# Deep Unfolded Extended Conjugate Gradient Method for Massive MIMO Processing with Application to Reciprocity Calibration

Samuel Sirois, Messaoud Ahmed Ouameur and Daniel Massicotte

Université du Québec à Trois-Rivières, Department of Electrical and Computer Engineering,
3351, Boul. des Forges, Trois-Rivières, Québec, Canada
Laboratoire des Signaux et Systèmes Intégrés,
{samuel.sirois, messaoud.ahmed.ouameur, daniel.massicotte}@uqtr.ca

***Abstract*—In this paper, we consider deep unfolding the standard iterative conjugate gradient (CG) algorithm to solve a linear system of equations. Instead of being adjusted with known rules, the parameters are learned via backpropagation to yield the optimal results. However, the proposed unfolded CG (UCG) is extended wherein a scalar parameter is substituted by a matrix-parameter to augment the degrees of freedom per layer. Once the training is completed, the UCG has revealed to require far a smaller number of layers than the number of iterations needed using the standard iterative CG. It is also shown to be very robust to noise and outperforms the standard CG in low signal to noise ratio (SNR) region. A key merit of the proposed approach is the fact that no explicit training data is dedicated to the learning phase as the optimization process relies on the residual error which is not explicitly expressed as a function of the desired data. As an example, the proposed UCG is applied to solve the reciprocity calibration problem encountered in massive MIMO (Multiple-Input Multiple-Output) systems.**

***Index Terms*— Deep unfolding, conjugate gradient, Massive MIMO, reciprocity calibration, least squares.**

## I. Introduction

Conjugate gradient (CG) algorithm is widely used to iteratively solve a large linear system of equations. It is well known to converge rapidly in sparse systems. Otherwise, it is assured to converge in the number of iterations less than or equal the number of unknowns [1-page 214]. It is worth noting that the residual error computed at every iteration doesn't depend on the explicit real solution of the problem. This fact is exploited in the proposed deep unfolded algorithm architecture to train a network without explicit knowledge of the training data.

Recent contributions advocate for the potential of using deep learning (DL) for communication system designs [2]-[6]. As such, some initial insights and findings, using state-of-the-art DL tools, on signal compression [4] and channel decoding [5] are revealed. On the other hand, massively parallel processing architectures, such as graphic processing units (GPUs), have shown to be very energy efficient with remarkable computational capabilities when fully exploited by concurrent algorithms [7]. So far, the goal of introducing DL is to either improve parts of existing algorithms or to completely replace them with an end-to-end approach [8] [9]. As an example, the authors in [2] have discussed several promising new applications of DL to the physical layer. On the other hand, two different deep architectures for point-to-point MIMO detection are introduced in [8] wherein the promising architecture relies on unfolding the iterations of a projected gradient descent algorithm into a network. Deep unfolding to solve a general system of equations consists of using the structure of a known iterative algorithm and consider every iteration as a layer of a neural network. Every parameter in the iterative method that is normally updated with a deterministic rule is instead trained with backpropagation process to yield optimum results in solving the system of equations. Certain parameters can be added or modified in the deep unfolded network to give more degrees of freedom in the training process to allow the algorithm to capture features that would not be considered with the original iterative method. This can augment the complexity of one layer compared to one iteration of the original algorithm but in the end, the number of layers required to solve the problem can be drastically reduced to yield an overall computation complexity and latency smaller than the one obtained with the standard iterative method. Also, another advantage to unfold an algorithm is that once the training is properly done, which can be very tedious sometimes, the network will end up capturing all the inherent details of the problem and can always find a solution with its parameters fixed, whereas the iterative algorithm usually needs to recompute its coefficients every time the system to solve has changed. In other words, once the training is done for a big variety of inputs, the network can generalize for all other inputs. The secret resides in the strategy to train the network to be sure that the parameters are not overtrained or undertrained. In this paper, we will show a training method we used to solve the least-squares (LS) problem inherent to a massive MIMO (Multiple-Input Multiple-Output) reciprocity calibration problem. Another advantage of deep unfolding is that activation functions can be added to the layers to consider nonlinearity of the given problem.

Deep unfolding methods were successfully developed in [10] and [11] to solve a linear system of equations. In [10], the authors deep unfolded the approximate message passing (AMP) iterative algorithm to solve the sparse linear inverse problem. Training data was necessary to learn the parameters of each layer. In [11], the authors unfolded the iterations of a projected gradient descent algorithm and applied their network to the massive MIMO detection problem.

To the best of our knowledge, all deep unfolded algorithm used to solve a linear system of equations need explicit and dedicated knowledge of the training data to be efficient. As pinpointed above, the residual error parameter in the CG algorithm can be exploited to create a deep network that can train without having access to explicit training data. This can be useful in many applications such as in reciprocity calibration and detection problems that are encountered in massive MIMO systems. To gain insights on the potential of this property, we proposed an extended deep unfolded conjugate gradient (UCG) architecture, wherein the parameters are learned via backpropagation, and we applied it to solve the reciprocity calibration problem.

As a matter of fact, massive MIMO is the key technology for the new mobile network generation also known as 5G [12]-[13]. Massive MIMO in the context of the 5G is defined as classical multiuser MIMO with every base station (BS) having a number of antennas much larger than the number of users (UTs) served at the same time [14]-[15]. Massive MIMO technology works in time division duplex (TDD) mode using channel state information (CSI) of the uplink channel for both uplink detection and downlink beamforming [16]-[17]. Unfortunately, the uplink and the downlink channels are not reciprocal. This implies that if the uplink CSI is used for beamforming, the sum-rate capacity performances would degrade [18]-[19]. This non-reciprocity is due to the impairments in the radio frequency (RF) chains in both the transmitters and the receivers of the BS and the UTs. In other words, reciprocity calibration is required to implicitly infer the CSI of the downlink channel [18]. Details of the impacts of non-reciprocal channels and possible solutions are addressed in [18]. Also, a more detailed introduction to the reciprocity calibration problem is addressed in [20] whereas the calibration solution is implemented in a real testbed in [21]. The main idea is that a reference antenna at the BS is used to do the calibration by sending and then receiving signals to and from the other antennas at the BS. Based on the collected data signals, calibration can be done by solving an optimization problem. Unfortunately, the performance of this method varies with the position of the reference antenna and is very noise sensitive. For distributed MIMO systems, access point calibration is proposed in [22] and [23] to generalize the method used in [21]. To eliminate the position-sensitive reference antenna issue, the authors in [19] proposed several LS estimators based on the mutual-coupling between each antenna pair at the BS. In other words, each antenna can be seen as a reference antenna that sends and then receives signals to and from the other antennas at the BS. The authors in [24] proposed a new efficient way to inverse a large matrix based on a successive column-wise update algorithm to solve the LS problem formulation related to the reciprocity calibration. From another point of view, the authors in [25] formulated an expectation-maximization algorithm (EM) that iteratively improved results obtained from other methods such as LS. Both LS and EM methods are well suited for collocated and distributed MIMO BS [23].

The LS problem formulation implies the inversion of a $(M-1)\times(M-1)$ matrix to solve a large linear system of equations, where $M$ is the number of antennas at the BS. Even if the calibration needs to be done approximately on an hourly basis for collocated MIMO systems, fast algorithms are necessary in the case of distributed MIMO systems where the reference clocks are not the same so that the calibration needs to be done frequently. With that in mind, we propose the UCG algorithm to solve the linear system of equations inherent to the reciprocity calibration problem.

