State-dependent preconditioning for the inner-loop in Variational Data Assimilation using Machine Learning

Victor Trappler, Arthur Vidard

▶ **To cite this version:**

Victor Trappler, Arthur Vidard. State-dependent preconditioning for the inner-loop in Variational Data Assimilation using Machine Learning. 2024. hal-04707967

HAL Id: hal-04707967
https://hal.science/hal-04707967v1

Preprint submitted on 24 Sep 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

State-dependent preconditioning for the inner-loop in Variational Data Assimilation using Machine Learning

Victor Trappler*†‡ and Arthur Vidard*

Abstract. Data Assimilation is the process in which we improve the representation of the state of a physical system by combining information coming from a numerical model, real-world observations, and some prior modelling. It is widely used to model and to improve forecast systems in Earth science fields such as meteorology, oceanography and environmental sciences. One key aspect of Data assimilation is the analysis step, where the output of the numerical model is adjusted in order to account for the observational data. In Variational Data Assimilation and under Gaussian assumptions, the analysis step comes down to solving a high-dimensional non-linear least-square problem. In practice, this minimization involves successive inversions of large, and possibly ill-conditioned matrices constructed using linearizations of the forward model. In order to improve the convergence rate of these methods, and thus reduce the computational burden, preconditioning techniques are often used to get better-conditioned matrices, but require either the sparsity pattern of the matrix to inverse, or some spectral information. We propose to use Deep Neural Networks in order to construct a preconditioner. This surrogate is trained using some properties of the singular value decomposition, and is based on a dataset which can be constructed online to reduce the storage requirements.

Key words. Variational Data Assimilation, Neural Networks, Preconditioning

Introduction. Numerical models are ubiquitous nowadays as they are used to better understand and predict complex physical phenomena. In order to improve the accuracy and the predictability of those modelled systems, real-world data are assimilated into the predictions to provide a better representation of the true underlying state of the systems studied. In Data Assimilation, this process is called the analysis step, where we combine different sources of information: the forecast of the previous time window, the available direct or indirect observations of various physical quantities within this time window, and some expert knowledge on the modelled processes, such as conservation and balance laws. Due to the time critical nature of those forecasts, the sheer size of the data involved, and the large computational power required to run numerical models, Data Assimilation methods have to be efficient since every improvement in those methods can lead to the use of more precise or more complex models, for a constant time budget.

In Variational Data Assimilation, the analysis is performed by minimizing a well-chosen objective function. This optimization can be very expensive since it happens in a high-dimensional space. Nonetheless, it can be tackled with gradient-based optimization, which boils down to successive high-dimensional linear system to solve. The speed of convergence of those methods depends on the condition number of the matrices involved, that is why several studies have been conducted on the condition number of various Data Assimilation problem, such as in [Haben et al., 2011, Gürol et al., 2014, Tabeart et al., 2021].

Machine-Learning, on the other hand, has been increasingly applied on various aspects of Data Assimilation, as reviewed in [Cheng et al., 2023]. Some works focus on the Data Assimilation process, as in [Boudier et al., 2020] where the authors propose a formalism of Data Assimilation, and apply recurrent Neural networks to perform the analysis and prediction steps. Same goes for [Arcucci et al., 2021]. In [Peyron et al., 2021], an auto-encoder architecture is proposed in order to reduce the dimension of the state vector, and perform the assimilation in a lower dimensional latent space. Learning the underlying dynamical system is also of big interest. In [Gottwald and Reich, 2021], the authors propose to use Data Assimilation to learn the time-

*Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 Grenoble, France.
†AI4Sim, Eviden BDS R&D, Echirolles, France
‡Current Affiliation: École Centrale de Lyon, CNRS UMR 5208, Institut Camille Jordan, 36 Avenue Guy de Collongue, 69134 Écully, France
Corresponding author: victor.trappler@gmail.com

propagator of a dynamical system, while in [Dubois et al., 2020], the dynamics of a Lorenz system is learned. [Fablet et al., 2020]

In this work, we propose to use Deep Neural Networks (DNN) to construct a preconditioner, in order to improve the convergence of the Conjugate Gradient algorithm in a Variational Data Assimilation system. Using ML in Linear Algebra problems has recently found some traction in some related works: in [Ackmann et al., 2021], the authors build a preconditioner for an implicit solver, or in [Sappl et al., 2019, Tang et al., 2022], where a preconditioner for conjugate gradient is built using a convolutional neural network for the former, and a Graph Neural Network in the latter. In [Luna et al., 2021], the authors proposes to use Neural Networks to improve the first guess in the GMRES method. Finally, in [Häusner et al., 2023], the authors use an approach similar to ours, where a sparse factorization of a matrix is learned using Graph Neural Networks and the Frobenius norm in order to precondition the Conjugate Gradient.

We will first review the classical method to obtain the inner/outer loop paradigm for optimization in order to introduce preconditioning, and then show how preconditioners can improve the convergence rate of CG, and how those can be constructed in a efficient way.

1. Variational Data Assimilation.

In this section, we will introduce common notations used throughout this work and how Variational Data assimilation relates to solving large-scale linear systems.

