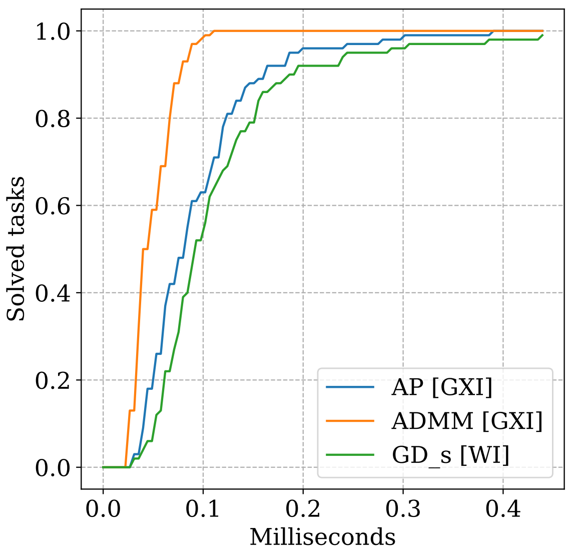


(c) Alternating projections (Algorithm 7)

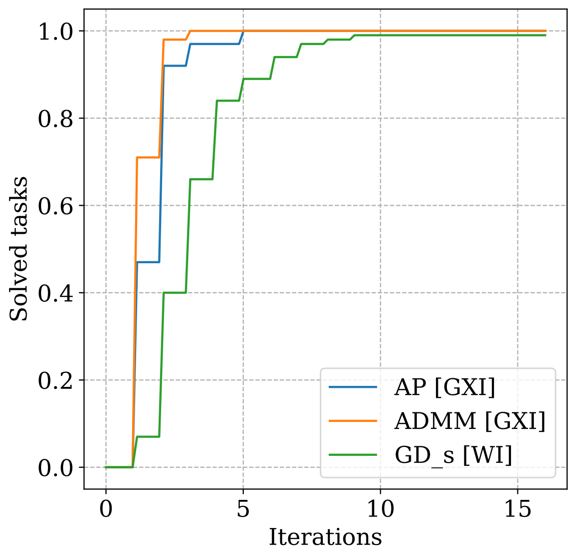


(d) ADMM (Algorithm 10)

Figure 5.1: Time profiles for the phase retrieval algorithms to solve phase correction problems with different initialization strategies: [RI] - random, [WI] - Wirtinger, [GXI] - Gao & Xu.

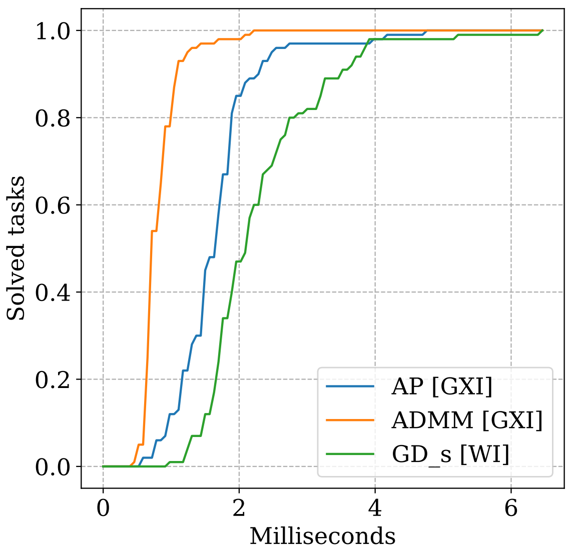


(a) Time profile

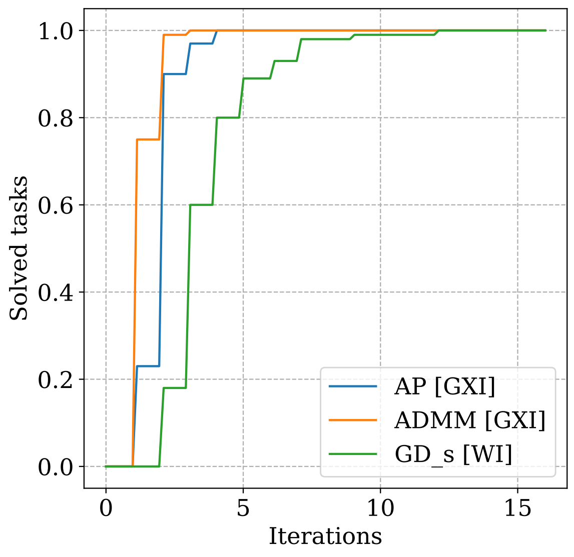


(b) Iteration profile

Figure 5.2: Profiles computed for $n = 8$, $m = 32$, $\sigma = 0$ and 100 problems in total.

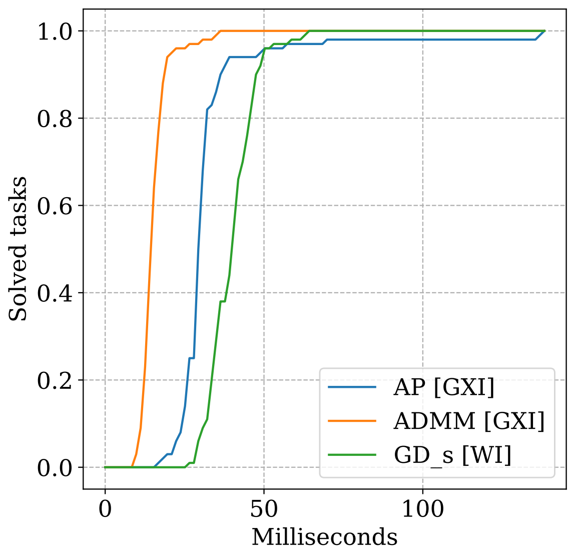


(a) Time profile

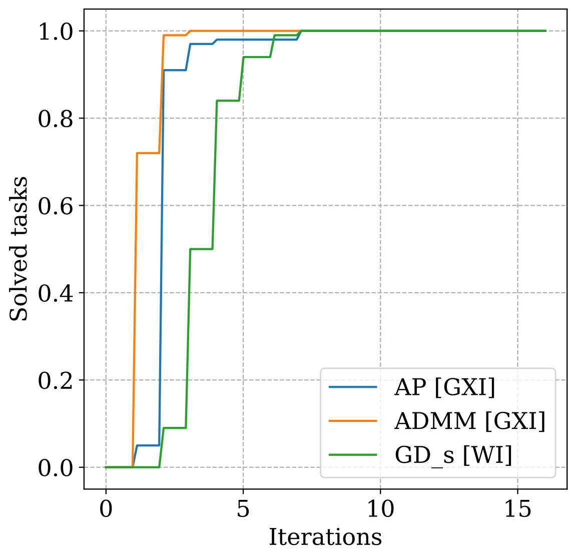


(b) Iteration profile

Figure 5.3: Profiles computed for $n = 32$, $m = 128$, $\sigma = 0$ and 100 problems in total.

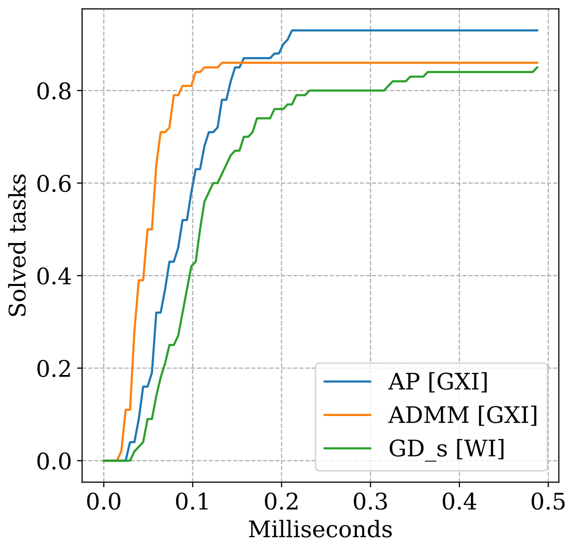


(a) Time profile

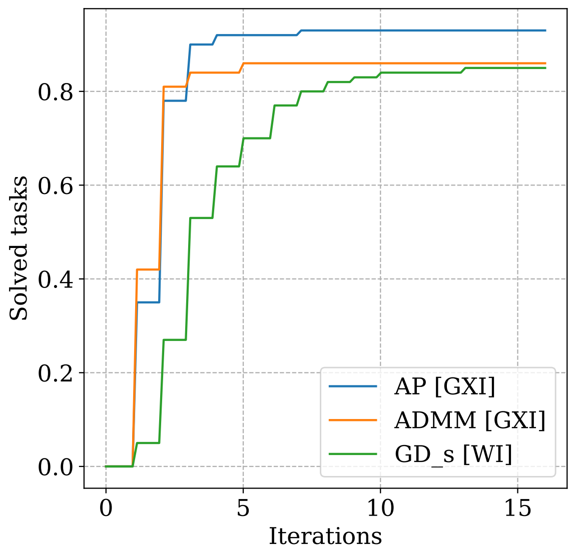


(b) Iteration profile

Figure 5.4: Profiles computed for $n = 128$, $m = 512$, $\sigma = 0$ and 100 problems in total.

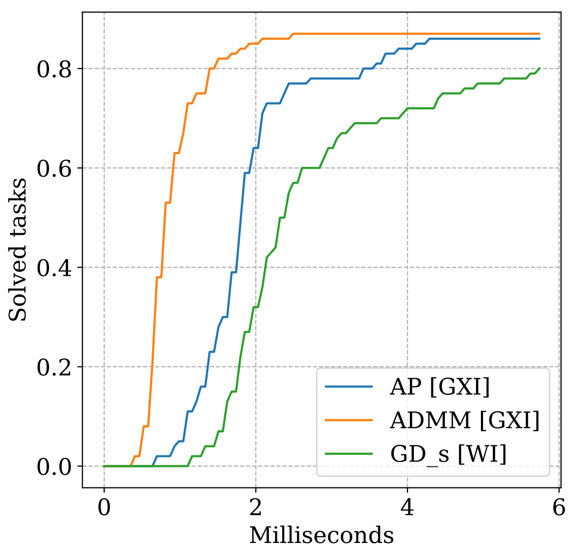


(a) Time profile

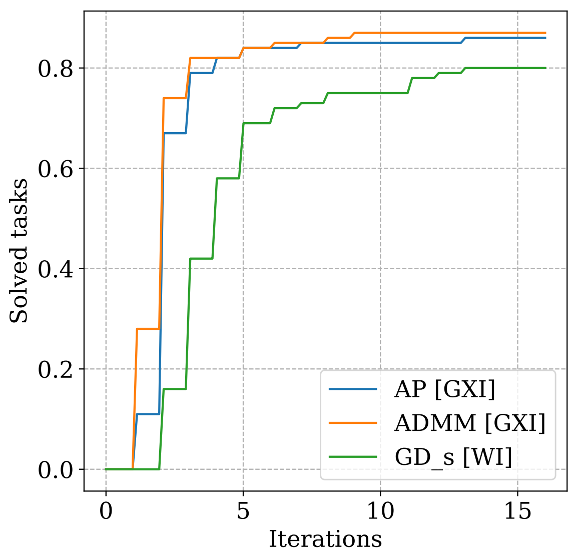


(b) Iteration profile

Figure 5.5: Profiles computed for $n = 8$, $m = 32$, $\sigma = 0.1$ and 100 problems in total.

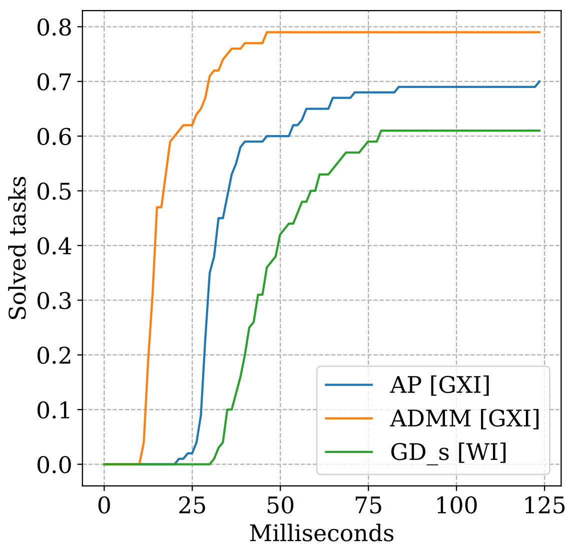


(a) Time profile

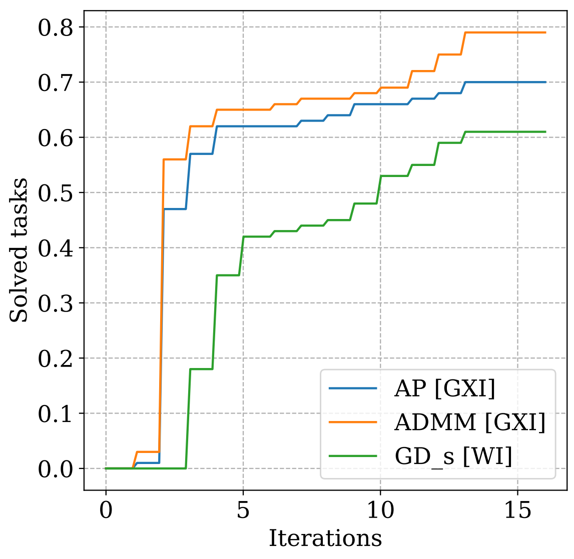


(b) Iteration profile

Figure 5.6: Profiles computed for $n = 32$, $m = 128$, $\sigma = 0.1$ and 100 problems in total.