There exist several efficient iterative techniques to solve a linear system of equations. These methods are well suited for problems where noise is only present on one side of the system of equations. In the LS problem formulation of reciprocity calibration, noise is present on both sides of the equality sign of the system of linear equations so that classical algorithms such as Gauss-Seidel (GS) and Neumann series expansion (NSE) need a lot of iterations to find a good solution [24].

The reason why we choose to unroll the CG algorithm, instead of other known iterative methods, is that it has a residual error parameter upon which the loss function is computed. A trigger can be set using this parameter to know when to stop learning without having access to explicit training data, which is the case in the reciprocity calibration problem. Also, the CG algorithm is one of the most popular methods to solve a large system of linear equations due to its quick convergence. Furthermore, the reason we decided to use deep unfolding technique in this paper instead of a normal deep neural network (DNN) is because the inner architecture is specially designed to solve linear system of equations whereas DNN is a more general architecture that suits for all sorts of problems so that it would probably need more layers to obtain similar results. This results in a higher algorithm complexity that needs to be avoided in most problems. Some may argue that the nonlinear activation functions of a standard DNN can efficiently solve the reciprocity calibration problem, but nothing prevents us to add a nonlinear function on the output of every layer of the UCG method. Nevertheless, we have chosen to keep the activation function linear.

Therefore, our approach starts by unfolding the iterative CG method to infer the data tree dependence graph over which the gradients are computed. As such, the contributions are: (i) We propose a data dependence tree graph based on unfolding the iterative CG algorithm where the parameters are now trained rather than being explicitly computed. (ii) We derive closed-form expressions of the gradients of the loss function with respect to the parameters. The loss function is set to be the squared norm of the residual error term, which as mentioned above is not an explicit function of the training data. (iii) We discuss the proposed algorithm performance in comparison to the standard iterative CG in the reciprocity calibration use case.

The paper is organized as follows: Section II presents the deep UCG method. Section III summarizes the reciprocity calibration problem. Section IV discusses some simulation results obtained with the UCG. Finally, a conclusion is drawn.

## II. Deep unfolded extended conjugate gradient

Consider solving a real valued linear system of equations of the form $\mathbf{Ax}=\mathbf{y}$ with $\mathbf{A}$ being a squared positive-definite matrix of dimensions $(2M-2)\times(2M-2)$, where $M$ is the number of antennas at the BS in the reciprocity calibration problem context. The standard iterative CG algorithm initialization steps [1-page 214] are depicted as follow

$$\hat{\mathbf{x}}_1 = \textit{Initial guess} \tag{1}$$

$$\mathbf{r}_1 = \mathbf{y} - \mathbf{A}\hat{\mathbf{x}}_1 \tag{2}$$

$$\mathbf{s}_1 = \mathbf{A}\mathbf{r}_1 \tag{3}$$

$$\mathbf{p}_1 = \mathbf{s}_1 \tag{4}$$

$$\gamma_1 = \left\|\mathbf{s}_1\right\|^2 \tag{5}$$

where $\hat{\mathbf{x}}_1$ is the first approximate solution of $\mathbf{x}$. Once these initializations are done, the main CG algorithm for the $k^{th}$ iteration goes as follow

$$\mathbf{q}_k = \mathbf{A}\mathbf{p}_k \tag{6}$$

$$\alpha_k = \frac{\gamma_k}{\left\|\mathbf{q}_k\right\|^2} \tag{7}$$

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_k + \alpha_k \mathbf{p}_k \tag{8}$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{q}_k \tag{9}$$

$$\mathbf{s}_{k+1} = \mathbf{A}\mathbf{r}_{k+1} \tag{10}$$

$$\gamma_{k+1} = \left\|\mathbf{s}_{k+1}\right\|^2 \tag{11}$$

$$\mathbf{p}_{k+1} = \mathbf{s}_{k+1} + \frac{\gamma_{k+1}}{\gamma_k}\mathbf{p}_k \tag{12}$$

Hence, we propose deep unfolding the conjugate gradient iterative algorithm into few layers and learn the parameters $\alpha_k$ and $\boldsymbol{\lambda}_k \triangleq \gamma_{k+1}/\gamma_k$ via backpropagation. In fact, the original problem is a complex valued linear system of equations $\overline{\mathbf{A}}\overline{\mathbf{x}} = \overline{\mathbf{y}}$, with $\overline{\mathbf{A}}$ being a square matrix of dimensions $(M-1)\times(M-1)$. We will use the following transformation to convert the problem into a real valued one.

$$\breve{\mathbf{y}} = \begin{pmatrix} \mathrm{Re}(\overline{\mathbf{y}}) \\ \mathrm{Im}(\overline{\mathbf{y}}) \end{pmatrix} \in \mathbb{R}^{(2M-2)\times 1} \tag{13}$$

$$\breve{\mathbf{A}} = \begin{pmatrix} \mathrm{Re}(\overline{\mathbf{A}}) & -\mathrm{Im}(\overline{\mathbf{A}}) \\ \mathrm{Im}(\overline{\mathbf{A}}) & \mathrm{Re}(\overline{\mathbf{A}}) \end{pmatrix} \in \mathbb{R}^{(2M-2)\times(2M-2)} \tag{14}$$

$$\mathbf{x} = \begin{pmatrix} \mathrm{Re}(\overline{\mathbf{x}}) \\ \mathrm{Im}(\overline{\mathbf{x}}) \end{pmatrix} \in \mathbb{R}^{(2M-2)\times 1} \tag{15}$$

where $\mathrm{Re}(\cdot)$ and $\mathrm{Im}(\cdot)$ denote the real and imaginary part of the terms in parenthesis respectively. Since the CG algorithm can only solve for positive definite symmetric matrix, we define

$$\mathbf{A} = \breve{\mathbf{A}}^T\breve{\mathbf{A}} \in \mathbb{R}^{(2M-2)\times(2M-2)} \tag{16}$$

$$\mathbf{y} = \breve{\mathbf{A}}^T\breve{\mathbf{y}} \in \mathbb{R}^{(2M-2)\times 1} \tag{17}$$

with these conventions established, we can write

$$\mathbf{A}\mathbf{x} = \mathbf{y} \tag{18}$$

and solve for $\mathbf{x}$ and then after retrieve the complex form $\overline{\mathbf{x}}$ using (15).

The conjugate gradient method iteratively computes an estimate of $\mathbf{x}$ denoted as $\hat{\mathbf{x}}$. To unfold the algorithm, equations (6) and (9) are combined, equations (8) and (10) are used as-is and the scalar term $\frac{\gamma_{k+1}}{\gamma_k}$ in equation (12) is replaced by the matrix $\boldsymbol{\lambda}$ to yield the following formulas describing the $k^{th}$ layer (unfolded iteration) of the UCG algorithm

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k \tag{19}$$

$$\mathbf{s}_k = \mathbf{A}\mathbf{r}_{k+1} \tag{20}$$

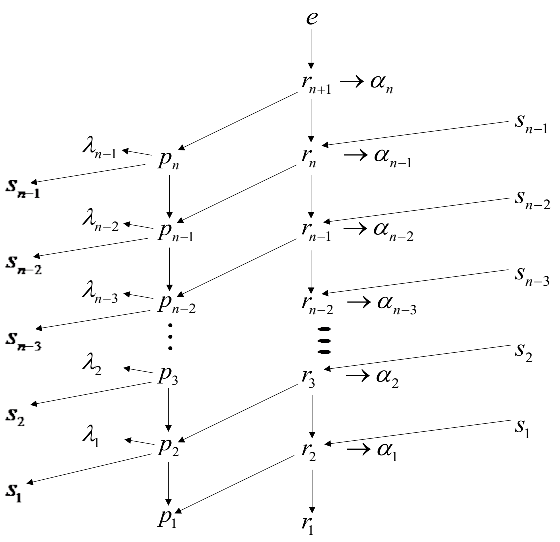

Figure 1. Dependence tree of the unfolded algorithm

$$\mathbf{p}_{k+1} = \mathbf{s}_k + \boldsymbol{\lambda}_k \mathbf{p}_k \tag{21}$$