1.1. Data Assimilation as an optimization problem.

We assume that the physical system studied can be represented as a n-dimensional state vector $x \in \mathbb{X} \subseteq \mathbb{R}^n$.

Let us consider a forward model $\mathcal{M}$ which maps the state-space onto itself. It usually outputs the state vector at some time in the future.

$$\begin{aligned} \mathcal{M} : \mathbb{X} \subseteq \mathbb{R}^n &\longrightarrow \mathbb{X} \\ x &\longmapsto \mathcal{M}(x) \end{aligned} \tag{1.1}$$

The output of this forward model (ie a state vector at a later time) often can not be compared directly to the observations y. Indeed the observations may come from different sources, and are sparse and noisy quantities derived from the state. An observation operator $\mathcal{H}$ is then required to map the state vector to the observation space:

$$\begin{aligned} \mathcal{H} : \mathbb{X} \subseteq \mathbb{R}^n &\longrightarrow \mathbb{Y} \subseteq \mathbb{R}^p \\ x &\longmapsto \mathcal{H}(x) \end{aligned} \tag{1.2}$$

In Data Assimilation, variational methods refer to approaches based on the optimization of an objective function, which measures the misfit between the model prediction and the observations, with a regularization that models the prior knowledge as a background term x^b and B:

$$J(x) = \frac{1}{2}\|\mathcal{G}(x) - y\|^2_{R^{-1}} + \frac{1}{2}\|x - x^b\|^2_{B^{-1}} \tag{1.3}$$

where the *Generalized forward model* is

$$\mathcal{G}(x) = (\mathcal{H} \circ \mathcal{M})(x) \tag{1.4}$$

and the vector norms are defined for $v \in \mathbb{R}^n$ and $\Sigma \in \mathbb{R}^{n\times n}$ positive definite as $\|v\|^2_\Sigma = v^T \Sigma v$.

From a probabilistic point of view, we can get to the same formulation by making the following Gaussian assumptions:

$$y \mid x \sim \mathcal{N}(\mathcal{G}(x), R) \tag{1.5}$$

$$x \sim \mathcal{N}(x^b, B) \tag{1.6}$$

which leads to the expression of the objective function of Eq. (1.3) as the negative log posterior probability of x given y.

1.2. Incremental 4D-Var. In some large-scale systems, the Tangent Linear Model (ie the linearization of the model operator) and its adjoint may be available, at the cost of proper derivation and maintenance, and at a computational cost roughly equivalent to the forward model. This means that we can consider gradient-based optimization methods in order to solve the analysis step.

Starting from a given state x, a perturbation δx gives

$$J(x+\delta x) = \frac{1}{2}\|\mathcal{G}(x+\delta x) - y\|^2_{R^{-1}} + \frac{1}{2}\|x+\delta x - x^b\|^2_{B^{-1}} \tag{1.7}$$

and linearizing $\mathcal{G}$ around x gives the incremental version of the cost function

$$J_{\text{inc}}(x,\delta x) = \frac{1}{2}\|\mathcal{G}(x) + \mathbf{G}_x\delta x - y\|^2_{R^{-1}} + \frac{1}{2}\|\delta x + x - x^b\|^2_{B^{-1}} \tag{1.8}$$

$$= \frac{1}{2}\|\mathbf{G}_x\delta x - d\|^2_{R^{-1}} + \frac{1}{2}\|\delta x + x - x^b\|^2_{B^{-1}} \tag{1.9}$$

where $d = \mathcal{G}(x) - y$ are the departures from the observations and $\mathbf{G}_x = H_{\mathcal{M}_x} M_x$ is the Jacobian matrix of $\mathcal{G}$ evaluated at x. Minimizing the incremental cost function with respect to δx is a quadratic minimization problem, and the optimal increment δx verifies

$$\underbrace{(\mathbf{G}_x^T R^{-1}\mathbf{G}_x + B^{-1})}_{\mathbf{A}_x}\delta x = \underbrace{-\mathbf{G}_x^T R^{-1} d - B^{-1}(x - x^b)}_{b_x} \tag{1.10}$$

and thus requires the resolution of a linear system of dimension n using iterative methods, since the explicit inversion of such a matrix is unfeasible in practice. A similar derivation can be achieved by applying Gauss-Newton Algorithm (see for instance [Gratton et al., 2007]), which approximates the Hessian matrix of the non-linear optimization problem by the matrix $\mathbf{A}_x$.

One can also see the incremental formulation as a Bayesian Inverse Linear problem, where we are looking for the posterior mode (or mean equivalently in this case) of $\delta x \mid d$

$$d \mid \delta x \sim \mathcal{N}(\mathbf{G}_x\delta x, R) \tag{1.11}$$

$$\delta x \sim \mathcal{N}(x - x^b, B) \tag{1.12}$$

and the posterior mean is given by solving Eq. (1.10) and the posterior covariance matrix is

$$\Gamma_{\text{post}} = \left(\mathbf{G}_x^T R^{-1}\mathbf{G}_x + B^{-1}\right)^{-1} = \mathbf{A}_x^{-1} \tag{1.13}$$

Optimal approximations of this posterior are studied in [Benner et al., 2018, Spantini et al., 2015].