(a) Time profile



(b) Iteration profile

Figure 5.7: Profiles computed for $n = 128$, $m = 512$, $\sigma = 0.1$ and 100 problems in total.

Chapter 6

Phase corrections by means of neural networks

In this chapter, the Algorithm 14 is considered where a neural network model is used in step 3 to compute an approximation $\widetilde{x}^{(k)}$. Two kind of results were obtained in this domain. The first is the way to train a neural network by the quasi-reinforcement learning algorithm (Algorithm 2) and use it in a phase correction loop where a target $\widehat{x} \in \mathbb{C}^n$ is fixed. It is shown that this model is highly scalable and needs extremely low time to compute $\widetilde{x}^{(k)} \in \mathbb{C}^n$. However, the practical application requires the possibility to dynamically change $\widehat{x}$ which is impossible in this context. To overcome this difficulty, the second result is presented where it is revealed how to build and train a target adaptive neural network. The advantages and disadvantages are discussed also.

6.1 Research path

The form of the networks that are considered in this chapter have a simple linear nature, which may be confusing at the first glance since the problem that they solve is complex and nonlinear. However, their simplicity arises from an among of numerical and experimental tests of different network architectures, and several learning approaches.

At the beginning, the idea was to train a deep network with many layers and different activation functions to solve directly a phase retrieval problem, which would have led to a phase correction in one step. However, this idea works just for a small number of beams (up to 6). That is why it was proposed to integrate a network in a phase correction loop in order to train to perform a set of corrections instead of just one. Because of a high similarity with the reinforcement learning framework we used its terminology to explain our idea.

Due to the limitation in a phase correction speed, the network must be lightweight and simultaneously capture nonlinear dependencies. The first idea was to use a neural network with two layers and ReLU activation function between them to compute phase correction values. However, the quality of this model was not appropriate for our goals.

In addition, it was noted that instead of predicting phase values, it is better to predict real and imaginary parts of a complex number and then apply $\arg(\cdot)$ function to retrieve phases. This has a logical reason because instead of predicting fixed values $\varphi \in [0, 2\pi]$, we aim that the relation of imaginary and real parts be fixed. We can say that this approach has more degree of freedom in some sense.

A prediction of real and imaginary parts and a usage of arg(·) function at the output gives not only an additional degree freedom, it also naturally introduce a nonlinearity to the network. Note, that even if the model is linear, the usage of arg(·) can be considered as a nonlinear activation function. It was observed numerically, that there is no need to use a multilayer neural network with classical activation functions because of the presence of arg(·). That is why the simple network form (6.1) is proposed.

6.2 Target fixed neural network

In this section we present the neural network based algorithm to solve the phase retrieval problem in step 3 of the Algorithm 14 approximately for a fixed target $\widehat{x}$.

Let $\mathrm{TFNN}(x) : \mathbb{R}^m \to [-\pi, \pi]^n$ be a neural network model such that

$$\mathrm{TFNN}(x) = \arg(Wx), \tag{6.1}$$

where $W \in \mathbb{C}^{n\times m}$ is a complex matrix of trainable parameters. The Algorithm 15 was developed to find W such that TFNN can compute phase corrections.

Algorithm 15: QRL algorithm for TFNN

```
1. Initialization: Let $N \in \mathbb{N}$ be the size of the batch. Let $K_{\max} \in \mathbb{N}$ be a number of corrections. Let $I_{\max} \in \mathbb{N}$ be a number of iterations. Let $\widehat{x} \in \mathbb{C}^{1\times n}$ be the known target signal. Let $W \in \mathbb{C}^{n\times m}$ be a random matrix. Set $l = 0$.

2. Data generation: Generate a batch of initial phases $X^{(0)} \in \mathbb{C}^{N\times n}$, such that $|x_j^{(0)}| = |\widehat{x}|$ for $j \in \{1, \dots, N\}$. Set $k = 0$
   (Step 2 in Algorithm 2 where $A_0 = X^{(0)}$ and $S_0$ is computed on the next step)

3. Magnitude measurement: Let $B^{(k)} = |X^{(k)} A^\top|$.
   (Step 3d in Algorithm 2 where $S_t = B^{(k)}$)

4. Phase correction computation: Let $\Phi^{(k)} = \arg(B^{(k)} W^\top)$
   (Step 3a in Algorithm 2 where $\hat{A}_t = \Phi^{(k)}$)

5. Gradients computation: $g = \frac{1}{N}\sum_{j=1}^{N} \begin{pmatrix} \nabla_{\mathrm{Re}[W]}\, \mathrm{q}_{\mathrm{norm}}\left(e^{i\Phi_j^{(k)}}, X_j^{(k)}\right) \\ \nabla_{\mathrm{Im}[W]}\, \mathrm{q}_{\mathrm{norm}}\left(e^{i\Phi_j^{(k)}}, X_j^{(k)}\right) \end{pmatrix}$
   (Step 3b in Algorithm 2 where $\Delta w_t^{(k)} = g$)

6. Parameters update: Update $\begin{pmatrix} \mathrm{Re}[W] \\ \mathrm{Im}[W] \end{pmatrix}$ applying the Adam update rule [18] with parameters $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$.
   (Step 3c in Algorithm 2)

7. Phase modulation: Modify the phases of input signal according to
   $$X^{(k+1)} = X^{(k)} \odot \exp(i(\arg(\widehat{x}) - \Phi^{(k)})). \tag{6.2}$$
   (Step 3d in Algorithm 2 where $A_{t+1} = X^{(k+1)}$)

8. Return correction loop: Set $k = k + 1$. If $k < K_{\max}$ then go to 3.

9. Return iteration loop: Set $l = l + 1$. If $l < I_{\max}$ go to 2.
```

The idea is to plug the model in the simulated phase correction loop in order to train the network to converge to target from any starting signal $x^{(0)}$. This idea can be formulated in terms of the quasi-reinforcement learning algorithm (Algorithm 2) where the agent *Agent* is the model (6.1), the action $\hat{A}_t$ is a phase correction vector $\varphi^{(k)} = \arg(\widehat{x}) - \arg(\widetilde{x}^{(k)})$, the state

S_t is the vector of intensities $b^{(k)}$ and the reward $\mathcal{R}$ can be computed by $\mathrm{q}_{\mathrm{norm}}$ between $x^{(k)}$ and $\widetilde{x}^{(k)}$. The adaptation of the Algorithm 2 is presented in Algorithm 15, where each step in Algorithm 15 is connected to some step in Algorithm 2.
Let us explain each step of Algorithm 15 in details. To simplify the notations for explanations of steps 2-7 let us consider that k is fixed and remove it from notations, and consider that $j \in \{1, \dots, N\}$ where it is not explicitly specified.

In step 1 the initial parameters N, $K_{\max}$, $I_{\max}$, $\widehat{x}$, and W are initialized. Let us give the information about all of them. The size of batch N affects the speed of convergence of the learning algorithm. A small value like 16, 32 or 64 can produce low informative vector of the ascent direction g in step 5. A large value like 10^4 or 10^5 gives to much information for g. The problem is that the function to optimize (in our case it is $f(W) := \frac{1}{N}\sum_{j=1}^{N} \mathrm{q}_{\mathrm{norm}}\big(e^{i\Phi_j}, X_j\big)$) is not convex which means that if g is computed with big N then with high probability we converge to one of local minimums. That is why there is an empirical trade off between little and highly informative g. It was observed that for $N = 4096$ this trade off is satisfied (see Figure 6.1).

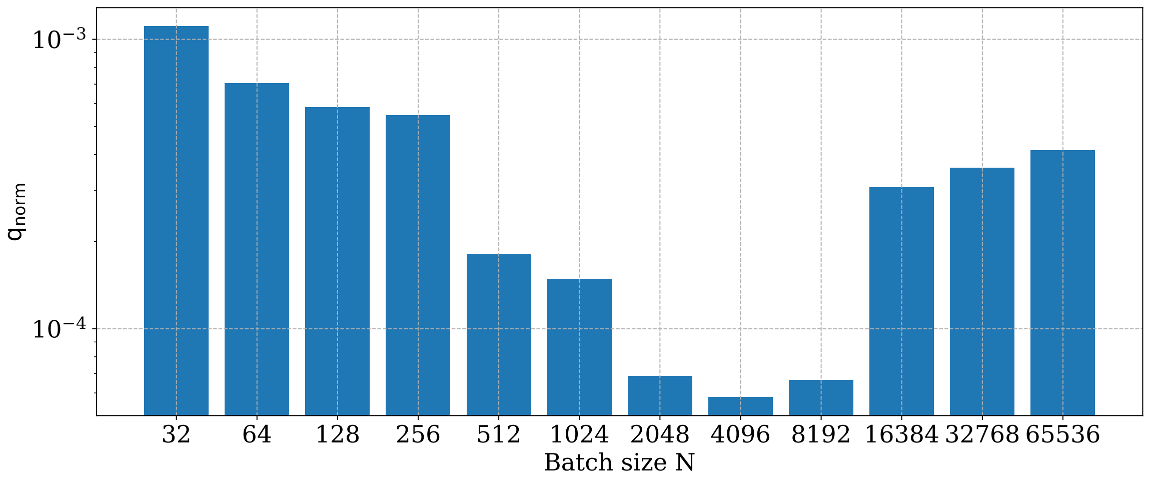


Figure 6.1: Average $\mathrm{q}_{\mathrm{norm}}$ after $K_{\max} = 8$ corrections for $n = 16$, $m = 64$.

The second parameter to explain is a maximum number of simulated corrections $K_{\max}$. The idea of the Algorithm 15 is to find parameters of (6.1) such that

$$\frac{1}{N}\sum_{j=1}^{N} \mathrm{q}_{\mathrm{norm}}\big(e^{i\Phi_j^{(K_{\max})}}, X_j^{(K_{\max})}\big) \approx 0. \tag{6.3}$$

Let $K_{\max} = 1$, then the Algorithm 15 learns the model to find a nearly exact solution of a phase retrieval problem (5.2). However, it was observed empirically that for large n the problem (5.2) is difficult to solve (see Figure 6.2).

That is why it is proposed to update the parameters of model (6.1) at each of the k-th correction for $K_{\max} > 1$ in order to improve the approximation of a solution for (5.2). The exact value of $K_{\max}$ depends on both number of beams n and number of measurements m. The dependence is such that if n increases, required $K_{\max}$ increase also. However, if m is increased for constant n, then $K_{\max}$ decreases. It was observed numerically that $K_{\max} = 9$ is sufficient for training up to $n = 128$ and $m = 4n$. (see Figure 6.3).

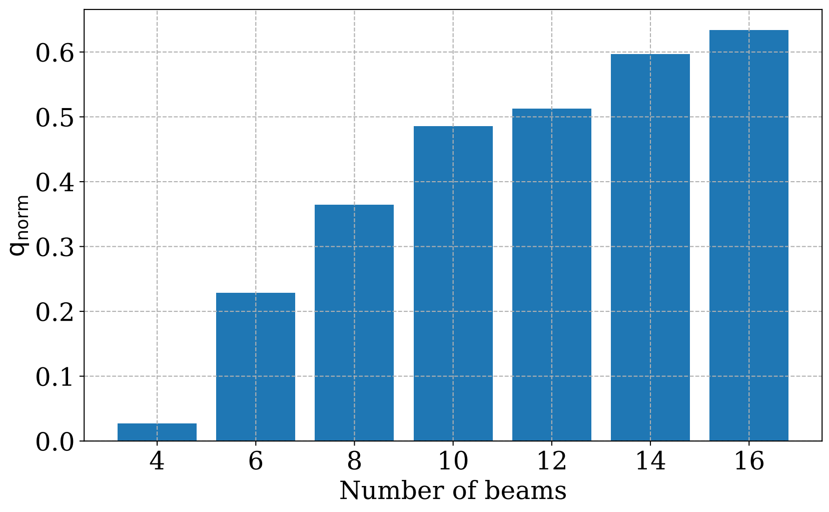


Figure 6.2: Average q_{norm} after $K_{max} = 1$ corrections for different n.