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_k + \alpha_k \mathbf{p}_k \tag{22}$$

Where $\mathbf{r}$ is the residual error on the estimation of the unknown parameter $\mathbf{X}$ with dimensions $(2M-2)\times 1$, $\mathbf{p}$ and $\mathbf{s}$ are intermediate variables with dimensions $(2M-2)\times 1$, $\alpha_k$ is a scalar parameter and $\boldsymbol{\lambda}$ is a $(2M-2)\times(2M-2)$ matrix parameter to learn. Normally, the parameter $\boldsymbol{\lambda}$ is represented by a scalar in the classical conjugate gradient method. Substituting such a scalar parameter with a matrix is intentionally introduced to provide more degree of freedom to every layer. Hence the name *extended* unfolded conjugate gradient which is we refer to as UCG for simplicity. The use of $\boldsymbol{\lambda}$ as a matrix yields better results against noise effects. The initial conditions for the UCG are defined by

$$\mathbf{r}_1 = \mathbf{y} - \mathbf{A}\hat{\mathbf{x}}_1 \tag{23}$$

$$\mathbf{p}_1 = \mathbf{A}\mathbf{r}_1 \tag{24}$$

with the initial guess $\hat{\mathbf{x}}_1$ that can be set to the zero vector or to the best initial guess. Figure 1 shows the dependence tree of the deep unfolded algorithm. The loss function $e$ is based on the $L_2$ norm applied on $\mathbf{r}_{n+1}$ as

$$e = \mathrm{Loss}\left(\Theta;\mathbf{A},\mathbf{x}\right) = \left\|\mathbf{r}_{n+1}\right\|_2^2 \tag{25}$$

$$\Theta = \left\{\alpha_k, \boldsymbol{\lambda}_k\right\}_{k=1}^{n \text{ for } \alpha \text{ and } (n-1) \text{ for } \boldsymbol{\lambda}} \tag{26}$$

The loss function $e$ is minimized over $\Theta = \left\{\alpha_k, \boldsymbol{\lambda}_k\right\}_{k=1}^{n \text{ for } \alpha \text{ and } (n-1) \text{ for } \boldsymbol{\lambda}}$. When the loss function based on the residual error tends to the zero vector, the error on the unknown parameter $\mathbf{x}$ will also tend towards zero. This fact is the key concept of the algorithm. Indeed, as we will see in the next section, it enables the algorithm to decide whether it needs to *retrain* or not based on a threshold derived from the loss function. In addition, since the loss function is not an explicit function of the desired data, it makes the proposed approach blind.

The derivation of the gradient of the loss function with

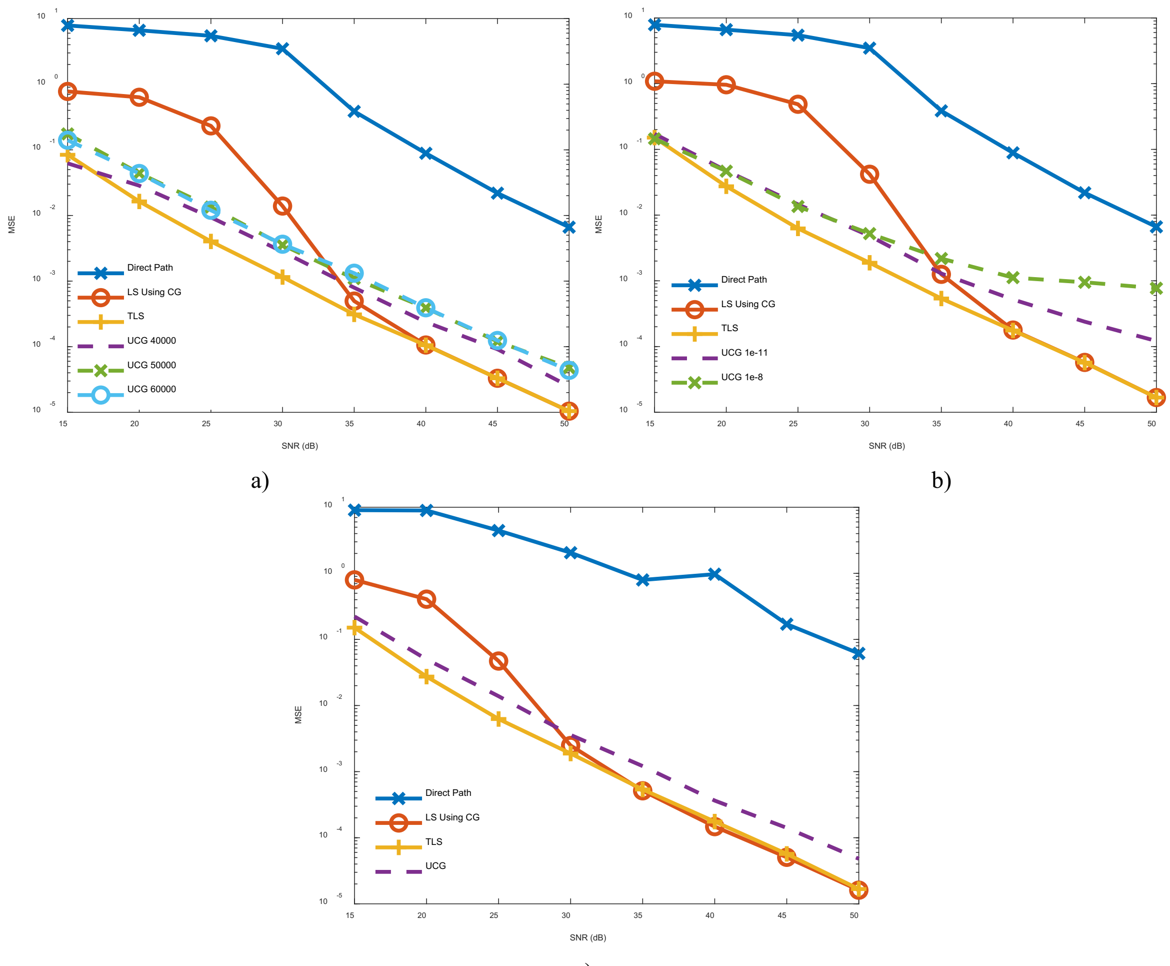

Figure 2. Reciprocity calibration results for UCG algorithm. a) Results obtained with a training SNR of 60 dB and a trigger value set to $10^{-13}$ for three different sizes of training, namely 40000, 50000 and 60000 iterations per realization of system of equations. b) Results obtained with a training SNR of 45 dB for two different triggers values, namely $10^{-8}$ and $10^{-11}$ with a size of training of 40000 iterations per realization of a system of equations. c) Results obtained with 50 antennas with a training SNR of 60 dB and a $10^{-13}$ trigger value.

respect to $\alpha$ and $\boldsymbol{\lambda}$ is presented in Appendix A as well as the training method used during the backpropagation process.