1.3. Nested loops. Once the optimal increment δx has been computed, the new point of linearization is chosen as $x + \delta x$, and a new approximation can be constructed. This can be repeated until convergence, or until a specified number of linearizations has been reached.

The whole minimization procedure can be organized in nested loops, as detailed in Figure 2 and Algorithm 1.1.

- The *Outer Loop*, which requires a run of the forward model $\mathcal{G}$ at a point x, and the evaluation of the Tangent Linear Model $\mathbf{G}_x$ in order to get a linearization. The linearization of the cost function, which implies the Tangent Linear Model, can be obtained by classical methods of automatic differentiation. The number of outer loops is critical when dealing with highly non-linear processes ([Bonavita et al., 2018])
- the *Inner Loop*, where we solve the minimization problem using the TLM (ie successive quadratic approximations). Once this minimization has been performed, the point of evaluation for the Outer Loop is chosen.

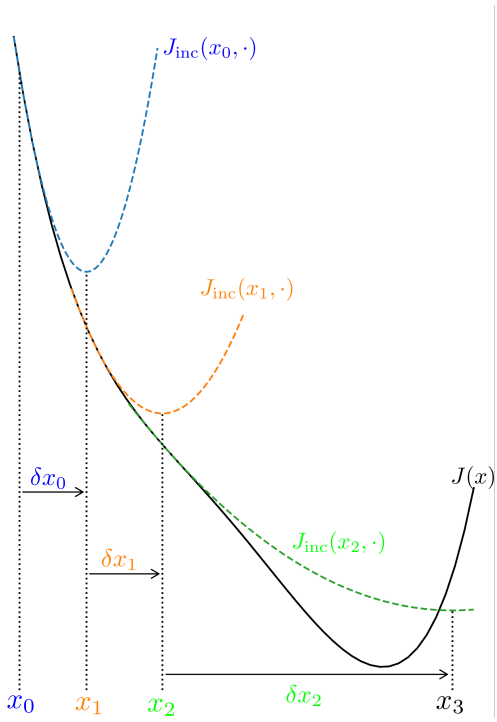

Figure 1. *Illustration of minimization using successive quadratic approximations*

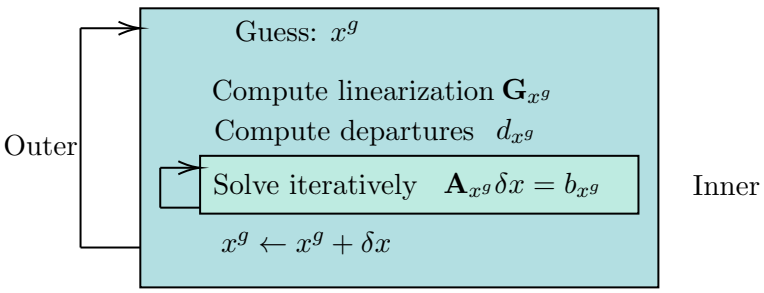

Figure 2. *Inner and outer loop paradigm for optimization*

Algorithm 1.1 Pseudocode of the minimization procedure in 4DVar

$n \leftarrow 1$
$x_i \leftarrow x_0$
while $i \leq n_{\text{outer}}$ **do** ▷ *Outer Loop*
  ▷ *Direct model and linearization at* x_i ◁
  Evaluate $\mathcal{G}(x_i)$, $J(x_i)$, $\mathbf{G}_{x_i}$
  $b_{x_i} \leftarrow -\mathbf{G}_{x_i}^T R^{-1}\left(\mathcal{G}(x_i) - y\right)$
  $\mathbf{A}_{x_i} \leftarrow (\mathbf{G}_{x_i}^T R^{-1}\mathbf{G}_{x_i} + B^{-1})$
  ▷ *The linear system to solve is* $\mathbf{A}_{x_i}\delta x_i = b_{x_i}$ ◁
  $j \leftarrow 0$, $\delta x^{(j)} \leftarrow 0$
  while $j \leq n_{\text{inner}}$ or $\|r_j\|_2 < \epsilon$ **do** ▷ *Inner Loop*
    $\delta x^{(j+1)} \leftarrow \text{ConjugateGradient}(\mathbf{A}_{x_i}, \delta x^{(j)})$
    $r_j \leftarrow \mathbf{A}_{x_i}\delta x^{(j+1)} - b_{x_i}$
    $j \leftarrow j + 1$
  $\delta x_i \leftarrow \delta x^{(j)}$
  $x_{i+1} \leftarrow x_i + \delta x_i$
  $i \leftarrow i + 1$