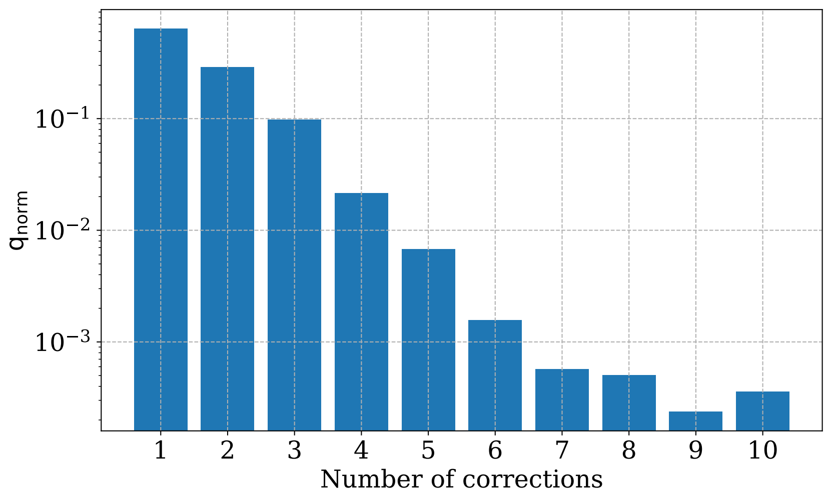


Figure 6.3: Average q_{norm} after different K_{max} corrections for $n = 16$, $m = 64$.

The number of learning iterations I_{max} is also an empirical value which is set to $I_{max} = 1000$ in the experiments. However, if n is large and it is observed that the value of q_{norm} still decreasing, then I_{max} must be increased. The Algorithm 15 also requires a target $\widehat{x}$ as an input parameter. This point is crucial for understanding because just considering a fixed target the learning process converges. The trained model (6.1) can be seen as an approximation of the inverse of $x \to |Ax|$ in the neighborhood of $\widehat{x}$ which is formulated in the Conjecture 1. This leads to the fact that if the target is changed, then the model (6.1) must be retrained.

Conjecture 1 *Let $\mathcal{F}(x) : \mathbb{C}^n \to \mathbb{R}^m_+$ be such that $\mathcal{F}(x) = |Ax|$. Let us consider an abstract inverse function $\mathcal{F}^{-1}(x) : \mathbb{R}^m_+ \to \mathbb{C}^n$ which solves (5.2) exactly. Let $NN(x)$ be a function of the form (6.1) trained with the Algorithm 15 for a fixed target $\widehat{x} \in \mathbb{C}^n$. Let $B_\varepsilon(\widehat{x}) \subset \mathbb{C}^n$ be a ball centered at $\widehat{x}$ with radius $\varepsilon > 0$. Then*

$$q_{norm}\big(x, e^{i\,TFNN(b)}\big) \approx 0 \text{ where } b = \mathcal{F}(x) \text{ for any } x \in B_\varepsilon(\widehat{x}).$$

Remark 2 *It was observed numerically that if $|\widehat{x}|$ is not a constant then the sufficiently small values of q_{norm} can not be reached during training. (see Figure 6.4).*

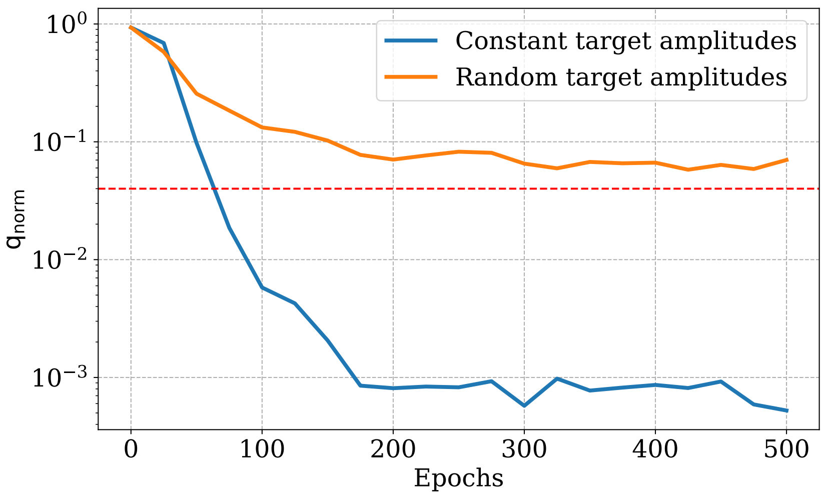


Figure 6.4: Average $\mathrm{q}_{\mathrm{norm}}$ after different $K_{\max}$ corrections for $n = 16$, $m = 64$, where red dashed line is a required threshold $\mathrm{q}_{\mathrm{norm}} = 0.04$.

The initial parameters W are distributed uniformly on the range $[-l, l]$ where $l = \sqrt{6/(m+n)}$. This is a standard way to initialize the parameters of a model which is called Glorot uniform initializer or Xavier uniform initializer [15].

In step 2 the batch of random signals $X = \{x_j\}_{j=1}^{N} \subset \mathbb{C}^{N\times n}$ is generated where $|x_j| = |\widehat{x}|$ and $\arg\left(x_{jk}\right) \sim \text{Uniform}(-\pi, \pi)$.

In step 3 the vectors of intensity measurements $b_j \in \mathbb{R}_{+}^{m}$ for each x_j are computed. This step can be performed either numerically or experimentally. For numerical measurements, a model like (1.2) is required. The advantage of this approach is a computational speed. The disadvantage is that the model must be built. For experimental measurements, a fast phase modulation physical device is required. The advantage is that we do not need a model. The disadvantage is that in practice to generate batch of signals $X \in \mathbb{C}^{N\times n}$ a Spatial Light Modulator (SLM) is used. This devise has low phase modulation speed around 500 milliseconds, which significantly slows down the learning process.

In step 4 the vectors of phase corrections $\varphi_j \in [-\pi, \pi]^n$ are computed for each vector of measurements $b_j \in \mathbb{R}_{+}^{m}$. There is a link between a vector of phases φ obtained in step 4 of the Algorithm 15 and $\widetilde{x}$ in step 3 of the Algorithm 14 is such that $\varphi = \arg(\widetilde{x})$.

In step 5 the gradients are computed separately with respect to real and imaginary parts of parameters $W \in \mathbb{C}^{m\times n}$. Despite the fact that the matrix is complex and potential application of Wirtinger calculus (see Section 2.3) to compute a complex-valued gradient is possible, the real-valued gradient is used instead. The motivation is that it was observed empirically that the convergence of the learning process is better with a real-valued gradient (see Figure 6.5)

The explicit formula of gradient is not given because it is computed by means of automatic differentiation algorithm implemented in Tensorflow library.

In step 6 the parameters W are updated using a real-valued gradient and following the update rule defined in [18, Algorithm 1] with default parameters $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$.

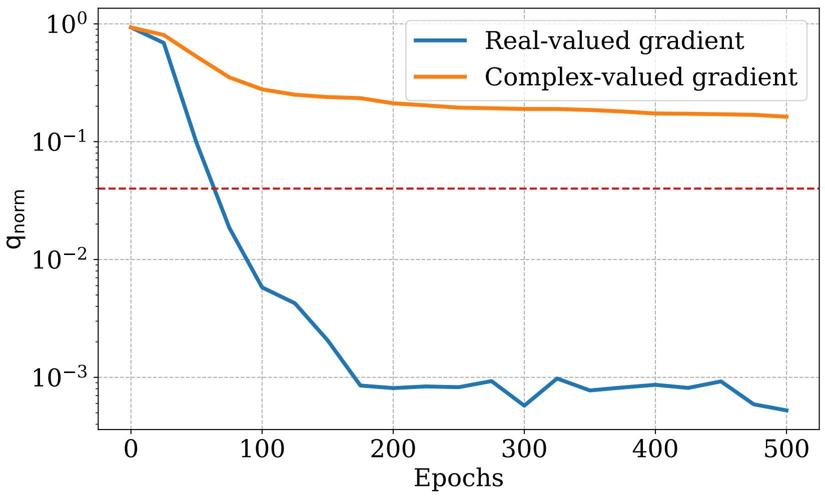


Figure 6.5: Average q_{norm} after $K_{\max} = 8$ corrections for $n = 16$, $m = 64$ during training, where red dashed line is a required threshold $q_{\text{norm}} = 0.04$.

In step 7 the phase corrections φ_j are applied for each x_j. In (6.2) the subtraction is considered between $\arg(\widehat{x}) \in [-\pi, \pi]^{1\times n}$ and rows of Φ

$$\arg(\widehat{x}) - \Phi = \begin{pmatrix} \arg(\widehat{x}) - \varphi_{1\cdot} \\ \dots \\ \arg(\widehat{x}) - \varphi_{N\cdot} \end{pmatrix}.$$

In step 8 it is checked if the maximum number of corrections $K_{\max}$ is reached. If it is true, then a new batch $X \in \mathbb{C}^{N\times n}$ is generated.

Also, there are several important properties of (6.1) which is necessary to mention. Let us show that the function (6.1) is invariant to changes of the absolute values of the measured signal.

Observation 9 *Let $\lambda > 0$ and $x \in \mathbb{R}^m$, then $TFNN(\lambda x) = TFNN(x)$.*

Proof. Let $W \in \mathbb{C}^{n\times m}$, then by linearity $\arg(W(\lambda x)) = \arg(\lambda W x) = \arg(Wx)$[1].

□

Observation 10 *Let $\lambda > 0$, $A \in \mathbb{C}^{m\times n}$, and $x \in \mathbb{C}^n$, then $|A(\lambda x)| = \lambda |Ax|$.*

Proof. $|A(\lambda x)| = |\lambda A x| = |\lambda||Ax| = \lambda |Ax|$. □

Proposition 14 *Let $x \in \mathbb{C}^n$ be such that $|x| = \lambda$ for $\lambda > 0$. Let $A \in \mathbb{C}^{m\times n}$. Then*

$$TFNN\big(|Ax|\big) = TFNN\big(|Ae^{i\arg(x)}|\big).$$

[1]Geometrically, it can be considered that the angle of vector does not change if we multiply it by the positive constant.

Proof. Using the Observation 10 we have that $\mathrm{TFNN}\big(|A(\lambda e^{i\arg(x)})|\big) = \mathrm{TFNN}\big(\lambda|Ae^{i\arg(x)}|\big)$, and then with the Observation 9 we get $\mathrm{TFNN}\big(\lambda|Ae^{i\arg(x)}|\big) = \mathrm{TFNN}\big(|Ae^{i\arg(x)}|\big)$ which shows the invariance to a constant amplitude. $\square$

For some applications, it is possible that the transmission matrix must be normalized as it is defined in Definition 5 to remove the bias which comes from the physical system. Thus, it is important to be normalization invariant for a phase correction model.

Observation 11 *Let $A \in \mathbb{C}^{m\times n}$ be a transmission matrix. Then, $|(A \odot e^{-i\arg(a_{\cdot 1})})x| = |Ax|$ for any $x \in \mathbb{C}^n$.*

Proof. Let us consider the j-th row of A and show that $|(a_j e^{-i\arg(a_{j1})})^\top x| = |a_j^\top x|$.

$$\begin{aligned}
|(a_j e^{-i\arg(a_{j1})})^\top x| &= \left|\sum_{k=1}^{n} a_{jk} e^{-i\arg(a_{j1})} x_k\right| \\
&= \left|e^{-i\arg(a_{j1})}\sum_{k=1}^{n} a_{jk} x_k\right| \\
&= \left|\sum_{k=1}^{n} a_{jk} x_k\right| \\
&= |a_j^\top x|.
\end{aligned}$$

Applying this result to each row we finish the proof. $\square$

Proposition 15 *Let $A \in \mathbb{C}^{m\times n}$ be a transmission matrix and let $\widehat{A}$ its normalized form (see Definition 5 in Appendix A). Then TFNN$(|\widehat{A}x|)$ = TFNN$(|Ax|)$.*

Proof. Applying the Observation 9 and the Observation 11 we obtain

$$\begin{aligned}
\mathrm{TFNN}(|\widehat{A}x|) &= \mathrm{TFNN}\left(\left|\frac{1}{\|A\|_\infty}(A \odot e^{-i\arg(a_{:1})})x\right|\right) \\
&= \mathrm{TFNN}\left(\left|(A \odot e^{-i\arg(a_{:1})})x\right|\right) \\
&= \mathrm{TFNN}(|Ax|).
\end{aligned}$$

$\square$

By the Proposition 15 we conclude that the model (6.1) is invariant to a transmission matrix normalization.