## III. Use case: Reciprocity calibration in massive MIMO

### *A. Least square problem formulation for reciprocity calibration*

This sub-section summarizes the LS problem formulation for reciprocity calibration. In fact, solving the LS linear system of equations permits us to find estimations of calibration coefficients. Once these coefficients are computed, they can be multiplied by the uplink channel coefficients to get the implicit downlink channel estimation. For more details on the LS formulation, readers can refer to [19], [21] and [24]. The calibration coefficients $b^{(m)}$, where $m = 1,\cdots,M$, are estimated with every antenna at the BS sending and receiving known pilots signals $s_p = +1$ from and to the other antennas. We denote $y^{(m,l)}$ the signal received by the antenna $m$ and transmitted by the antenna $l$. Grouped in pair, the signals exchanged between two antennas are written as [9]

$$\begin{pmatrix} y^{(l,m)} \\ y^{(m,l)} \end{pmatrix} = \alpha^{(l,m)} \begin{pmatrix} b^{(l)} \\ b^{(m)} \end{pmatrix} + \begin{pmatrix} n^{(l,m)} \\ n^{(m,l)} \end{pmatrix} \tag{27}$$

where $\alpha^{(l,m)} = t_B^{(l)} t_B^{(m)} \tilde{h}^{(l,m)} = t_B^{(m)} t_B^{(l)} \tilde{h}^{(m,l)}$, $t_B^{(m)}$ and $t_B^{(l)}$ are the complex-valued coefficients (gain and phase) that model the RF transmitter of the BS, and $\tilde{h}^{(l,m)} = \tilde{h}^{(m,l)}$ is the reciprocal propagation channel. $\begin{pmatrix} n^{(l,m)} & n^{(m,l)} \end{pmatrix}^T$ is an independent zero-mean circularly symmetric complex Gaussian distributed random noise vector whose components have variance $N_0$.

To model the reciprocal channel $\tilde{h}^{(l,m)}$, the authors in [19] empirically determine an equation based on measurements done in an anechoic chamber. The channel CSI between antenna $l$ and antenna $m$ can be approximated as

$$\tilde{h}^{(l,m)} = \beta^{(l,m)} \exp\left( j\phi^{(l,m)} \right) + w^{(l,m)} \tag{28}$$

where $\beta^{(l,m)} = 0.03\left(d^{(l,m)}\right)^{-3.7}$, the distance $d^{(l,m)}$ between the antennas is in multiple of half the wavelength, the phase $\phi^{(l,m)}$ is uniformly distributed in the range $[0, 2\pi]$ and the component $w^{(l,m)}$ is a circularly symmetric complex Gaussian distributed random variable whose components have a variance $N_w$ to model the weak effects of the scattering due to far elements in the environment.

The direct path method that only use one reference antenna to estimate the calibration coefficients $\tilde{\mathbf{b}} = \left(\tilde{b}^{(0)},\ \tilde{b}^{(1)},\ldots,\ \tilde{b}^{(M-1)}\right)^T$ with the pairs of signals $y^{(ref,m)}$ and $y^{(m,ref)}$ can be written as

$$\tilde{b}^{(m)} = y^{(ref,m)} \big/ y^{(m,ref)} \tag{29}$$

where *ref* designates the reference antenna.

The direct path method can be generalized with every antenna at the BS being used as a reference antenna to augment diversity and to get a better LS-based estimator. In other words, all pairs of signals in equation (27) are used in the estimator to yield the following cost function to minimize

$$J\left(\tilde{\mathbf{b}}\right) = \sum_{m,l\neq m} \left\| \tilde{b}^{(m)} y^{(m,l)} - \tilde{b}^{(l)} y^{(l,m)} \right\|^2 \tag{30}$$

which leads to the closed-form solution

$$\tilde{\mathbf{b}} = \left(\tilde{b}^{(1)},\ \tilde{b}^{(2)},\ldots,\ \tilde{b}^{(\mathrm{M}-1)}\right)^T = -\left(\widetilde{\mathbf{A}}_1^H \widetilde{\mathbf{A}}_1\right)^{-1} \widetilde{\mathbf{A}}_1^H \tilde{\mathbf{a}} \tag{31}$$

where $\tilde{\mathbf{a}}$ is the first column of $\widetilde{\mathbf{A}} \triangleq \left[\tilde{\mathbf{a}} \quad \widetilde{\mathbf{A}}_1\right]$ and $\tilde{b}^{(1)}$ is set to 1 to avoid the all zeros solution $\tilde{\mathbf{b}} = \mathbf{0}$. This constraint does not impact the quality of the calibration coefficients because the precision of the estimation can be done up to a multiple of a constant without degrading the performances of downlink beamforming. The matrix $\widetilde{\mathbf{A}}$ is defined as

$$\widetilde{\mathbf{A}}^{(m,l)} = \begin{cases} \sum_{l=0}^{M-1} \left| y^{(m,l)} \right|^2, & m = l \\ -y^{*(m,l)} y^{(l,m)}, & m \neq l \end{cases} \tag{32}$$

Other types of LS estimators such as weighted LS are also presented in [9].

### B. Total least squares estimator

Rewriting equation (31) leads to $\widetilde{\mathbf{A}}_1 \tilde{\mathbf{b}} = -\tilde{\mathbf{a}}$ which has the same form as $\overline{\mathbf{A}}\overline{\mathbf{x}} = \overline{\mathbf{y}}$ (refer to Section II). When Gaussian noise is only present in $\tilde{\mathbf{a}}$, the LS estimator is an optimal solution to solve for $\tilde{\mathbf{b}}$ [26]. Unfortunately, in the context of reciprocity calibration, noise is present in both $\widetilde{\mathbf{A}}_1$ and $\tilde{\mathbf{a}}$. A more reliable technique than directly inverting the matrix $\widetilde{\mathbf{A}}_1$ is to solve the system of equations with the total least squares (TLS) method which is more robust when Gaussian noise is on both sides of the system of equations. The TLS estimator finds a value of $\tilde{\mathbf{b}}$ which minimize

$$\left\| \Delta \widetilde{\mathbf{A}}_1 \right\|^2 + \left\| \Delta \tilde{\mathbf{a}} \right\|^2 \tag{33}$$

subject to the equality constraint

$$\left( \widetilde{\mathbf{A}}_1 - \Delta \widetilde{\mathbf{A}}_1 \right) \tilde{\mathbf{b}} = \left( -\tilde{\mathbf{a}} - \Delta \tilde{\mathbf{a}} \right) \tag{34}$$

where $\Delta \widetilde{\mathbf{A}}_1$ and $\Delta \tilde{\mathbf{a}}$ define the noise on $\widetilde{\mathbf{A}}_1$ and $\tilde{\mathbf{a}}$ respectively [27]. To obtain the TLS solution, a singular value decomposition (SVD) of the extended matrix $\left( \widetilde{\mathbf{A}}_1 \quad \tilde{\mathbf{a}} \right)$ needs to be done in order to retrieve the right singular vector $\mathbf{v} = \left( v^{(1,M)},\ v^{(2,M)},\ldots,\ v^{(M,M)} \right)^T$ corresponding to the smallest singular value of the decomposition. One can normalize the estimation as

$$\tilde{\mathbf{b}}_{TLS} = \frac{1}{v^{(1,M)}} \mathbf{v} \tag{35}$$

to set $\hat{b}_{TLS}^{(1)} = 1$. As it will be shown in the results, TLS and LS estimators perform equally well at high SNR because the noise is less important in the system of equations. Unfortunately, this method has a high computational cost for real-time implementation.