1.4. Conjugate Gradient. In the inner loop, the matrix $\mathbf{A}_x$ cannot be constructed explicitely let alone be inverted. We can use Krylov-subspace based methods to approximately solve the linear system, such as GMRES, or Conjugate Gradient which only require matrix-vectors products. Since the matrix to inverse is symmetric positive definite, we use the Conjugate Gradient algorithm to solve the linear system [Freitag, 2020] and the error $e_k = \delta x_k - \delta x^*$ between the computed increment at the kth step and the true value $\delta x^* = \mathbf{A}_x^{-1} b_x$ can be bounded, giving a rough rate of convergence

$$\|e_k\| \leq 2 \left(\frac{\sqrt{\kappa(\mathbf{A}_x)} - 1}{\sqrt{\kappa(\mathbf{A}_x)} + 1} \right)^k \|e_0\| \tag{1.14}$$

where $\kappa(\mathbf{A}_x) = \|\mathbf{A}_x^{-1}\|_2 \cdot \|\mathbf{A}_x\|_2 \geq 1 = \kappa(I_n)$ is the condition number of the matrix $\mathbf{A}_x$. As this matrix is symmetric positive definite, this condition number can be written as the ratio between the largest and smallest eigenvalues:

$$\kappa(\mathbf{A}_x) = \frac{\lambda_1(x)}{\lambda_n(x)} \tag{1.15}$$

where the spectrum of $\mathbf{A}_x$: $\mathrm{sp}(\mathbf{A}_x) = (\lambda_1(x), \ldots, \lambda_n(x))$ is sorted in descending order.

It is clear from Eq. (1.14) that a smaller condition number leads to a better convergence rate of the CG algorithm. Since the matrix $\mathbf{A}_x$ is fully determined by the problem, its condition number is not directly adjustable. We can however use a preconditioner in order to improve the condition number of the problem, and thus improve the convergence rate for this iterative method.

Since we are solving iteratively a system of the form $\mathbf{A}_x \delta x = b$ (the subscript i is dropped for convenience), the quantity of interest chosen to track the convergence of the Conjugate Gradient method is often the L_2 norm of the residual $e_j = \mathbf{A}_x \delta x^{(j)} - b$. However, the CG method does not guarantee a monotonic decrease of the euclidian norm of the residuals $\|e_j\|_2$, nor its energy norm $\|e_j\|_A$, which can explain some oscillations in some visualizations.

1.5. Preconditioning the Inner Loop. Instead of directly solving the linear system $\mathbf{A}_x \delta x = b$ using iterative methods, one can look for a system which possesses the same solution, ie $\mathbf{A}_x^{-1} b$, but for which the CG method converges faster. One approach is to left multiply the two sides of the equation by an invertible matrix of size $n \times n$, say L^T giving the linear system $(L^T \mathbf{A}_x)\delta x = (L^T b)$.

In order to conserve the symmetric property of the matrix to inverse and use CG, we can rewrite the linear system as

$$\underbrace{(L^T \mathbf{A}_x L)}_{\tilde{\mathbf{A}}} \underbrace{(L^{-1} \delta x)}_{\tilde{x}} = L^T b \tag{1.16}$$

If $\tilde{x} \in \mathbb{R}^n$ verifies the linear equation $\tilde{\mathbf{A}} \tilde{x} = L^T b$, the solution of the original linear system can be retrieved by $\delta x = L \tilde{x}$. The new linear system can also be preconditioned if needed, but focus here on "first-level" preconditioning.

$P\mathbf{A}_x$ and $L^T \mathbf{A}_x L$ share the same spectrum, and the matrix $P = LL^T$ is called a preconditioner while L is sometimes called a split preconditioner. Trivial examples of preconditioners include $P = I_n$ and $P = \mathbf{A}_x^{-1}$, but for the former the problem to solve is left unchanged, while for the latter the solution is found trivially, at the cost of computing directly the inverse of the matrix. The choice of a preconditioner is largely problem dependent, but some desirable properties can be listed:

- P should be symmetric and non-singular
- P should be cheap to apply as a linear operator
- P should improve the condition number of $\mathbf{A}_x$ in order to improve the convergence of iterative methods

In data assimilation, given the definition of $\mathbf{A}_x$ in Eq. (1.10) , particular choices of L can be useful to simplify the problem. Indeed, preconditioning the matrix $\mathbf{A}$ using $L = B^{-1/2}$ gives

$$\tilde{\mathbf{A}} = B^{-T/2} \mathbf{G}_x^T R^{-1} \mathbf{G}_x B^{-1/2} + I_n \tag{1.17}$$

In this case, all the eigenvalues of $\tilde{\mathbf{A}}$ are larger than 1, so its condition number is smaller than its largest eigenvalue (see [Gürol et al., 2014]).

In many cases, one may look for a solution of the linear system in a smaller subspace generated by the columns of L. This method is often named in the literature Control Variable Transform, and thus L is not a square matrix. However the two problems are not necessarily equivalent, and [Ménétrier and Auligné, 2015] studies further the conditions for equivalence. In the case of sparse matrices, a preconditioner can be found by looking for a product $P\mathbf{A}_x$ which approximates the identity matrix. That is the principle of Sparse Approximate Inverse (see [Grote and Huckle, 1997]), where the preconditioner is found by minimizing $\|I_n - P\mathbf{A}_x\|$ for P with the given sparsity pattern.

Since the convergence properties of the CG method is dependent on the distribution of the eigenvalues of the matrix $\mathbf{A}_x$, we will focus on preconditioners constructed using its spectral properties.

1.6. Spectral preconditioners. We will now drop the subscript x for notation sake, but all those quantities depend implicitely on the point of linearization x. The spectral preconditioners introduced here are studied more generally as Limited Memory Preconditioners in [Tshimanga et al., 2008]. The idea is to construct a matrix which will reduce the r largest eigenvalues of $\mathbf{A}$ to some values smaller.