To obtain a full picture regarding the capabilities of TFNN, several additional information slices are presented in Figure 6.6.

It was numerically observed that in order to achieve a sufficiently low $\mathrm{q}_{\mathrm{norm}}$, say $\mathrm{q}_{\mathrm{norm}} < 0.04$, there is a minimal required ratio m/n for different n, which equals $m = 4n$. When the beam count varies from 4 to 128, the required m/n ratio increases from 2 to 12. Thus, it is important to show a minimal required ratio m/n for different n to achieve a sufficiently low $\mathrm{q}_{\mathrm{norm}}$. Different TFNNs were trained for the various number of beams $n \in \{4, 8, 16, 32, 64, 128\}$ and the different ratios between the number of measurements and the number of beams $m/n \in \{2, 4, 6, 8, 10, 12\}$.

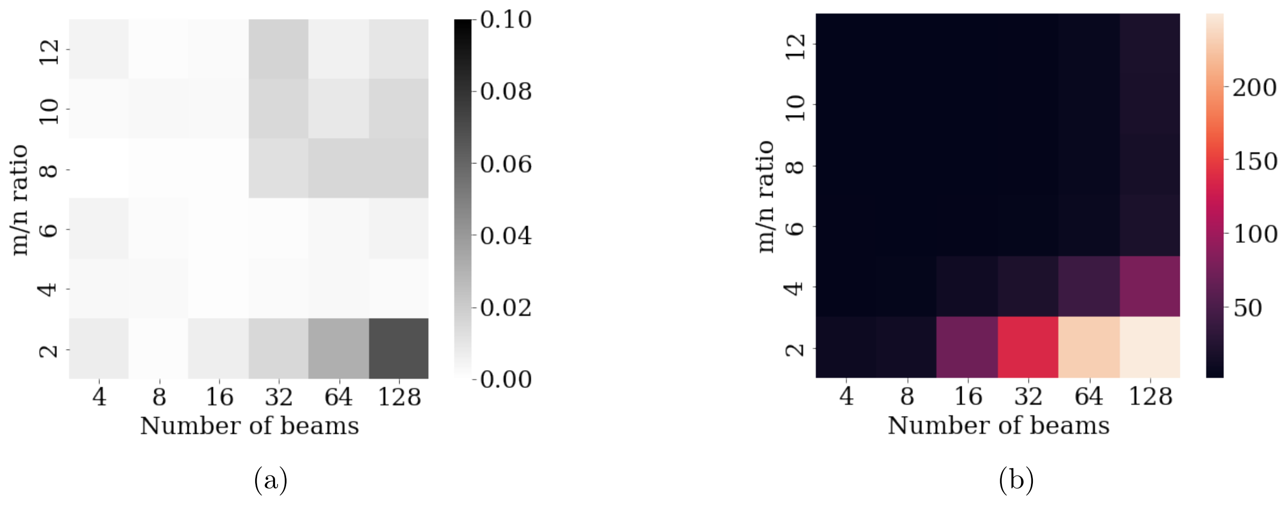


Figure 6.6: (a) Heat maps of the minimal achievable mean q_{norm} in grey scale and (b) its required relative training time. The relative time on (b) is computed by dividing a learning time in seconds for each n and m/n by the minimal time to obtain GPU invariant information. The minimal time required by the GPU used in for this experiments was 1.3 s.

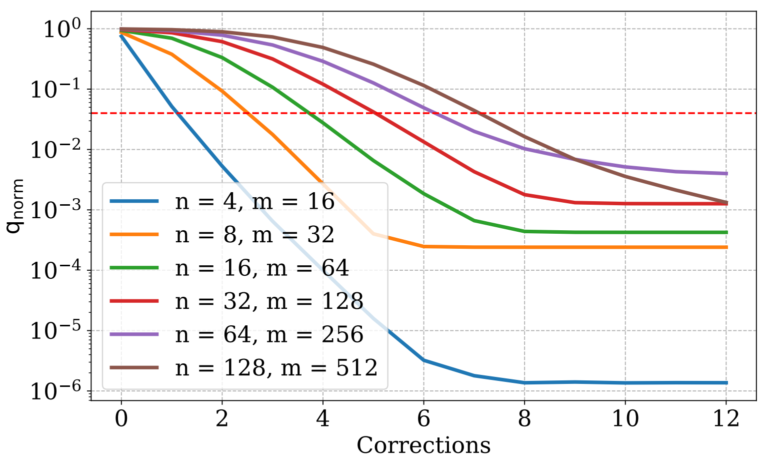


Figure 6.7: Mean q_{norm} behavior during corrections.

The minimal achievable q_{norm} was recorded and visualized as a heat map in Figure 6.6a, with the corresponding relative training time shown in Figure 6.9b. The minimal achievable q_{norm} is obtained by solving 1000 phase correction problems with fixed targets for each combination of n and m/n, and computing average of the q_{norm} at the last correction. Also, an average q_{norm} during corrections is visualized in Figure 6.7, which gives the information about the number of corrections required to achieve a target.

6.3 Target adaptive neural network

In the previous section it is shown how to train a neural network to use in the phase correction loop with a fixed target. However, it is impossible to retrain the model (6.1) fast when this change appears. In this section, we present the neural network to use in the phase correction loop where a target changes with a high frequency. The matrix W in (6.1) implicitly depends on $\widehat{x}$, which means that there exists a mapping $\text{TANN} : [-\pi,\pi]^n \to \mathbb{C}^{n\times m}$. Let us construct this mapping in the following form

$$\text{TANN}(x) = \text{Reshape}(Ue^{ix}), \tag{6.4}$$

where $U \in \mathbb{C}^{nm\times n}$ is a complex matrix of trainable parameters and $\text{Reshape} : \mathbb{C}^{mn} \to \mathbb{C}^{n\times m}$ is a matrix from vector building operator. The approach to learn (6.4) is presented in the Algorithm 16.

Algorithm 16: QRL algorithm for TANN

```
1. Initialization: Let $M \in \mathbb{N}$ be the size of the batch for signals and $N \in \mathbb{N}$ for targets. Let $K_{\max} \in \mathbb{N}$ be a number of corrections. Let $I_{\max} \in \mathbb{N}$ be a number of iterations. Let $U \in \mathbb{C}^{nm\times n}$ be some random matrix. Set $l = 0$.
2. Data generation: Generate a batch of initial targets $\widehat{X} \in \mathbb{C}^{N\times n}$ and initial phases $\{X_j^{(0)}\}_{j=1}^N \in \mathbb{C}^{N\times M\times n}$, such that $|x_{jp}^{(0)}| = |\widehat{x}_j| = c$ for $j \in \{1,\dots,N\}$, $p \in \{1,\dots,M\}$, $c > 0$. Set $k = 0$
3. Models retrieval: $W_j = \text{TANN}(\arg(\widehat{x}_j))$ for $j \in \{1,\dots,N\}$ ($W_j \in \mathbb{C}^{n\times m}$).
4. Magnitude measurement: Let $B_j^{(k)} = |X_j^{(k)}A^\top|$ for $j \in \{1,\dots,N\}$ ($B_j^{(k)} \in \mathbb{R}^{M\times m}$).
5. Phase correction: Let $\Phi_j^{(k)} = \arg(B_j^{(k)}W_j^\top)$ for $j \in \{1,\dots,N\}$ ($\Phi_j^{(k)} \in [-\pi,\pi]^{M\times n}$).
6. Gradients computation: $g = \frac{1}{NM}\sum_{j=1}^N\sum_{p=1}^M \nabla_U\, \text{q}_{\text{norm}}\big(e^{i\Phi_{jp}^{(k)}}, X_{jp}^{(k)}\big)$
7. Parameters update: Update $U$ applying the Adam update rule [18] with parameters $\alpha = \beta_1 = \beta_2 = 0.1$.
8. Phase modulation: Modify the phases of input signal according to
       $$X_j^{(k+1)} = X_j^{(k)} \odot \exp(i(\arg(\widehat{X}_j) - \Phi_j^{(k)})), \quad j \in \{1,\dots,N\}. \tag{6.5}$$
9. Return correction loop: Set $k = k + 1$. If $k < K_{\max}$ then go to 3.
10. Return iteration loop: Set $l = l + 1$. If $l < I_{\max}$ go to 2.
```

Let us explain each step of the Algorithm 16 in details. To simplify the notations for explanations of steps 2-8 let us fix k and remove it from the notations, and consider that $j \in \{1,\dots,N\}$, $p \in \{1,\dots,M\}$ where it is not explicitly specified.

In step 1 the parameters M, N, $K_{\max}$, $I_{\max}$ and U are initialized. The information about $K_{\max}$ and $I_{\max}$ is presented in parts in the explanation of the Algorithm 15. The parameters N and M represent the size of training batch. In contrast to the Algorithm 15, it is necessary to generate not only initial signals to correct, but also targets. The parameter N represents the number of targets in a generated batch. The parameter M represents the number of initial signals generated for each of N targets. The exact values that are used in computations are $N = 1024$, $M = 256$. Matrix $U \in \mathbb{C}^{mn \times n}$ is initialized such that the entries of real and imaginary parts are distributed by a standard normal law.

In step 2 the sets of target signals $\widehat{X} \in \mathbb{C}^{N \times n}$ and initial signals $X \in \mathbb{C}^{N \times M \times n}$ are generated such that $|\widehat{X}| = |X_j| = c_j$ for some constant $c_j > 0$, $\arg(X_{jpq})$, $\arg(\widehat{X}_{jq})$ are distributed uniformly on range $[-\pi, \pi]$ for $q \in \{1, \cdots, n\}$.

In step 3 matrices $W_j \in \mathbb{C}^{n \times m}$ are computed for each target signal $\widehat{x}_j \in \mathbb{C}^n$ by means of model (6.4).

In step 4 the measurements $B_j \in \mathbb{R}^{M \times m}$ for each $X_j \in \mathbb{C}^{M \times n}$ are performed. The same as in the explanation of Algorithm 15, this step can be performed either numerically or experimentally.

In step 5 the phase corrections $\Phi_j \in [-\pi, \pi]^{M \times n}$ are computed for each $B_j \in \mathbb{R}_+^m$ by means of matrices $U_j \in \mathbb{C}^{n \times m}$.

In step 6 the gradients are computed. Note, that Wirtinger calculus is used to get g in contrast to the Algorithm 15. It was observed numerically that the real-valued gradient with respect to real and imaginary parts of U is less informative for optimization (see Figure 6.8).

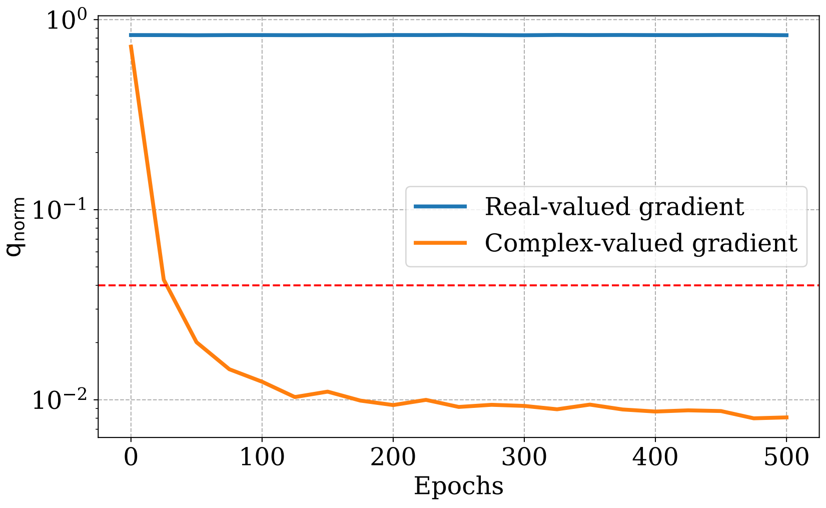


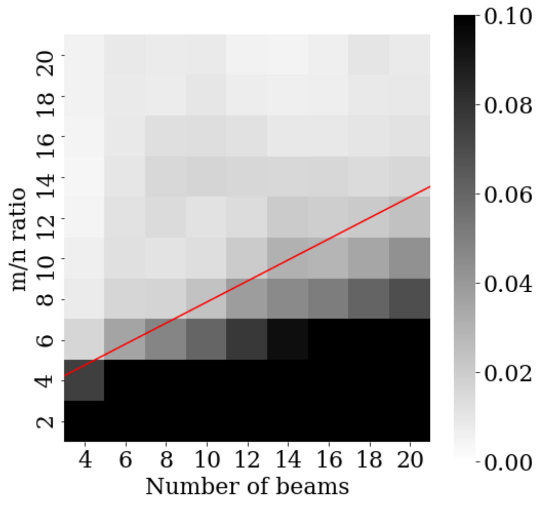
Figure 6.8: Average q_{norm} after $K_{\max} = 8$ corrections for $n = 6$, $m = 64$.