## IV. Performance Analysis

### A. Simulation set up and parameters

The simulation set up is inspired from [19] and [24] with a $10 \times 10$ planar patch array and the variance $N_w$ of the channel Rayleigh component is set to -50 dB. The center antenna (antenna element 49) is set as the reference element for the direct path method. The RF chain is modeled with the parameters in [23] where the coefficients of the transmitters and the receivers have a uniformly distributed phase in the range of $[-\pi, \pi]$ and their magnitude is uniformly distributed within $[1-\delta, 1+\delta]$, where $\delta$ is selected to respect the following condition

$$\sqrt{E\left( \left( \left| t_m^B \right| - 1 \right)^2 \right)} = \sqrt{E\left( \left( \left| r_m^B \right| - 1 \right)^2 \right)} = 0.1 \tag{35}$$

where $E(\cdot)$ is the expectation operator. The UCG algorithm has 7 layers ($n$=7) where $\alpha$ and $\boldsymbol{\lambda}$ are initialized with non-zero small values so that the algorithm doesn't diverge. The ratio $\mu_1 / \mu_2$ is approximately $10^5$ and the momentum coefficient $\beta$ is set as close as to 1 to prevent the algorithm from diverging. The reader can refer to appendix A for more details.

The calibration SNR, in the simulation results (c.f. figure 2), is normalized with respect to the SNR of two adjacent antennas. Since solving equation (18) implies a $(2M-2)\times(2M-2)$ matrix inversion, iterative methods such as GS and NSE can be used to solve for $\tilde{\mathbf{b}}$ [24]. Indeed, these methods have been successfully used in the zero-forcing (ZF) and the minimum mean square error (MMSE) massive MIMO detection problem where the direct matrix inversion is computationally expensive. Unfortunately, as it is shown in [24], such methods will require a large number of iterations to converge in the context of solving the LS problem encountered in reciprocity calibration.

### B. Simulation results and discussion

Figure 2 depicts the MSE as a function of the normalized SNR. The TLS estimator has good results at low SNR compared to the LS because of the presence of the noise in both $\widetilde{\mathbf{A}}_1$ and $\tilde{\mathbf{a}}$. For the simulation purposes, the LS problem is solved with the original conjugate gradient method which would have yielded the same results as if it would have been done with the direct matrix inversion. Also, as expected, when the SNR increases, both LS and TLS estimators tend to reach the same results.

TABLE 2. COMPLEXITY IN TERMS OF REAL ADDITIONS AND MULTIPLICATIONS OF THE INITIALIZATION OF THE CG ALGORITHM

| EQ. | ADD. | MULT. |
|---|---|---|
| (2.2) | $2(M-1)^2+2(M-1)(M-2)+2(M-1)$ | $4(M-1)^2$ |
| (2.3) | $2(M-1)^2+2(M-1)(M-2)$ | $4(M-1)^2$ |
| (2.5) | $(M-1)+(M-2)$ | $2(M-1)$ |
| TOT | $8M^2-16M+8$ | $8M^2-14M+6$ |

TABLE 3. COMPLEXITY IN TERMS OF REAL ADDITIONS, MULTIPLICATIONS, AND DIVISIONS OF ONE ITERATION OF THE CG ALGORITHM

| EQ. | ADD. | MULT. | DIV. |
|---|---|---|---|
| (3.1) | $2(M-1)^2+2(M-1)(M-2)$ | $4(M-1)^2$ | 0 |
| (3.2) | $(M-1)+(M-2)$ | $2(M-1)$ | 1 |
| (3.3) | $2(M-1)$ | $2(M-1)$ | 0 |
| (3.4) | $2(M-1)$ | $2(M-1)$ | 0 |
| (3.5) | $2(M-1)^2+2(M-1)(M-2)$ | $4(M-1)^2$ | 0 |
| (3.6) | $(M-1)+(M-2)$ | $2(M-1)$ | 0 |
| (3.7) | $2(M-1)$ | $2(M-1)$ | 1 |
| TOT | $8M^2-10M$ | $8M^2-6M-2$ | 2 |

TABLE 4. COMPLEXITY IN TERMS OF REAL ADDITIONS AND MULTIPLICATIONS OF THE INITIALIZATION OF THE UCG ALGORITHM

| EQ. | ADD. | MULT. |
|---|---|---|
| (17) | $(2M-2)(2M-3)+(2M-2)$ | $(2M-2)^2$ |
| (18) | $(2M-2)(2M-3)$ | $(2M-2)^2$ |
| TOT. | $8M^2-18M+10$ | $8M^2-16M+8$ |

TABLE 5. COMPLEXITY IN TERMS OF REAL ADDITIONS AND MULTIPLICATIONS OF ONE LAYER OF THE UCG ALGORITHM

| EQ. | ADD | MULT |
|---|---|---|
| (13) | $(2M-2)(2M-3)+(2M-2)$ | $(2M-2)^2+(2M-2)$ |
| (14) | $(2M-2)(2M-3)$ | $(2M-2)^2$ |
| (15) | $(2M-2)(2M-3)+(2M-2)$ | $(2M-2)^2$ |
| (16) | $(2M-2)$ | $(2M-2)$ |
| TOT | $12M^2-24M+12$ | $12M^2-20M+8$ |

TABLE 6. COMPLEXITY IN TERMS OF REAL ADDITIONS AND MULTIPLICATIONS TO MAKE THE SYSTEM OF EQUATION POSITIVE DEFINITE SYMMETRIC FOR THE CG AND THE UCG

| ALG. | ADD. | MULT. |
|---|---|---|
| CG | $2M^3+M^2-5M+2$ | $2M^3+2M^2-4M$ |
| UCG | $4M^3-4M^2-M+1$ | $4M^3-2M^2-2M$ |

TABLE 7. TOTAL COMPLEXITY IN TERMS OF REAL ADDITIONS AND MULTIPLICATIONS FOR THE CG AND THE UCG

| ALG. | ADD. | MULT. |
|---|---|---|
| CG | $2M^3+2021M^2-2543M+19$ | $2M^3+2006M^2-1516M-492$ |
| UCG | $4M^3+80M^2-169M+85$ | $4M^3+82M^2-142M+56$ |

Of particular interest is the proposed UCG method which only requires 7 layers and has better results than LS at low SNR. The original CG algorithm requires approximately 250 iterations to converge and it gets the same results as the LS method. These improvements are mainly due to the fact that $\boldsymbol{\lambda}$ is a matrix that provides extra degrees of freedom in the learning process. Based on our early simulation results, unfolding the CG algorithm without extending $\boldsymbol{\lambda}$ as a matrix parameter shows similar performances as the normal iterative algorithm, so that $\boldsymbol{\lambda}$ has to be a matrix to improve the performance.

The complexity per layer/iteration in UCG becomes more important because it requires the multiplication of a matrix with a vector instead of a scalar with a vector. Tables 2, 3, 4 and 5 show the complexity of both CG and UCG in terms of real additions (ADD) and multiplications (MULT) with respect to the number of antennas at the BS. It is important to mention that the complexity of the CG has been calculated with the assumption that equations (13), (14) and (15) are not used. In other words, the CG algorithm solves directly $\tilde{\mathbf{A}}_1^H \tilde{\mathbf{A}}_1 \tilde{\mathbf{b}} = \tilde{\mathbf{A}}_1^H(-\tilde{\mathbf{a}})$. This explains why the complexity to make the system of equations positive definite symmetric depicted in Table 6 is not the same for both algorithms. The reason we introduced the convention of equations (13), (14) and (15) for UCG was for sake of simplicity during the development process of the algorithm. The UCG could have been derived using the Wirtinger derivatives to yield even better complexity results. Therefore, for large $M$, it is easy to see that one layer of the UCG requires approximately $4M^2$ more additions and multiplications than one iteration of the CG. However, matrix-vector multiplication can easily be parallelized on GPU or on field-programmable gate array (FPGA) to compensate this disadvantage. Also, since UCG requires far fewer layers than the CG, the overall complexity of the UCG becomes much smaller than the complexity of the CG. In fact, the total complexity of both CG and UCG are shown in Table 7. As expected, the UCG is far computationally less expensive than the CG.