Since $\mathbf{A}$ is symmetric positive definite, eigendecomposition and singular value decomposition are equivalent. Let $\mathbf{A} = U\Lambda U^T$ be the Singular Value Decomposition (SVD) of $\mathbf{A}$ with $U = (u_1 \mid u_2 \mid \cdots \mid u_n) \in \mathbb{R}^{n\times n}$ an orthonormal matrix, and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ where the λ_i are all strictly positive and sorted in descending order.

Truncating the SVD on its r first components gives the low-rank approximation of $\mathbf{A}$:

$$\mathbf{A}_r = U_r \Lambda_r U_r^T \tag{1.18}$$

where $U_r = (u_1 \mid \cdots \mid u_r) \in \mathbb{R}^{n\times r}$, and $\Lambda_r = \mathrm{diag}(\lambda_1, \ldots, \lambda_r)$.

Eckart–Young–Mirsky theorem provides another characterization of the low-rank approximation, in terms of an optimization problem, which will be used in subsection 2.1:

$$\min_{\tilde{\mathbf{A}};\mathrm{rk}(\tilde{\mathbf{A}})=r} \|\mathbf{A} - \tilde{\mathbf{A}}\|_{\mathrm{F}}^2 = \|\mathbf{A} - \mathbf{A}_r\|_{\mathrm{F}}^2 = \sum_{i=r+1}^{n} \lambda_i^2 \tag{1.19}$$

where $\|\cdot\|_{\mathrm{F}}$ is the Frobenius matrix norm defined for a matrix D as

$$\|D\|_{\mathrm{F}}^2 = \mathrm{tr}\left(DD^T\right) = \sum_{i,j} d_{ij}^2 \tag{1.20}$$

Based on the decomposition Eq. (1.18), we can define for μ and $\beta > 0$ the symmetric matrix

$$P_\alpha = \beta I_n + U_r(\mu\Lambda_r^\alpha - \beta I_r)U_r^T \tag{1.21}$$

with

- $\mu > 0$ the shift which will affect the r leading eigenvalues
- $\beta > 0$ the value which will multiply the $n - r$ other eigenvalues

We can better understand the effect of this matrix by decomposing a vector $x \in \mathbb{R}^n$, into an element $x_r \in \mathrm{range}(U_r)$, the span of the first r eigenvalues, and an element $x_\perp$ in its null-space. There exists then $w \in \mathbb{R}^r$ such that $x = x_r + x_\perp = U_r w + x_\perp$. Applying P_α gives

$$\begin{aligned} P_\alpha x &= (\beta I_n + U_r(\mu\Lambda_r^\alpha - \beta I_r)U_r^T)(U_r w + x_\perp) \\ &= U_r\left(\mu\Lambda_r^\alpha\right) w + \beta x_\perp \end{aligned} \tag{1.22}$$

thus the components in $\mathrm{range}(U_r)$ are multiplied by the diagonal matrix $\mu\Lambda_r^\alpha$, while the components in the null-space are multiplied by β.

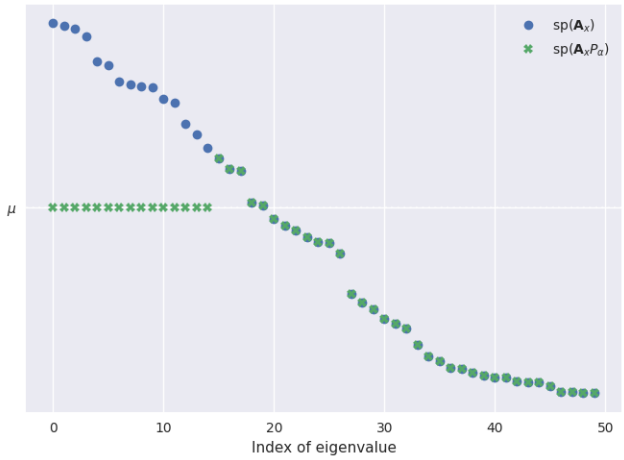

Figure 3. *Illustration of the spectrum of an example spd matrix $\mathbf{A}_x$, and the preconditioned matrix using P_α for $\alpha = -1$*

By construction, P_α is a spd matrix with spectrum

$$\mathrm{sp}(P_\alpha) = \{\mu\lambda_1^\alpha, \dots, \mu\lambda_r^\alpha, \beta \dots, \beta\} \tag{1.23}$$

and $P_{\alpha/2}$ is a matrix square root of P_α. Since $\mathbf{A}$ and P_α share the same eigenvectors, the spectrum of the product is

$$\mathrm{sp}(P_{\alpha/2}^T \mathbf{A} P_{\alpha/2}) = \mathrm{sp}(\mathbf{A}P_\alpha) = \{\mu\lambda_1^{\alpha+1} \dots, \mu\lambda_r^{\alpha+1}, \beta\lambda_{r+1} \dots, \beta\lambda_n\} \tag{1.24}$$

Choosing $\alpha = -1$ and $\beta = 1$, as in Figure 3, shifts the r leading eigenvalues of the matrix product to μ, so by choosing μ inbetween the smallest eigenvalue λ_n and λ_r, the condition number of the preconditioned matrix $\mathbf{A}P_\alpha$ is less than $\frac{\lambda_r}{\lambda_n}$.