In step 7 the matrix W is updated. Note, that the default parameters of Adam update rule [18] are changed to $\alpha = \beta_1 = \beta_2 = 0.1$. It was observed empirically that the optimization with complex-valued gradient is faster with these parameters.

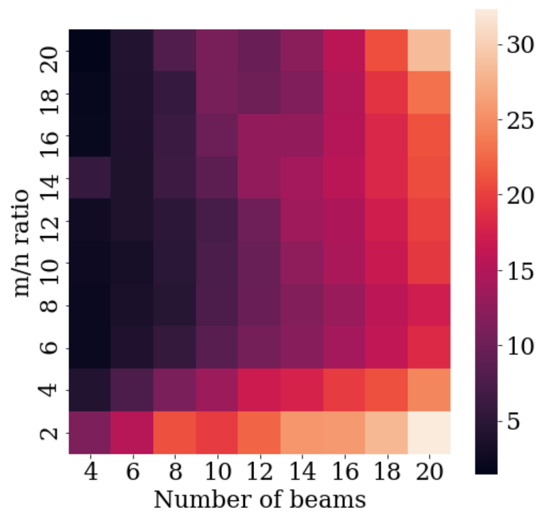
In step 8 phases of each $X_j \in \mathbb{C}^{M \times n}$ are corrected with $\Phi_j \in [-\pi, \pi]^{M \times n}$.

To obtain a full picture regarding the capabilities of TANN, several additional information slices are presented in Figure 6.9. It was numerically observed that in order to achieve a sufficiently low q_{norm}, say $q_{\text{norm}} < 0.04$, there is a minimal required ratio m/n for different n. When the beam count varies from 4 to 20, the required m/n ratio increases from 4 to 12. Thus, it is important to show a minimal required ratio m/n for different n to achieve a sufficiently low q_{norm}. Different TANNs were trained for the various number of beams $n \in \{4, 6, 8, 10, 12, 14, 16, 20\}$ and the different ratios between the number of measurements and the number of beams $m/n \in \{2, 4, 6, 8, 10, 12, 14, 16, 18, 20\}$. The minimal achievable average q_{norm}

was recorded and visualized as a heat map in Figure 6.9a, with the corresponding relative training time shown in Figure 6.9b. The minimal achievable q_{norm} is obtained by solving 1000 phase correction problems with different targets for each combination of n and m/n.



(a)



(b)

Figure 6.9: (a) Heat maps of the minimal achievable mean q_{norm} in grey scale and (b) its required relative training time. The red line in (a) approximates the separation line for which $q_{\text{norm}} = 0.04$. The relative time on (b) is computed by dividing a learning time in seconds for each n and m/n by the minimal time to obtain GPU invariant information. The minimal time required by the GPU used in for this experiments was 13 s.

The red line in Figure 6.9a reveals the dependency between n and m/n to obtain $q_{\text{norm}} = 0.04$ and is defined as $f(n) = \frac{n}{2} + 1$. This gave us information about the minimal number of measurements needed to obtain $q_{\text{norm}} \leq 0.04$, which was $m = \frac{n^2}{2} + n$.

An average q_{norm} during corrections is visualized in Figure 6.10, which gives the information about the number of corrections required to achieve a target.

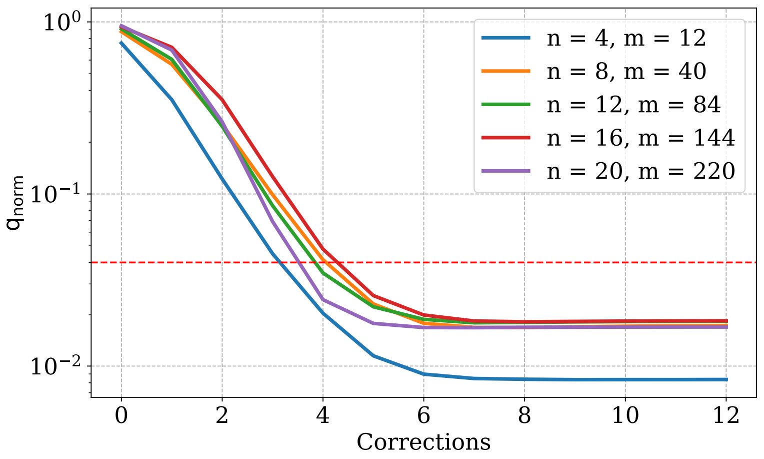


Figure 6.10: Mean q_{norm} behavior during corrections.

Chapter 7

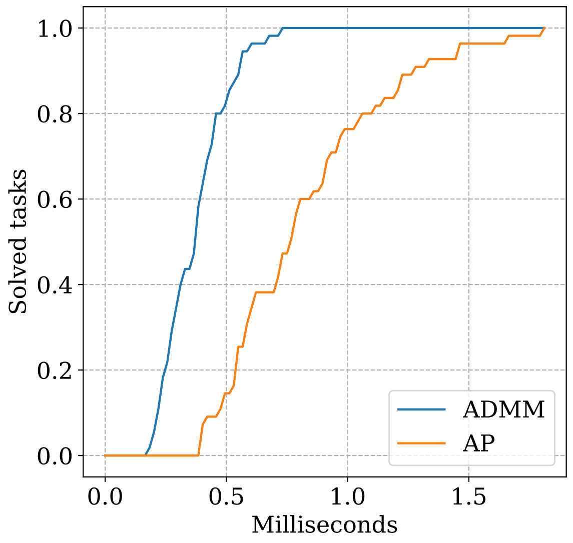
Experimental results

In this chapter we compare the efficiency and robustness of the phase retrieval methods used in the experimental phase correction loop. In this work, we developed two kinds of such methods: Adaptive Relaxed Alternating Directions Method of Multipliers (Algorithm 10) to solve a phase retrieval problem and Target Adaptive Neural Network (6.4) (learned by Algorithm 16) to solve directly a phase correction problem. Both of them were applied to correct $n = 16$ laser beams with $4n$ measurements for ADMM and $12n$ measurements for TANN. Both of them were compared with a baseline alternating projections algorithm (Algorithm 7). The initialization strategy was not used when ADMM and alternating projections algorithms were compared, which means that the same random starting point was used instead. The reason is that it was observed on Figure 5.1 that both algorithm have the same kind of profit from the spectral initialization.

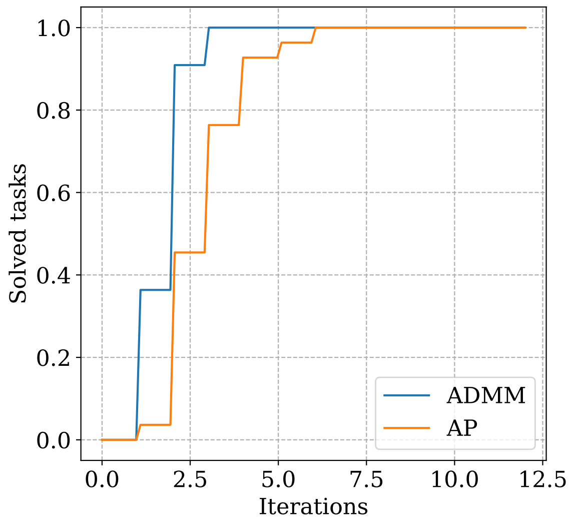
To compare the algorithms the profiling method (Section 2.5) is used with modifications. In the initial approach, we generate a set of problems (the set of transmission matrices) that are solved by selected methods. However, in practice, it is too complicated to perform such computations since around 10 minutes is required to collect experimental data $X \in \mathbb{C}^{N\times n}$ and $B \in \mathbb{R}_{+}^{N\times m}$ for $N = 400$ to compute a transmission matrix A. That is why we generate different initial state of lasers instead and then apply the phase correction loop for a fixed transmission matrix $A \in \mathbb{C}^{m\times n}$.

The transmission matrix was computed by the Algorithm 17 using unbiased data $B - \beta$, where β was computed by the Algorithm 18. The same matrix was used to train TANN in Algorithm 16.

The experimental profiles are presented on Figure 7.1 and Figure 7.3 reveal that both algorithms performs faster that the baseline algorithm and both can solve all problems. ADMM was applied for $m = 4n$ measurements with stopping tolerance 0.01 of $\mathrm{q}_{\mathrm{norm}}$ metric and 15 iterations maximum and solve 80% of all problems in 0.5 milliseconds where, at the same time, the alternating projections with stopping tolerance 0.001 of $\mathrm{q}_{\mathrm{norm}}$ metric solves 15%. The stopping tolerances were selected empirically in order to solve a phase correction problem as fast as possible. It was observed that these values are different for different algorithms. The speed of TANN is incomparably greater than for the alternating projections algorithm, which is obvious since one correction requires a single matrix multiplication. As it was shown on Figure 6.9a, TANN requires more measurements than optimization algorithms, which is a disadvantage. However, when this amount is available, than the highest speed of corrections can be achieved. The experimental setup that was built by PhD student Alexandre Boju and postdoctoral researcher Geoffrey Maulion is presented on Figure 7.5. The details about the experimental setup are described in the thesis of Alexandre Boju.

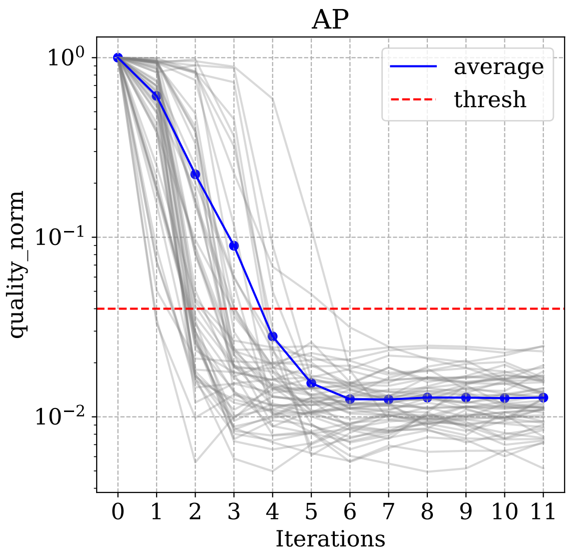


(a) Time profile

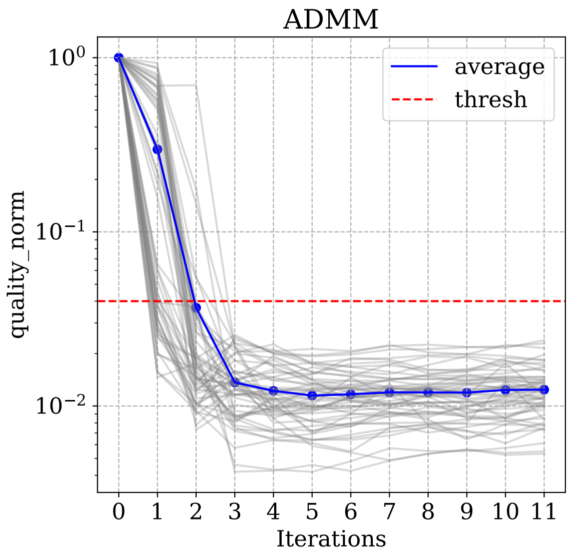


(b) Iteration profile

Figure 7.1: Profiles computed for $n = 16$, $m = 64$, 100 random starting lasers state.