To train the algorithm for one particular value of the calibration coefficients vector, 5 to 15 realizations, depending on the value of the threshold of the system of equation $\tilde{\mathbf{A}}_1 \tilde{\mathbf{b}} = -\tilde{\mathbf{a}}$ with different noise values, are necessary. This represents a big advantage for heterogeneous systems because a low interconnect bandwidth is used to transmit data. For example, the training phase of the UCG could be done on a GPU and the application of the algorithm could be implemented in an FPGA. To continue, on Figure 2.a, the backpropagation process is performed with 40000, 50000 and 60000 iterations per realization of a system of equations respectively. The optimal number of iterations is obviously 40000. On the other hand, simulations with a number of iterations below 35000 do not converge. Figure 2.b shows the performances of the UCG with training at an SNR of 45 dB with the optimal number of iterations and two different trigger values of $10^{-8}$ and $10^{-11}$ respectively. One can easily see the effect of the trigger on the performances of the UCG method. That said, when the trigger is set correctly, the MSE is excellent only for the values of SNR below the value of the training SNR. This is not a problem since with the performances of the generalized LS method, the sum-rate loss is negligible at a calibration SNR above 20 dB for maximum ratio transmission (MRT) precoding and above 35 dB for ZF precoding [18], [19]. Hence, as long as the training is done in a region of the SNR where the performances of the UCG are better than those of the generalized LS at 20 dB or 35 dB, no problem should occur. For sake of comparison, Figure 2.c shows the UCG performances with 50 antennas instead of 100. The results for UCG seem to be independent of the number of antennas whereas the LS method has a better MSE values in high SNR region as the number of antennas decreases. It is also worth mentioning that the training SNR can vary to ±5 dB from a system of equations to another without perturbing the performances. Once the training is done for one particular value of the calibration coefficients vector, one or more layers of the UCG can be updated every time the calibration needs to be redone. Eventually, the unfolded algorithm would consider the effects of the slight variation with time and external factors (temperature, humidity, clock drift, etc.) of the parameters of the system of equations $\tilde{\mathbf{A}}_1\tilde{\mathbf{b}} = -\tilde{\mathbf{a}}$ until it doesn't need to retrain very often. This implies a sacrifice in the complexity of the UCG in the learning phase but once it is done, the complexity becomes as depicted in Table 7. Figure 2 shows the results with a fixed value of the calibration coefficients vector. Evaluation of the performances of the UCG algorithm with coefficients of calibration varying with time and external factors is out of the scope of this paper and is reserved for future work.

Also, as mentioned in the previous section, the UCG algorithm is blind and knows when to stop learning with the value of the threshold set on the loss function. In other words, the algorithm keeps doing the backpropagation process as long as the value of the loss function is above the threshold value. This is a big advantage compared to other unfolded algorithms such as the learned iterative hard-thresholding (LISTA) method [10] and the Detection Network (DetNet) [11] which need explicit training data in order to learn their parameters. The goal is then to try to find the smallest trigger depending on the training SNR that the algorithm can reach during the learning process. This enables the UGC to first evaluate the solution of a system of equations and then decides if it needs to *retrain* or not even if it doesn't have access to the solution.

Finally, it is worth mentioning that the UCG can be seen as a linear neural network. That being said, nonlinear activation functions could be added to enhance the algorithm in a real-world implementation where nonlinearities often occur. This is also part of future work.

## V. Conclusion

Deep unfolding for solving a large linear system of equations consists of taking separately each iteration of an iterative algorithm and consider them as a layer of a deep neural network. Instead of updating the parameters within each layer with a known formula, backpropagation is used to optimally adjust them to fit the specific problem. Therefore, the conjugate gradient algorithm is successfully unfolded to solve a large linear system of equations. Also, a scalar parameter within the algorithm is transformed into a matrix parameter to augment the versatility of the deep unfolded network. We have chosen to unfold the conjugate gradient method because there is an intrinsic parameter (residual error) calculated at every iteration to deduce if the algorithm is close or not to an acceptable solution. In other words, without learning data, we can set a threshold on this parameter to know when to stop training the deep unfolded network. This makes indeed the proposed network blind. In many real-life problems, access to training data is very difficult to obtain so that our proposed UCG method can overcome this issue. For example, in the massive MIMO context, the reciprocity calibration problem requires to solve a linear system of equations based on a LS estimator without having access to learning pilots. Toward that end, we have shown that our network can solve the problem with much fewer layers than the number of iterations required with the original conjugate gradient algorithm. Also, at low SNR, our method has performances close to the TLS algorithm which is considered to be optimal.

## Acknowledgments

This work has been funded by the Natural Sciences and Engineering Research Council of Canada, Prompt and NUTAQ Innovation, and the CMC Microsystems for equipments. This work was supported by the Chaire de recherche sur les signaux et l'intelligence de systems haute performance for technical support.

## References

[1] J. E. Gentle, Matrix Algebra: Theory, computations and applications in statistics, Springer, 2007, ISBN: 978-0-387-70872-0.

[2] T. O'Shea and J. Hoydis, "An Introduction to Deep Learning for the Physical Layer," in *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563-575, Dec. 2017.

[3] N. Samuel, T. Diskin and A. Wiesel, "Deep MIMO detection," *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, Sapporo, 2017, pp. 1-5.

[4] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Unsupervised representation learning of structured radio communication signals," in Proc. IEEE Int. Workshop Sensing, Processing and Learning for Intelligent Machines (SPLINE), 2016, pp. 1–5.

[5] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning based channel decoding," in Proc. IEEE 51st Annu. Conf. Inf. Sciences Syst. (CISS), 2017, pp. 1–6.

[6] T. Schenk, RF imperfections in high-rate wireless systems: Impact and digital compensation. Springer Science & Business Media, 2008.

[7] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy efficient reconfigurable accelerator for deep convolutional neural networks," IEEE J. Solid-State Circuits, vol. 52, no. 1, pp. 127–138, 2017.

[8] V. Raj and S. Kalyani, "Backpropagating through the air: Deep learning at physical layer without channel models," in IEEE Communications Letters, vol. 22, no. 11, pp. 2278-2281, Nov. 2018.

[9] H. Ye, G. Y. Li and B. Juang, "Power of deep learning for channel estimation and signal detection in OFDM systems," in IEEE Wireless Communications Letters, vol. 7, no. 1, pp. 114-117, Feb. 2018.

[10] M. Borgerding, P. Schniter. "Onsager-corrected deep learning for sparse linear inverse problems," 2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP). IEEE, 2016.

[11] S. Neev, T. Diskin, A. Wiesel. "Learning to detect," IEEE Transactions on Signal Processing, 2019, vol. 67, no 10, p. 2554-2564.

[12] F. Boccardi, R. W. Heath, A. Lozano, T. L. Marzetta and P. Popovski, "Five disruptive technology directions for 5G," IEEE Commun. magazine, vol. 52, no. 2, Feb. 2014.

[13] T. L. Marzetta, "Noncooperative Cellular Wireless with Unlimited Numbers of Base Station Antennas," IEEE Trans. Wireless Commun., vol. 9, no. 11, Nov. 2010.