Such a preconditioner can be used to cluster the r leading eigenvalues at μ, and thus improve the convergence rate in the Conjugate Gradient algorithm. However, a precise computation of the SVD might be challenging in practice: methods such as the Lanczos iterations require the evaluations of multiple matrix-vector products (usually more than r). Recently, randomized methods have been proposed for these kind of computations in data assimilation, see [Daužickaitė et al., 2021]. Such kind of procedures are dependent on the matrix $\mathbf{A} = \mathbf{A}_x$ at the point of linearization x, so even if some eigen-information can be reused when the linearization point does not change much ([Tshimanga et al., 2008]), most computations are discarded at the start of a new assimilation window.

Instead, we propose to use Deep Neural Networks in order to map the state of linearization x to an approximate low-rank decomposition of $\mathbf{A}_x$ which can be used as a preconditioner.

2. Deep Neural Network to construct state-dependent preconditioners. In order to construct a preconditioner based on Eq. (1.21), we need a matrix of size $n \times r$, whose columns are orthonormal to approximate U_r, and a vector of size r, with positive elements to approximate Λ_r. We propose to use a Deep Neural Network, parameterized by θ, in order to compute those quantities. Given $x \in \mathbb{R}^n$, the Neural Network outputs both a set of r non-orthonormal vectors $\tilde{U}_\theta(x) \in \mathbb{R}^{n\times r}$, and a vector $\tilde{\Lambda}_\theta(x) \in \mathcal{R}^r$ of positive approximated eigenvalues. To ensure the orthonormal property of the vectors, we use the QR decomposition, which is numerically stable compared to a classical Gram-Schmidt orthonormalization procedure, while the positivity of the approximate eigenvalues is imposed using any function $\mathbb{R} \to \mathbb{R}^+$ elementwise. In this work, we will use a scaled sigmoid function: $x \longmapsto \frac{M}{1+e^{-x}}$, where M can be chosen as a rough upper bound on the singular values of $\mathbf{A}_x$. This mapping is summarized Eq. (2.1), and Figure 4.

$$\begin{array}{ccccc} \mathbb{R}^n & \longrightarrow & \mathbb{R}^{(n+1)r} & \longrightarrow & \mathbb{R}^{n\times r} \times \mathbb{R}^r_+ \\ x & \overset{\text{DNN}}{\longmapsto} & f_\theta(x) & \overset{\text{split}}{\longmapsto} & (U_\theta(x), \Lambda_\theta(x)) = \left(\mathtt{qr}(\tilde{U}_\theta(x)), \exp\left(\tilde{\Lambda}_\theta(x)\right)\right) \end{array} \tag{2.1}$$

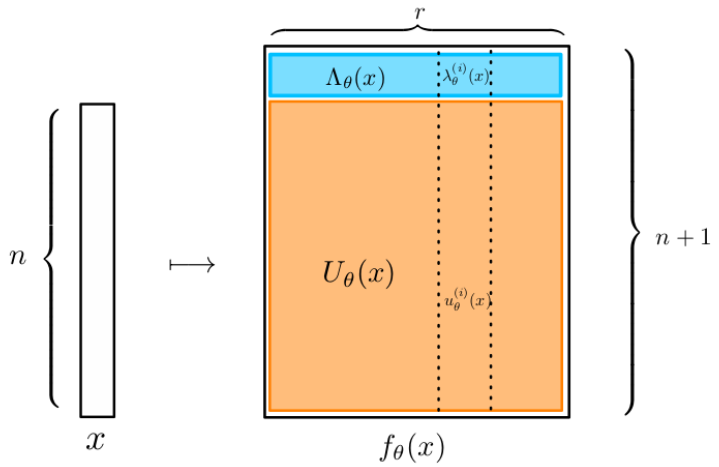

Figure 4. *Schematic representation of the input/output signature of the Neural Network*

Given the output of f_θ, the Neural Network-based low-rank reconstruction of rank r is

$$\mathbf{A}_\theta(x) = U_\theta(x)\Lambda_\theta(x)U_\theta(x)^T \tag{2.2}$$

$$= \sum_{i=1}^{r} \lambda_\theta^{(i)}(x) u_\theta^{(i)}(x) \left(u_\theta^{(i)}(x)\right)^T \tag{2.3}$$

with $U_\theta(x) = \left(u_\theta^{(1)}(x) \mid \cdots \mid u_\theta^{(r)}(x)\right)$, and a split preconditioner can be defined as in Eq. (1.21) for $\beta = 1$ and $\alpha = -1/2$:

$$L_\theta(x) = I_n + U_\theta(x)\left(\mu\Lambda_\theta(x)^{-1/2} - I_r\right)U_\theta(x)^T \quad \text{with } \mu \geq 1 \tag{2.4}$$

In theory, if the DNN provides the optimal low-rank approximation of $\mathbf{A}_x$, choosing $\mu = 1$ would allow to shift all the r first eigenvalues to 1, thus reducing the condition number of the matrix. In practice, the DNN only produces an approximation of the eigenvectors and of the eigenvalues, meaning that there is a risk to worsen the condition number. Experiments have shown that choosing μ chosen between $\min_i \lambda_\theta^{(i)}(x)$ and $\max_i \lambda_\theta^{(i)}(x)$ helps to account for the approximation error due to the DNN. This is further discussed in section 3.