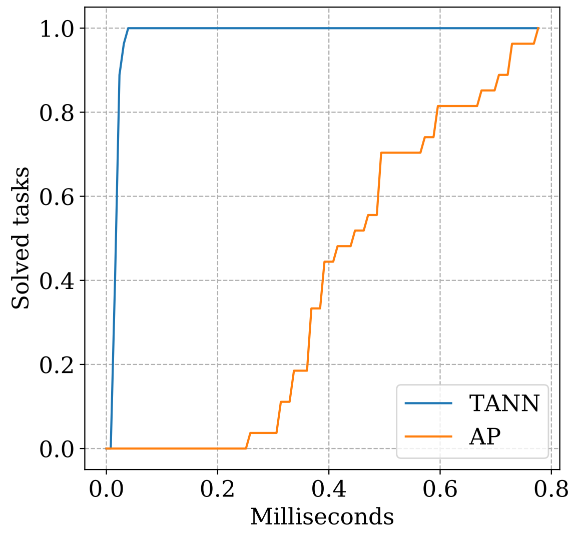


(a) Alternating projections

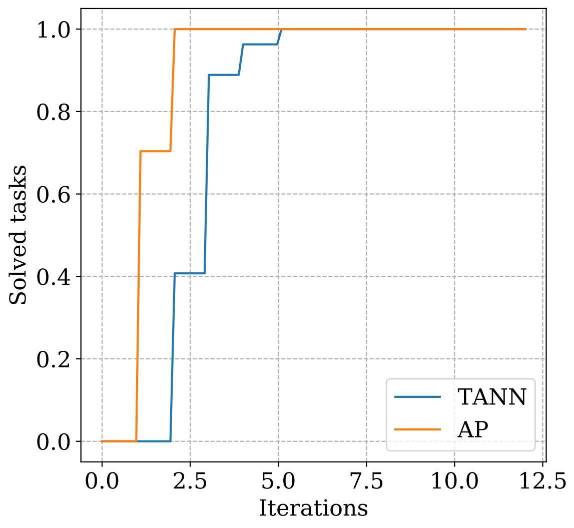


(b) ADMM

Figure 7.2: q_{norm} experimental traces computed for $n = 16$, $m = 64$, 100 random starting lasers state.

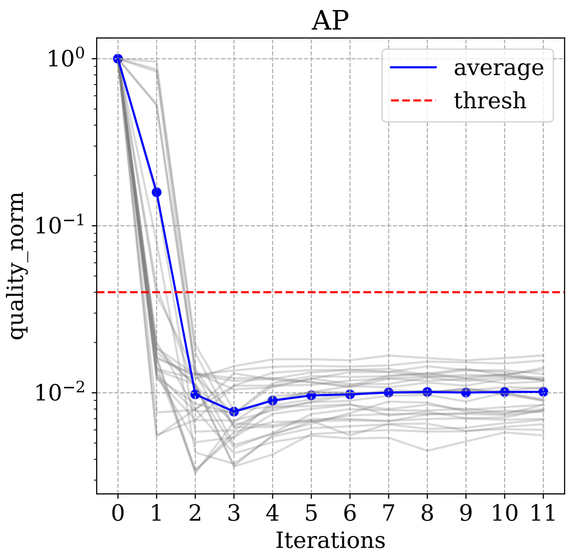


(a) Time profile

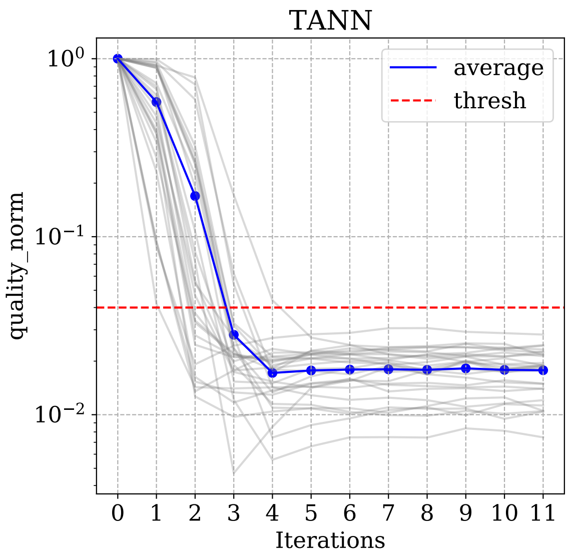


(b) Iteration profile

Figure 7.3: Profiles computed for $n = 16$, $m = 192$, 100 random starting lasers state.



(a) Alternating projections



(b) ADMM

Figure 7.4: q_{norm} experimental traces computed for $n = 16$, $m = 192$, 100 random starting lasers state.

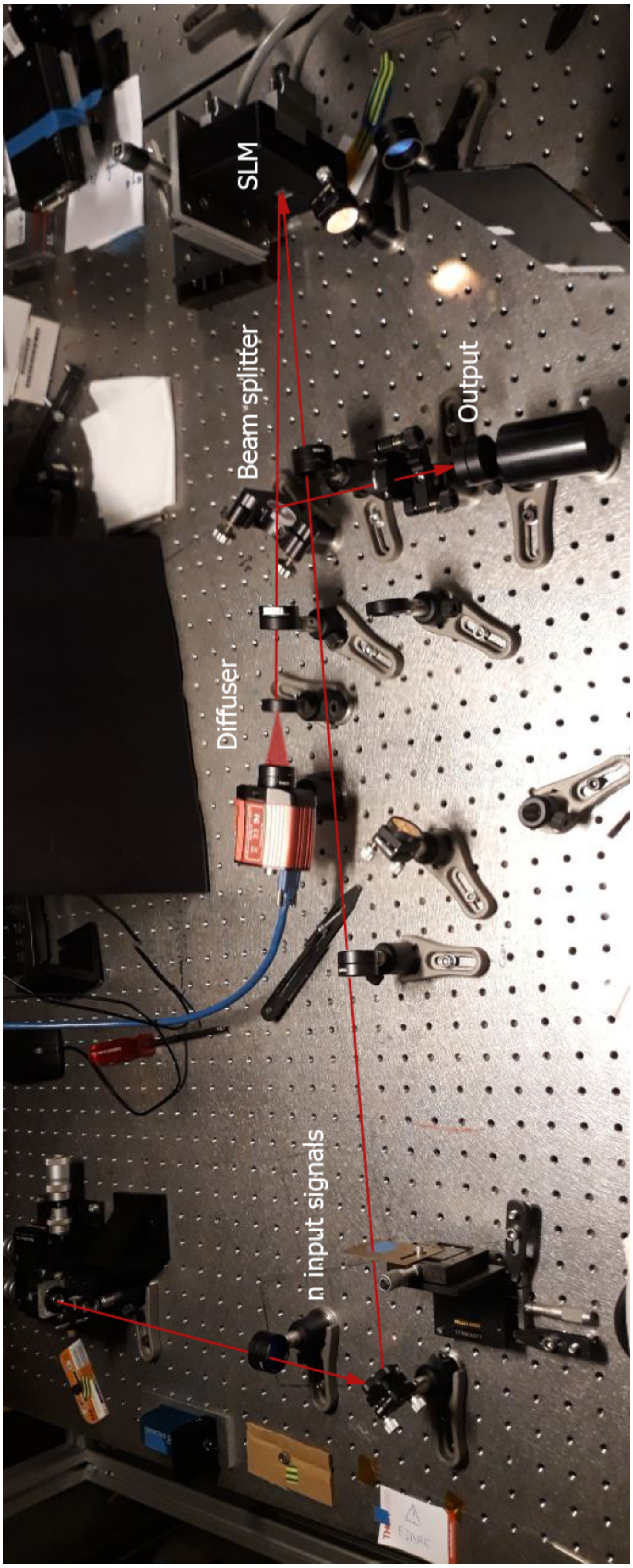


Figure 7.5: Experimental setup picture.

Conclusion and perspectives

Our main contributions in this work were first to develop the ADMM algorithm (Chapter 5) and to show that this algorithm can outperform the alternating projections method initially used to perform phase corrections in the context of physical experiments in the photonics laboratory (Chapter 7). In particular, we have observed and shown that the choice of the penalty parameter is sensitive for the efficiency of the algorithm and we have proposed adaptive updating rules that we believe are relevant. Our second main contribution was the development "from scratch" of a neural network approach (Chapter 6) for the phases control.

The first algorithm (ADMM) has been developed to solve the phase correction problem, but it can also be used to solve the phase retrieval problem (Algorithm 10). The updated formulas of an algorithm and its speed of convergence depends on the reformulation of the original system of nonlinear equations $|Ax| = b$. The proper reformulation was found, which allowed us to achieve a higher speed in solving a phase correction task. It was also shown how to select a regularization parameter ρ of the augmented Lagrangian to speed up the algorithm and, at the same time, maintain the ability to converge to a global solution frequently. To summarize, there are the following advantages and disadvantages of ADMM in comparison with the baseline Alternating Projections (AP) algorithm. The comparison is valid for a number of beams n up to 128 and for fixed $m = 4n$.

Advantages:

1. ADMM requires less time and iterations to solve both a phase retrieval and a phase correction problems.
2. ADMM converges to a global solution of a phase retrieval problem with higher probability than AP.

Disadvantages:

1. ADMM requires a proper selection of the parameter γ which is used to switch to the AP algorithm when it is near a solution of a phase retrieval problem. This must be used when there is noise in a transmission matrix.

The second algorithm solves a phase correction problem only and is based on the neural network. The main contribution is not the form of a model but the way of its training : Algorithm 15 for a fixed target phase and Algorithm 16 for a random target phase. Since a target phase changes with high frequency in the physical application we will consider model (6.4) only in this conclusion. Let us reveal the advantages and disadvantages of this approach.

Advantages:

1. Because of a simple form of the neural network (6.4), the corrections are performed with high speed. If we compare the profiles for experimental data (Figure 7.3), it will be clear that TANN is approximately 15 times faster than AP, which is the biggest advantage of this approach.
2. Figure 7.3 reveals that the neural network model is robust to noise.

Disadvantages:

1. To train TANN model (6.4) we require much more measurements than for the optimization algorithms. It can be clearly seen on Figure 6.9a.
2. TANN model has memory limitations. For example, to train a model for 20 beams, we require GPU with 6GB. Taking into account that raises is quadratic, this algorithm can not be applied for a huge number of beams.

To conclude, both methods perform faster than the alternating projection algorithm. However, there are limitations in memory for TANN, which makes this approach not applicable for $n > 20$, and additional parameter γ for the ADMM algorithm, which must be selected with respect to the noise level.

This conclusion leads us to the future perspectives of this work. Let us first specify them for the ADMM algorithm, and then for TANN.

From a practical point of view, ADMM requires the possibility to select the parameter γ. This can be easily done if the experimental data of signals $X \in \mathbb{C}^{N \times n}$ and measurements $B \in \mathbb{R}_+^m$ is available. For this aim, the standard deviation σ of noise must be estimated statistically and then $\gamma = 2\sigma$ (empirical result). However, the very last results from the physicists team give a possibility to compute a transmission matrix without X. This modification reduces the size of the laser system, which is important for real applications. That is why an approach for the computation of γ without X must be developed.

From a theoretical point of view, ADMM requires a convergence result. This must be done to fully understand the capabilities of this algorithm. However, since the original optimization problem is nonconvex and complex-valued, there are not a lot of available results in the literature, which makes this analysis very hard and time expensive. For instance, in [44] there is a result for the convergence of the alternating projection algorithm. However, the assumptions that must be satisfied to achieve the convergence are not possible to meet in the real-life (millions of measurements), which is also mentioned by the author. That is why it is a really challenging problem to solve.

Concerning the neural network approach, the biggest problem is a memory limitation. The quadratic raise of the trainable parameters is a huge disadvantage. In this context, the complex-valued convolutional layer could be potentially useful. It is already implemented in the Tensorflow extension library, which is used in this work, however, the proper way of its application was not found. That is why it is reasonable to work in this direction in the future.

The results about the neural network approach were published in [33, 36] and presented in the following conferences [35, 41, 34, 40, 39, 37, 38], where [41, 34, 40, 38] were invited conferences.