[14] J. Hoydis, K. Hosseini, S. T. Brink, and M. Debbah, "Making Smart Use of Excess Antennas: Massive MIMO, Small Cells, and TDD," Bell Labs Technical Journal, vol. 18, no. 2, Sep. 2013.

[15] H. Q. Ngo, Massive MIMO: Fundamentals and System Designs, Ph.D. Thesis, Linköping University Electronic Press, 2015.

[16] N. Shariati, E. Björnson, M. Bengtsson and M. Debbah, "Low-Complexity Polynomial Channel Estimation in Large-Scale MIMO With

Arbitrary Statistics," in *IEEE Journal of Selected Topics in Signal Processing*, vol. 8, no. 5, pp. 815-830, Oct. 2014.

[17] H. Yin, D. Gesbert, M. Filippou, and Y. Liu, "A coordinated approach to channel estimation in large-scale multiple-antenna systems," IEEE J. Sel. Areas Commun., vol. 31, no. 2, pp. 264–273, 2013.

[18] A. Bourdoux and L. Van der Perre, "Analysis of non-reciprocity impact and possible solutions," Technical report, MAMMOET project, ref. no. ICT-619086/D2.4, September 2015.

[19] J. Vieira, F. Rusek, and F. Tufvesson, "Reciprocity calibration methods for massive MIMO based on antenna coupling," 2014 IEEE Global Communications Conference, Austin, TX, 2014, pp. 3708-3712.

[20] F. Kaltenberger, H. Jiang, M. Guillaud, and R. Knopp, "Relative channel reciprocity calibration in MIMO/TDD systems," in Future Network and Mobile Summit, 2010, June 2010, pp. 1–10

[21] C. Shepard, H. Yu, N. Anand, E. Li, T. Marzetta, R. Yang, and L. Zhong, "Argos: Practical many-antenna base stations," in Proceedings of the 18th Annual International Conference on Mobile Computing and Networking, ser. Mobicom '12. New York, NY, USA: ACM, 2012, pp. 53–64.

[22] E. Björnson, E. G. Larsson and T. L. Marzetta, "Massive MIMO: ten myths and one critical question," in *IEEE Communications Magazine*, vol. 54, no. 2, pp. 114-123, February 2016.

[23] R. Rogalin *et al*., "Scalable Synchronization and Reciprocity Calibration for Distributed Multiuser MIMO," in *IEEE Transactions on Wireless Communications*, vol. 13, no. 4, pp. 1815-1831, April 2014.

[24] M. Ahmed Ouameur and D. Massicotte, "Successive Column-wise Matrix Inversion Update for Large Scale Massive MIMO Reciprocity Calibration," accepted in IEEE Wireless Communications and Networking Conference (WCNC), Marrakech, Morocco, April 2019.

[25] J. Vieira, F. Rusek, O. Edfors, S. Malkowsky, L. Liu and F. Tufvesson, "Reciprocity Calibration for Massive MIMO: Proposal, Modeling, and Validation," in IEEE Transactions on Wireless Communications, vol. 16, no. 5, pp. 3042-3056, May 2017.

[26] K. S. Arun, 'A unitary constrained total least squares problem in signal processing,' SIAM J. Matrix Anal. Appl. Vol. 13, no. 3, pp. 728-745.

[27] A. Cichocki and R. Unbehauen, "Simplified neural networks for solving linear least squares and total least squares problems in real time," in IEEE transactions on neural networks, 1994, vol. 5, no 6, pp. 910-923.

## APPENDIX A

The reader must refer to the dependence tree shown in Figure 1 and to equations (19), (20) and (21) to understand the following derivations. The explicit forms of the gradient of the loss function $e$ with respect to $\alpha_k$ and $\boldsymbol{\lambda}_k$ are

$$\overline{\frac{\partial e}{\partial \alpha_k}} = 2\mathbf{r}_{n+1}^T \overline{\frac{\partial \mathbf{r}_{n+1}}{\partial \alpha_k}} \tag{A.1}$$

$$\overline{\frac{\partial e}{\partial \boldsymbol{\lambda}_k}} = 2\sum_{z=1}^{2M} \mathbf{r}_{n+1}^{(z)} \overline{\frac{\partial \mathbf{r}_{n+1}^{(z)}}{\partial \boldsymbol{\lambda}_k}} \tag{A.2}$$

where $\overline{\frac{\partial(\cdot)}{\partial(\cdot)}}$ indicates the total derivative and $\overline{\frac{\partial \mathbf{r}_{n+1}^{(z)}}{\partial \boldsymbol{\lambda}_k}}$ represents the total derivative of the $z^{th}$ element of $\mathbf{r}_{n+1}$ with respect to $\boldsymbol{\lambda}_k$ where $k\in[1:n]$ for equation (A.1) and $k\in[1:n-1]$ for equation (A.2). The formulas for the total derivative of $\mathbf{r}_{n+1}$ with respect to $\alpha_k$ and $\boldsymbol{\lambda}_k$ use

$$\overline{\frac{\partial \mathbf{r}_{n+1}}{\partial \alpha_n}} = \frac{\partial \mathbf{r}_{n+1}}{\partial \alpha_n} \tag{A.3}$$

where the total derivative of $\mathbf{r}_{n+1}$ with respect to $\alpha_n$ corresponds to the only possible path on figure 1 that goes from $\mathbf{r}_{n+1}$ to $\alpha_n$. Before going further, it is important to notice that the allowed paths are those following the same direction as the arrows in figure 1. Therefore, we have

$$\overline{\frac{\partial \mathbf{r}_{n+1}}{\partial \alpha_k}} = \overline{\frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{r}_{k+1}}} \frac{\partial \mathbf{r}_{k+1}}{\partial \alpha_k} \tag{A.4}$$

where the chain rule is applied between the total derivative of $\mathbf{r}_{n+1}$ with respect to $\mathbf{r}_{k+1}$, which depends on every possible path on figure 1 between $\mathbf{r}_{n+1}$ and $\mathbf{r}_{k+1}$, and the straight unique path derivative between $\mathbf{r}_{k+1}$ and $\alpha_k$ with $k\in[1:n-1]$. In a similar way, we derive

$$\overline{\frac{\partial \mathbf{r}_{n+1}^{(z)}}{\partial \boldsymbol{\lambda}_k}}(i,j) = \overline{\frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{p}_{k+1}}}(i,z) \frac{\partial \mathbf{p}_{k+1}}{\partial \boldsymbol{\lambda}_k}(i,j) \tag{A.5}$$

where the two indices in parenthesis next to every term represent the row and the column of the resulting matrix with $i,j,z\in[1:2M]$ and $k\in[1:n-1]$. The left-hand side of equation (A.5) shows that there is a distinct derivative for each element in the vector $\mathbf{r}_{n+1}$ with respect to each $(i,j)$ element in the $\boldsymbol{\lambda}_k$ matrix. The reason why the form of equation (A.5) is slightly different from the one of equation (A.4) is due to the fact that $\boldsymbol{\lambda}_k$ is a matrix instead of being a scalar. We then continue with the total derivative of $\mathbf{r}_{n+1}$ with respect to $\mathbf{p}_n$

$$\overline{\frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{p}_n}} = \frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{p}_n} \tag{A.6}$$

which is simply the direct path in Figure 1 between $\mathbf{r}_{n+1}$ and $\mathbf{p}_n$. Things get slightly more complicated for the general form of the derivative of $\mathbf{r}_{n+1}$ with respect to $\mathbf{p}_k$ with $k\in[2:n-1]$