2.1. Loss definition and norm estimation. Neural networks are parameterized by $\theta \in \mathbb{R}^{\mathfrak{N}}$, which combines all the weights and biases of the individual neurons of f_θ. To set this parameter, one need to define an appropriate metric which is then optimized. Given the Eckart–Young–Mirsky theorem Eq. (1.19), which defines the SVD in terms of an optimization problem and the reconstruction defined in Eq. (2.2), we define the loss for a single state of linearization x_i as

$$\mathcal{L}_{\text{explicit}}(\theta; x_i) = \|\mathbf{A}_\theta(x_i) - \mathbf{A}_{x_i}\|_{\text{F}}^2 \tag{2.5}$$

where this term would be minimized for $\mathbf{A}_\theta(x_i)$ as the low-rank approximation of $\mathbf{A}_{x_i}$.

This loss requires the evaluation of the norm of the difference of two $n \times n$ non-sparse matrices, which brings several challenges. Constructing the matrix $\mathbf{A}_{x_i}$ is computationally expensive, since in most differentiated computer codes, this matrix is only accessible as an operator. In Data Assimilation especially, given the definition of $\mathbf{A}_x$ in Eq. (1.10), computing $\delta x \mapsto \mathbf{A}_x \delta x$ requires the applications of two linear (with respect to the second argument) operators: The Tangent Linear operator

$$\text{TL} : (x_i, \delta x) \longmapsto \mathbf{G}_{x_i} \cdot \delta x \tag{2.6}$$

and the adjoint operator

$$\text{Adj} : (x_i, y) \longmapsto \mathbf{G}_{x_i}^T \cdot y \tag{2.7}$$

From a computational point of view, applying one of those operators is within the same order of magnitude of complexity as the forward model $\mathcal{G}$. Obviously, in order to construct the full Jacobian matrix $\mathbf{A}_x$, one could apply the linear operator to each vector e_i of the canonical basis since $A_x = (A_x e_1 \mid \ldots, \mid A_x e_n)$, but this is impractical since it requires n evaluations, on top of the large memory requirements needed to store the matrix $\mathbf{A}_x$ for a single linearization point.

Same goes for the matrix $\mathbf{A}_\theta(x)$: constructing the full matrix is hard from a storage point of view, even though using it as a linear operator is cheaper since it requires only r dot products of n-dimensional vectors as seen from Eq. (2.3),

Since we are only interested in the Frobenius norm of the difference of the operators, we can instead directly estimate it using statistical estimators. Let D be a real matrix of size $n \times n$. Its squared norm $\|D\|^2$ can be rewritten as the expectation of a vector norm using the linearity of the trace and expectation operator:

$$\mathbb{E}_\xi \left[\|D\xi\|^2\right] = \mathbb{E}_\xi \left[\text{tr}\left(D\xi\xi^T D^T\right)\right] = \text{tr}\left(\mathbb{E}_\xi \left[\xi\xi^T\right] D^T D\right) = \|D\|_{\text{F}}^2 \tag{2.8}$$

where $\xi \sim \mathcal{N}(0, I_n)$. Given a matrix $Z \in \mathbb{R}^{n\times k}$ whose k columns $z^{(j)}$ are sampled from a standard Gaussian distribution, we can use a Monte-Carlo estimator of the expectation:

$$\frac{1}{k}\|DZ\|_{\text{F}}^2 = \frac{1}{k}\sum_{j=1}^{k} \|Dz^{(j)}\|^2 \quad \text{estimator of} \quad \|D\|_{\text{F}}^2 \tag{2.9}$$

Other estimators of this norm using random samples are studied in [Gudmundsson et al., 1995, Gratton and Titley-Peloquin, 2018], while in [Indyk et al., 2019], the authors use ML to construct the matrix to evaluate.

Using Eq. (2.9), for a state-vector x_i in the training dataset and $z_i^{(1)}, \ldots z_i^{(k)}$ i.i.d. samples of a standard Gaussian random variable, an estimate of the matrix norm of Eq. (2.5) is

$$\mathcal{L}(\theta; x_i) = \frac{1}{k}\sum_{j=1}^{k} \|\mathbf{A}_\theta(x_i) z_i^{(j)} - \mathbf{A}_{x_i} z_i^{(j)}\|^2 \tag{2.10}$$

where $\mathbf{A}_\theta(x_i)$ is defined as in Eq. (2.2). We can also use the same estimator in order to estimate the norm of $\mathbf{A}_x$ as $\frac{1}{k}\sum_{j=1}^{k} \|\mathbf{A}_x z^{(j)}\|^2$, which is an estimate of the sum of all its eigenvalues squared. This can be used in order to normalize the loss in Eq. (2.10), and can be interpreted as the fraction of *unexplained* variance, by analogy with classical Principal Components Analysis:

$$\mathcal{L}_{\text{relative}}(\theta, x_i) = \frac{\mathcal{L}(\theta; x_i)}{\frac{1}{k}\sum_{j=1}^{k} \|\mathbf{A}_{x_i} z_i^{(j)}\|^2} \tag{2.11}$$