The Python implementations of all algorithms that are presented in this work can be downloaded from the following links:

- Implementation of the phase retrieval algorithms discovered in Chapter 4.
  https://gitlab.xlim.fr/shpakovych/phrt-opt
- Implementation of the Algorithm 14.
  https://gitlab.xlim.fr/shpakovych/phcr-opt
- Implementation of the Algorithm 15 and Algorithm 16.
  https://gitlab.xlim.fr/shpakovych/phcr-nn
- Implementation of the profiling method explained in Section 2.5.
  https://gitlab.xlim.fr/shpakovych/ph-profile
- Implementation of the TensorFlow extension library for complex-valued neural networks (Section 2.4). https://gitlab.xlim.fr/shpakovych/cvnn

Appendix A

Transmission matrix computational details

In this section, we give additional details on how a transmission matrix $A \in \mathbb{C}^{m\times n}$ can be computed to build a mathematical model (1.2), which then is used for phase corrections in Algorithm 14. As it is explained in the Introduction chapter, we obtain matrix A by solving the following problem

$$\text{Find } A \in \mathbb{C}^{m\times n} \text{ such that } |XA^\top| = B, \tag{A.1}$$

where the sets of signals $X \in \mathbb{C}^{N\times n}$ and measurements $B \in \mathbb{R}_+^{N\times m}$ are obtained experimentally. This problem can be split into m phase retrieval problems that can be solved by one of the methods presented in this work. In addition, there are no time constraints. For this goal, we select the alternating projections algorithm since it does not require any additional parameters that sensitive to the level of noise like γ in ADMM. Recall, that even with Gao & Xu initialization strategy [13] there is no guarantee that the alternating projections algorithm will find a global solution. Since this part is critical and directly affects the quality of the model, algorithm with restarts is considered, where the main idea is to change a starting point until a global solution is found.

However, it is not obvious how to determine the global solution. In practice, there is no A such that $|XA^\top| = B$ nearly exactly because of the presence of noise. The difficulty is that the level of noise is not known in advance and thus the choice of a threshold is not obvious. That is why, we propose to use the following statistical rule to answer this question. Let $\widetilde{A}$ be an approximate solution of $|XA^\top| = B$. Then the j-th row $\widetilde{a}_j$ of $\widetilde{A}$ is a global solution of $|Xa| = b_{\cdot j}$ if

$$\frac{\| |X\widetilde{a}_j| - b_{\cdot j}\|}{\|b_{\cdot j}\|} < \min\{0.2, \eta\}, \tag{A.2}$$

where $\eta = \bar{\mu} + 3s$, $\bar{\mu} = \mu + z_\alpha s/\sqrt{m}$,

$$\mu = \frac{1}{m}\sum_{j=1}^{m} \frac{\| |X\widetilde{a}_j| - b_{\cdot j}\|}{\|b_{\cdot j}\|}, \qquad s = \sqrt{\frac{1}{m}\sum_{j=1}^{m}\left(\frac{\| |X\widetilde{a}_j| - b_{\cdot j}\|}{\|b_{\cdot j}\|} - \mu\right)^2}.$$

Here, μ and s are sample mean and standard deviation of all m relative norms. $\bar{\mu}$ is an upper bound of a trust region for true value of μ with a critical value z_α of the standard normal distribution for the significance level α. We set $\alpha = 0.05$ which means that the true value of μ is inside the range $[\mu - z_\alpha s/\sqrt{m}, \mu + z_\alpha s/\sqrt{m}]$ with probability $P = 0.95$, where $z_\alpha = 1.96$. Finally, assuming that all relative norms are drawn from the normal distribution with parameters $\bar{\mu}$ and σ, we use a rule of 3-sigma which says that the sample ξ from this distribution is such that $\xi \in [\bar{\mu} - 3\sigma, \bar{\mu} + 3\sigma]$ with probability $P = 0.9973$.

This computations assumes that most of the approximations $\widetilde{a}_j$ are global solution and just some of them are local. This assumption is reasonable since it was shown numerically that with Gao & Xu initialization more than 90% of all problems can be solved by means the alternating projections algorithm. However, to be sure that the threshold is not too high we set a limitation on the level of 0.2 of relative norm. Then, the algorithm of finding $\widetilde{A}$ can be formulated in Algorithm 17.

Algorithm 17: Transmission matrix computation.

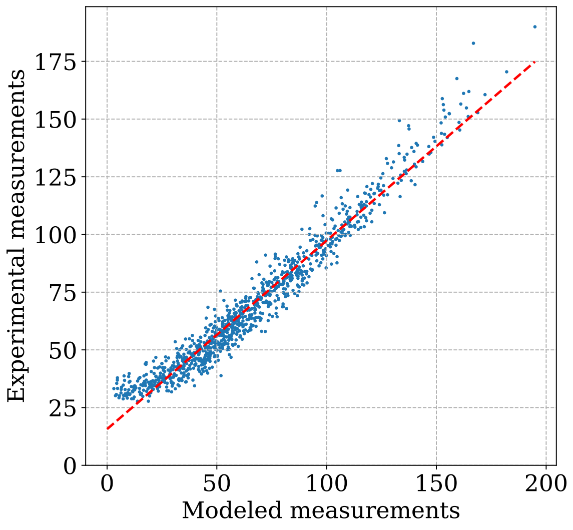
```
   Input: Signals $X \in \mathbb{C}^{N\times n}$ and measurements $B \in \mathbb{R}_+^{N\times m}$, tolerance $\varepsilon > 0$, maximal
          number of restarts $R_{\max} \in \mathbb{N}\backslash\{0\}$.
   Output: An approximation of the transmission matrix $\widetilde{A} \in \mathbb{C}^{m\times n}$.
 1 Initialize each row of matrix $\widetilde{A}^{(0)} \in \mathbb{C}^{m\times n}$ by solving $m$ phase retrieval problems for $X$
     and $B$ with the starting point defined by Gao & Xu initialization [13];
 2 Set $k = 0$;
 3 for $k < R_{\max}$ do
 4     Find set $J$ such that for all $j \in J$ condition (A.2) is not satisfied;
 5     if $J$ is empty then
 6         break
 7     Find an approximate solution $\widetilde{a}_j \in \mathbb{C}^n$ of equations $|Xa| = b_{\cdot j}$ for all $j \in J$ by
         means of Algorithm 7, where the starting points are generated randomly;
 8     Build the matrix $\widetilde{A}^{(k+1)}$ where its rows $\widetilde{a}_j^{(k+1)}$ are suth that
           $$\widetilde{a}_j^{(k+1)} = \begin{cases} \widetilde{a}_j^{(k)} & \text{if } j \notin J \text{ or } \frac{\||X\widetilde{a}_j| - b_{\cdot j}\|}{\|b_{\cdot j}\|} > \frac{\||X\widetilde{a}_j^{(k)}| - b_{\cdot j}\|}{\|b_{\cdot j}\|}, \\ \widetilde{a}_j & \text{otherwise} \end{cases}$$
       Set $k = k + 1$;
 9 return $\widetilde{A} = \widetilde{A}^{(k)}$.
```

Note, that this algorithm gives no guarantee that rows $\widetilde{a}_j$ of $\widetilde{A}$ for $j \in \{1, \dots, m\}$ are global solutions of respective phase retrieval problems $|Xa| = b_{\cdot j}$. The are two reasons. First, we fix the maximal number of restarts $R_{\max}$, which is an empirical value and can be wrongly defined by user. Second, the maximal value of threshold 0.2 is also an empirical value, which means that we do not assume that the level of noise in measurements is greater than 20%. This can be not true in practice, however consequently it means that the measurements are too noisy and the model will have a low quality, which will lead to poor phase corrections. That is why, there must be warnings that the program can throw to inform user about problems with matrix computation and suggest possible solutions: either to increase $R_{\max}$ or to inform about a high presence of noise.

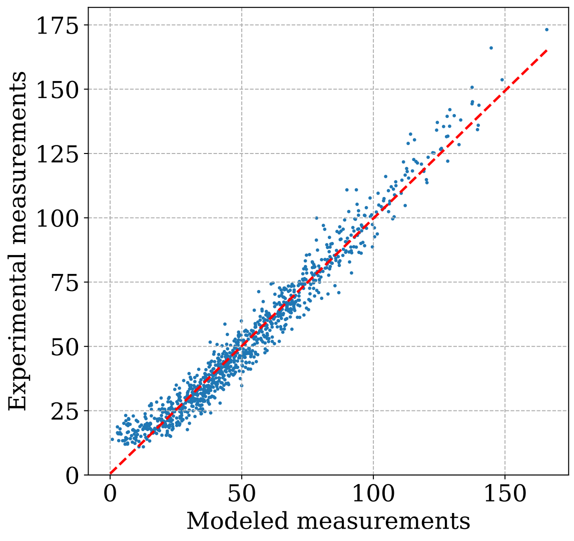
When the transmission matrix is retrieved we can compute an error ϵ between experimental and modeled measurements as

$$\epsilon = \frac{\||X\widetilde{A}^\top| - B\|}{\|B\|}, \tag{A.3}$$

where a typical value is $\epsilon \leq 0.15$. Another way of comparison can be done by meas of 2-dimensional plot where the first dimension represents a modeled measurement $\left(|X\widetilde{A}^\top|\right)_{jk}$ and the second dimension represents an experimental measurement B_{jk} for $j \in \{1, \dots, N\}$, $k \in \{1, \dots, m\}$ (Figure A.1a). Despite the fact that the model assumes a linear dependence between near and far complex fields, on Figure A.1a we observe a bias in data (minimal experimental measurement is not near zero). It is reasonable to remove it in order to obtain a linear



(a) Biased measurements



(b) Unbiased measurements

Figure A.1: Modeled and experimental measurements comparison.

dependence. For this goal, we present the Algorithm 18 which can compute the shift value, which must be applied to all measurements before transmission matrix computation.

Algorithm 18: Bias computation.

```
  Input: Signals $X \in \mathbb{C}^{N\times n}$ and measurements $B \in \mathbb{R}_+^{N\times m}$, tolerance $\varepsilon > 0$.
  Output: Bias $\beta \in \mathbb{R}_+$.
1 Set $\beta = 0$, $\beta^{(k)} = +\infty$, $k = 0$, $B_{\text{unb}} = B$;
2 while $\beta^{(k)} > \varepsilon$ do
3     Compute $\widetilde{A}^{(k)}$ using Algorithm 17 for $X$ and $B_{\text{unb}}$;
4     Set $B_{\text{mod}}^{(k)} = |X\widetilde{A}^{(k)}|$;
5     Find parameters $\alpha^{(k)}$ and $\beta^{(k)}$ of a regression model $B_{\text{unb}}(:) = \alpha^{(k)}B_{\text{mod}}^{(k)}(:) + \beta^{(k)}$;
6     Set $\beta = \beta + \beta^{(k)}$;
7     Set $B_{\text{unb}} = B - \beta$;
8     Set $k = k + 1$;
9 return $\beta$.
```

Then, to obtain an unbiased transmission matrix we first compute β by Algorithm 18, and then compute $\widetilde{A}$ by Algorithm 17 for X and $B - \beta$ input data. Note, that to use this model, we require subtract β from experimental measurements each time when they are performed. The result of unbiasing can be observed on Figure A.1b. Also, the value of (A.3) can be compared. On Figure A.2 it can be clearly seen that unbiasing technique improves the quality

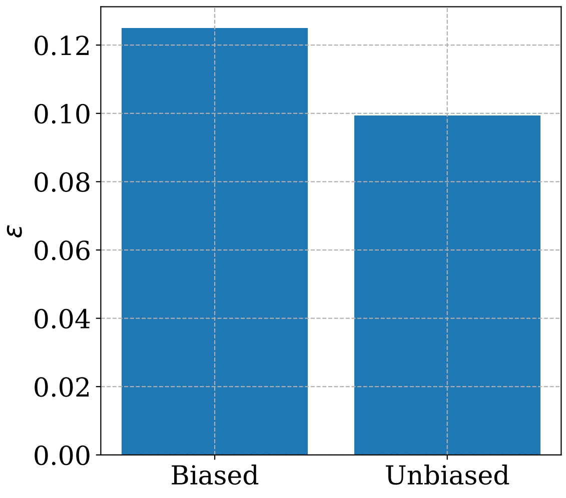
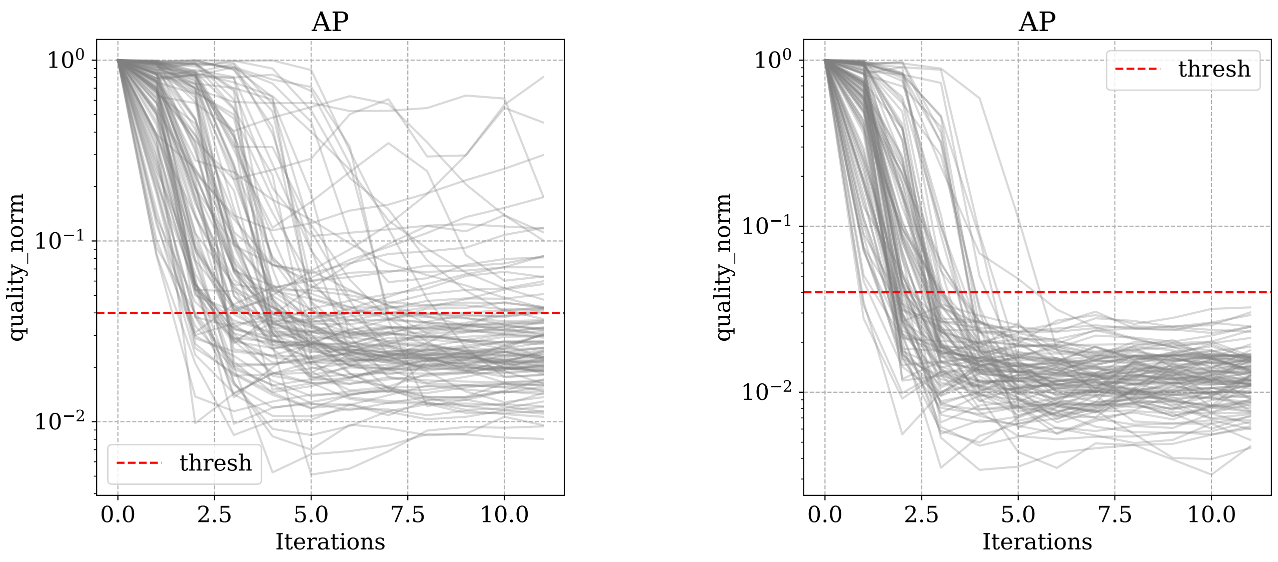


Figure A.2: Comparison of (A.3) for biased and unbiased models using experimental data X and B for $n = 16$, $m = 64$ and $N = 1000$.

of a transmission matrix model. Another confirmation of the improvement can be observed on Figure A.3 which was built on the experimental data. Thus we can conclude that unbiasing approach improves the quality of phase corrections.