$$\overline{\frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{p}_k}} = \overline{\frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{p}_{k+1}}}\left(\frac{\partial \mathbf{p}_{k+1}}{\partial \mathbf{p}_k} + \frac{\partial \mathbf{p}_{k+1}}{\partial \mathbf{s}_k}\frac{\partial \mathbf{s}_k}{\partial \mathbf{r}_{k+1}}\frac{\partial \mathbf{r}_{k+1}}{\partial \mathbf{p}_k}\right) + \overline{\frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{r}_{k+1}}}\frac{\partial \mathbf{r}_{k+1}}{\partial \mathbf{p}_k} \tag{A.7}$$

where the chain rule is first applied between the total derivative of $\mathbf{r}_{n+1}$ with respect to $\mathbf{p}_{k+1}$, which depends on every possible path on Figure 1 between these two nodes, and the straight path derivative between $\mathbf{p}_{k+1}$ and $\mathbf{p}_k$. In a similar way, the chain rule is applied between the total derivative of $\mathbf{r}_{n+1}$ with respect to $\mathbf{p}_{k+1}$ and the straight paths derivatives going through $\mathbf{p}_{k+1}$, $\mathbf{s}_k$, $\mathbf{r}_{k+1}$ and $\mathbf{p}_k$ respectively. Finally, the same process is done between the total derivative of $\mathbf{r}_{n+1}$ with respect to $\mathbf{r}_{k+1}$ and the unique straight path derivative between $\mathbf{r}_{k+1}$ and $\mathbf{p}_k$. At that point, the only unknown we are left with is the total derivative of $\mathbf{r}_{n+1}$ with respect to any other $\mathbf{r}$. We begin with

$$\overline{\frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{r}_n}} = \frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{r}_n} + \frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{p}_n}\frac{\partial \mathbf{p}_n}{\partial \mathbf{s}_{n-1}}\frac{\partial \mathbf{s}_{n-1}}{\partial \mathbf{r}_n} \tag{A.8}$$

where the first possible path is the direct one between $\mathbf{r}_{n+1}$ and $\mathbf{r}_n$ and the second possible path is the one that goes through $\mathbf{r}_{n+1}$, $\mathbf{p}_n$, $\mathbf{s}_{n-1}$ and $\mathbf{r}_n$ respectively. We then continue with

$$\overline{\frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{r}_{n-1}}} = \overline{\frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{r}_n}}\left(\frac{\partial \mathbf{r}_n}{\partial \mathbf{r}_{n-1}} + \frac{\partial \mathbf{r}_n}{\partial \mathbf{p}_{n-1}}\frac{\partial \mathbf{p}_{n-1}}{\partial \mathbf{s}_{n-2}}\frac{\partial \mathbf{s}_{n-2}}{\partial \mathbf{r}_{n-1}}\right) + \frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{p}_n}\frac{\partial \mathbf{p}_n}{\partial \mathbf{p}_{n-1}}\frac{\partial \mathbf{p}_{n-1}}{\partial \mathbf{s}_{n-2}}\frac{\partial \mathbf{s}_{n-2}}{\partial \mathbf{r}_{n-1}} \tag{A.9}$$

where the three possible paths between $\mathbf{r}_{n+1}$ and $\mathbf{r}_{n-1}$ are considered. Finally, we have the general case for $k\in[2:n-2]$

$$\overline{\frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{r}_k}} = \overline{\frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{r}_{k+1}}}\left(\frac{\partial \mathbf{r}_{k+1}}{\partial \mathbf{r}_k} + \frac{\partial \mathbf{r}_{k+1}}{\partial \mathbf{p}_k}\frac{\partial \mathbf{p}_k}{\partial \mathbf{s}_{k-1}}\frac{\partial \mathbf{s}_{k-1}}{\partial \mathbf{r}_k}\right)$$
$$+\frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{p}_n}\left(\prod_{i=n}^{k+1}\frac{\partial \mathbf{p}_i}{\partial \mathbf{p}_{i-1}}\right)\frac{\partial \mathbf{p}_k}{\partial \mathbf{s}_{k-1}}\frac{\partial \mathbf{s}_{k-1}}{\partial \mathbf{r}_k} \tag{A.10}$$
$$+\sum_{j=1}^{n-k-1}\left(\overline{\frac{\partial \mathbf{r}_{n+1}}{\partial \mathbf{r}_{n-j+1}}}\frac{\partial \mathbf{r}_{n-j+1}}{\partial \mathbf{p}_{n-j}}\prod_{z=n-j}^{k+1}\left(\frac{\partial \mathbf{p}_z}{\partial \mathbf{p}_{z-1}}\right)\right)\frac{\partial \mathbf{p}_k}{\partial \mathbf{s}_{k-1}}\frac{\partial \mathbf{s}_{k-1}}{\partial \mathbf{r}_k}$$

where all possible paths going from $\mathbf{r}_{n+1}$ to $\mathbf{r}_k$ are selected. By differentiating equations (19), (20) and (21), every symbolic derivative can be replaced by its true value

$$\begin{cases} \frac{\partial \mathbf{r}_{k+1}}{\partial \mathbf{r}_k} = \mathbf{I}, \frac{\partial \mathbf{r}_{k+1}}{\partial \mathbf{p}_k} = -\alpha_k \mathbf{A} \\ \frac{\partial \mathbf{r}_{k+1}}{\partial \alpha_k} = -\mathbf{A}\mathbf{p}_k, \frac{\partial \mathbf{s}_{k-1}}{\partial \mathbf{r}_k} = \mathbf{A} \\ \frac{\partial \mathbf{p}_{k+1}}{\partial \mathbf{p}_k} = \boldsymbol{\lambda}_k, \frac{\partial \mathbf{p}_{k+1}}{\partial \mathbf{s}_k} = \mathbf{I} \\ \frac{\partial \mathbf{p}_{k+1}}{\partial \boldsymbol{\lambda}_k} = \begin{pmatrix} \mathbf{p}_k^{(1)} & \cdots & \mathbf{p}_k^{(2\mathrm{M})} \\ \vdots & \vdots & \vdots \\ \mathbf{p}_k^{(1)} & \cdots & \mathbf{p}_k^{(2\mathrm{M})} \end{pmatrix} \in \mathbb{R}^{2M\times 2M} \end{cases} \tag{A.11}$$

where $\mathbf{I}$ is a $2M \times 2M$ identity matrix and $\frac{\partial \mathbf{p}_{k+1}}{\partial \boldsymbol{\lambda}_k}$ is a matrix with the $i^{th}$ column having a constant value corresponding to the $i^{th}$ element of $\mathbf{p}_k$.

To get better and faster results during the backpropagation process, the adaptation step for $\alpha_k$ and $\boldsymbol{\lambda}_k$ can be set to different values and a momentum technique is used on $\boldsymbol{\lambda}_k$ :

$$\boldsymbol{\lambda}_k(\mathrm{t}+1) = \boldsymbol{\lambda}_k(\mathrm{t}) - \mu_1 \overline{\frac{\partial e}{\partial \boldsymbol{\lambda}_k(t)}} + \beta\left(\boldsymbol{\lambda}_k(\mathrm{t}) - \boldsymbol{\lambda}_k(\mathrm{t}-1)\right) \tag{A.12}$$

$$\alpha_k(\mathrm{t}+1) = \alpha_k(\mathrm{t}) - \mu_2 \overline{\frac{\partial e}{\partial \alpha_k(t)}} \tag{A.13}$$

where $\mu_1$ and $\mu_2$ are the fixed adaptation step, $\beta$ is the momentum coefficient between 0 and 1 and t represents the current iteration step (epoch) in the backpropagation process.