(a) Biased model (b) Unbiased model

Figure A.3: Traces of q_{norm} during experimental phase corrections performed by the alternating projections algorithm.

In practice, it is possible that the transmission matrix must be normalized as it is defined in Definition 5 to remove the bias which comes from the physical system. The way to do that is presented below.

Definition 5 *Let $A \in \mathbb{C}^{m \times n}$ be a transmission matrix. Then the normalized matrix $\widehat{A}$ computes as*

$$\widehat{A} = \frac{1}{\|A(:)\|_\infty} A \odot e^{-i \arg(a_{\cdot 1})}, \tag{A.4}$$

where $\|A(:)\|_\infty = \max_{i \in \{1,\dots,m\}, j \in \{1,\dots,n\}} |a_{ij}|$ and $a_{\cdot 1} = (a_{11}, a_{21}, \dots, a_{m1})^\top$ is the first column of A.

Bibliography

[1] R. Balan, Pete Casazza, and D. Edidin. On signal reconstruction without noisy phase. *arXiv: Functional Analysis*, 2004.

[2] Afonso S. Bandeira, Jameson Cahill, Dustin G. Mixon, and Aaron A. Nelson. Saving phase: Injectivity and stability for phase retrieval. *Applied and Computational Harmonic Analysis*, 37(1):106–125, 2014.

[3] J. A. Barrachina, C. Ren, C. Morisseau, G. Vieillard, and J.-P. Ovarlez. Complex-valued vs. real-valued neural networks for classification perspectives: An example on non-circular data. pages 2990–2994, 2021.

[4] Jonathan Barzilai and Jonathan M. Borwein. Two-Point Step Size Gradient Methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 01 1988.

[5] C. Beck and R. D'Andrea. Computational study and comparisons of lft reducibility methods. 2:1013–1017 vol.2, 1998.

[6] A. Brignon. *Coherent Laser Beam Combining.* Wiley-VCH, 2013.

[7] Emmanuel J. Candès, Xiaodong Li, and Mahdi Soltanolkotabi. Phase retrieval via wirtinger flow: Theory and algorithms. *IEEE Trans. Information Theory*, 61(4):1985–2007, 2015.

[8] Emmanuel J. Candès, Yonina C. Eldar, Thomas Strohmer, and Vladislav Voroninski. Phase retrieval via matrix completion. *SIAM Journal on Imaging Sciences*, 6(1):199–225, 2013.

[9] P. Chen, Albert Fannjiang, and Gi-Ren Liu. Phase retrieval with one or two diffraction patterns by alternating projections of the null vector. *Journal of Fourier Analysis and Applications*, 24, 06 2018.

[10] Yuxin Chen and Emmanuel Candes. Solving random quadratic systems of equations is nearly as easy as solving linear systems. 28, 2015.

[11] Aldo Conca, Dan Edidin, Milena Hering, and Cynthia Vinzant. An algebraic characterization of injectivity in phase retrieval. *Applied and Computational Harmonic Analysis*, 38(2):346–356, 2015.

[12] J. R. Fienup. Phase retrieval algorithms: a comparison. *Appl. Opt.*, 21(15):2758–2769, Aug 1982.

[13] Bing Gao and Zhiqiang Xu. Phaseless recovery using the gauss–newton method. *IEEE Transactions on Signal Processing*, PP:1–1, 08 2017.

[14] R. W. Gerchberg and W. O. Saxton. Practical algorithm for determination of phase from image and diffraction plane pictures. *OPTIK*, 35(2):237–&, 1972.

[15] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. 9:249–256, 13–15 May 2010.

[16] Akira Hirose. *Complex-Valued Neural Networks*. Wiley, 2013.

[17] Tianyue Hou, Yi An, Qi Chang, Pengfei Ma, Jun Li, Dong Zhi, Liangjin Huang, Rongtao Su, Jian Wu, Yinzhe Ma, and Pu Zhou. Deep-learning-based phase control method for tiled aperture coherent beam combining systems. *High Power Laser Science and Engineering*, 7, 11 2019.

[18] Jimmy Kingma, Diederik P. Lei Ba. Adam: A method for stochastic optimization. *Published as a conference paper at ICLR*, 2015.

[19] Ken Kreutz–Delgado. The complex gradient operator and the cr-calculus. *ArXiv e-prints*, June 2009.

[20] Junli Liang, Petre Stoica, Yang Jing, and Jian Li. Phase retrieval via the alternating direction method of multipliers. *IEEE Signal Processing Letters*, PP:1–1, 10 2017.

[21] Jian-Wei Liu, Zhi-Juan Cao, Jing Liu, Xiong-Lin Luo, Weimin Li, Nobuyasu Ito, and Long-Chuan Guo. Phase retrieval via wirtinger flow algorithm and its variants. pages 1–9, 07 2019.

[22] D. Luke. Phase retrieval, what's new? *SIAM SIAG/OPT*, 25:1–, 04 2017.

[23] Stefano Marchesini, Yu-Chao Tu, and Hau-Tieng Wu. Alternating projection, ptychographic imaging and phase synchronization. *Applied and Computational Harmonic Analysis*, 41(3):815 – 851, 2016.

[24] M. Mesbahi and G.P. Papavassilopoulos. On the rank minimization problem over a positive semidefinite linear matrix inequality. *IEEE Transactions on Automatic Control*, 42(2):239–243, 1997.

[25] Praneeth Netrapalli, Prateek Jain, and Sujay Sanghavi. Phase retrieval using alternating minimization. *IEEE Trans. Signal Processing*, 63(18):4814–4826, 2015.

[26] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, 2e edition, 2006.

[27] Scott W. Paine and James R. Fienup. Machine learning for improved image-based wavefront sensing. *Opt. Lett.*, 43(6):1235–1238, Mar 2018.

[28] Neal Parikh and Stephen Boyd. Proximal algorithms. *Found. Trends Optim.*, 1(3):127–239, January 2014.

[29] Atilim Güneş Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *ArXiv e-prints*, 2018.

[30] D. G. Sandler, T. K. Barrett, D. A. Palmer, R. Q. Fugate, and W. J. Wild. Use of a neural network to control an adaptive optics system for an astronomical telescope. *Nature*, 351(6324):300–302, May 1991.

[31] J. Saucourt, P. Armand, V. Kermène, A. Desfarges-Berthelemot, and A. Barthélémy. Random scattering and alternating projection optimization for active phase control of a laser beam array. *IEEE Photonics Journal*, 11(4):1–9, Aug 2019.

[32] Y. Shechtman, Y. C. Eldar, O. Cohen, H. N. Chapman, J. Miao, and M. Segev. Phase retrieval with application to optical imaging: A contemporary overview. *IEEE Signal Processing Magazine*, 32(3):87–109, May 2015.

[33] Maksym Shpakovych, Geoffrey Maulion, Alexandre Boju, Paul Armand, Alain Barthélémy, Agnès Desfarges-Berthelemot, and Vincent Kermene. On-demand phase control of a 7-fiber amplifiers array with neural network and quasi-reinforcement learning. *Photonics*, 9(4), 2022.

[34] Maksym Shpakovych, Geoffrey Maulion, Vincent Kermene, Alexandre Boju, Paul Armand, Agnès Desfarges-Berthelemot, and Alain Barthélemy. Phase control of a laser beam array, from optimization process to artificial intelligence. Advanced Fiber Laser Conference, Shengdu, China, Invited conference, 1-3 december 2021.

[35] Maksym Shpakovych, Geoffrey Maulion, Vincent Kermene, Alexandre Boju, Paul Armand, Agnès Desfarges-Berthelemot, and Alain Barthélemy. Scalable and fast phase control of laser beam array with quasi-reinforcement learning of a neural network in an error reduction loop: simulation and experiment. CLEO/Europe 2021, Munich, Allemagne, 20-24 Juin 2021.

[36] Maksym Shpakovych, Geoffrey Maulion, Vincent Kermene, Alexandre Boju, Paul Armand, Agnès Desfarges-Berthelemot, and Alain Barthélemy. Experimental phase control of a 100 laser beam array with quasi-reinforcement learning of a neural network in an error reduction loop. *Opt. Express*, 29(8):12307–12318, Apr 2021.

[37] Maksym Shpakovych, Geoffrey Maulion, Vincent Kermene, Alexandre Boju, Paul Armand, Agnès Desfarges-Berthelemot, and Alain Barthélemy. Machine learning algorithm applied to the phase control of an array of laser beams. IA POUR LES SCIENCES DE L'INGÉNIERIE, 28 et 29 juin, en ligne.

[38] Maksym Shpakovych, Geoffrey Maulion, Vincent Kermene, Alexandre Boju, Paul Armand, Agnès Desfarges-Berthelemot, and Alain Barthélemy. Phase locking of fiber laser array using quasi-reinforcement learning, principle and experiments. AI special symposium at Europhoton 2022, Hannover, Invited conference, 28th Aug-2nd Sept.

[39] Maksym Shpakovych, Geoffrey Maulion, Vincent Kermene, Alexandre Boju, Paul Armand, Agnès Desfarges-Berthelemot, and Alain Barthélemy. Optimization and machine learning algorithms applied to the phase control of an array of laser beams-renforcement. SMAI Mode, Limoges, 30 mai au 3 juin 2022.

[40] Maksym Shpakovych, Geoffrey Maulion, Vincent Kermene, Alexandre Boju, Paul Armand, Agnès Desfarges-Berthelemot, and Alain Barthélemy. Verrouillage et structuration dynamique des phases d'un réseau de 7 lasers par approche neuronale. OPTIQUE, Nice, Invited conference, 4-8 juillet 2022.

[41] Maksym Shpakovych, Geoffrey Maulion, Vincent Kermene, Alexandre Boju, Paul Armand, Agnès Desfarges-Berthelemot, and Alain Barthélemy. Combinaison cohérente de 100 faisceaux lasers grâce à un réseau de neurones artificiels appris par quasi-renforcement. OPTIQUE, Dijon, 5-9 juillet 2021.

[42] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv e-prints*, July 2012.

[43] Ju Sun, Qing Qu, and John Wright. A geometric analysis of phase retrieval. pages 2379–2383, 2016.

[44] I. Waldspurger. Phase retrieval with random gaussian sensing vectors by alternating projections. *IEEE Transactions on Information Theory*, 64(5):3301–3312, 2018.

[45] Irène Waldspurger, Alexandre d'Aspremont, and Stéphane Mallat. Phase recovery, maxcut and complex semidefinite programming. *Mathematical Programming*, 149(1):47–81, 2015.

[46] Dan Wang, Qiang Du, Tong Zhou, Derun Li, and Russell Wilcox. Stabilization of the 81-channel coherent beam combination using machine learning. *Opt. Express*, 29(4):5694–5709, Feb 2021.