2.2. Dataset Construction. In order to train the Neural Network, the construction of a dataset is needed in order to optimize the loss function defined in Eq. (2.10). Each element (indexed by i) in this dataset consists of three elements: a state x_i which is used for the linearization, a random matrix $Z_i = (z_i^{(1)} \mid \cdots \mid z_i^{(k)}) \in \mathbb{R}^{n\times k}$ whose components are iid and normally distributed, and finally the evaluation of this sample by the matrix of interest: $\mathbf{A}_{x_i} Z_i$. The training dataset is then

$$\mathfrak{D}_{\text{training}} = \left\{(x_i, Z_i, \mathbf{A}_{x_i} Z_i)) \in \mathbb{R}^n \times \mathbb{R}^{n\times k} \times \mathbb{R}^{n\times k} \quad \text{s.t.} \quad 1 \leq i \leq N_{\text{training}}\right\} \tag{2.12}$$

However, we do not have to store all the training set in memory: Z_i is independent of x_i, and can be sampled when needed, and $\mathbf{A}_{x_i}$ depends only on x_i.

Algorithm 2.1 Pseudocode for the generation of a batch for online training

for $1 \leq i \leq n_{\text{batch}}$ **do**
  Sample and store $Z_i = (z_i^{(1)} | \dots | z_i^{(k)}) \in \mathbb{R}^{n \times k}$ with $z_i^{(j)} \sim \mathcal{N}(0, I_n)$ iid for $1 \leq j \leq k$
  Compute and store $\mathbf{A}_{x_i} Z_i \in \mathbb{R}^{n \times k}$
  $x_{i+1} \leftarrow$ New state generated from x_i

The method to generate a batch of n_{batch} samples is summarized Algorithm 2.1. In order to train a Deep Neural Network, the constructed batches should be representative enough of the whole state space. To get appropriate diversity in the states used to build the batch, we propose to generate the new state iteratively by advancing the current state using the numerical model $\mathcal{M}$ with a randomly generated lead time, large enough so that the x_i used for the batch are not too correlated, and by potentially adding a small random perturbation before propagation.

3. Application to a Shallow Water Assimilation system.

3.1. Shallow Water equations and Data Assimilation setting. The Shallow Water equations describe the motion of large bodies of water, for which the horizontal scale is larger than the vertical scale which is the case for rivers, seas and oceans. They consist in PDEs obtained by vertically averaging the Navier-Stokes equations. In this application, the variables of interest are the deviation of sea surface height η around a mean height H_0, the velocity u in the x-direction, and v, the velocity in the y-direction.

$$\left\{ \begin{array}{rl} \frac{\partial \eta}{\partial t} + \frac{\partial (H_0+\eta)u}{\partial x} + \frac{\partial (H_0+\eta)v}{\partial y} & = 0 \\ \frac{\partial u}{\partial t} - \xi v + \frac{\partial B}{\partial x} & = \nu \Delta u - c_b u + \frac{\tau_x}{\rho_0 h_0} \\ \frac{\partial v}{\partial t} + \xi u + \frac{\partial B}{\partial y} & = \nu \Delta v - c_b v \end{array} \right. \tag{3.1}$$

Those equations are discretized using a Arakawa C-grid of 64×64 cells, on a square domain of size $L_x = L_y = 1800$km, meaning that the three prognostic variables are $\eta \in \mathbb{R}^{64 \times 64}$, $u \in \mathbb{R}^{63 \times 64}$ and $v \in \mathbb{R}^{64 \times 63}$. Once flattened and concatenated, the state vector is then $x = (\eta, u, v) \in \mathbb{R}^{12160}$. Explicitely storing the Gauss-Newton matrix would require 4.7GB (without exploiting the symmetry).

We consider the model $\mathcal{M}$ that simulates the evolution of the state vector with a lead time of T corresponding to 2 days.

$$\begin{array}{rcl} \mathbb{R}^n & \longrightarrow & \mathbb{R}^n \\ \mathcal{M} : x_t & \longmapsto & \mathcal{M}(x_t) = x_{t+T} \end{array} \tag{3.2}$$

The cost function is defined as in Eq. (1.3)

$$J(x) = \frac{1}{2} \|(\mathcal{H} \circ \mathcal{G})(x) - y\|^2_{R^{-1}} + \frac{1}{2} \|x - x^b\|^2_{B^{-1}} \tag{3.3}$$

where $\mathcal{H}(x) = \mathcal{H}((\eta, u, v)) = \eta$, $R = I_{64^2}$, meaning that only the free-surface height is observed. The background state $x^b \in \mathbb{R}^n$ is computed as the average of states obtained during a previous simulation with a large lead time.

3.2. Neural Network Architecture. For this problem, the state vector represents three spatial variable, arranged on a regular grid. By padding the u and the v component, we can reshape the state vector as a tensor of shape $(64, 64, 3)$, ie like an image with 3 channels. Each of those components is scaled so that each channel has approximately unit variance. Because of this image-like structure, we can use Neural Network architecture well-suited for such data, such as Convolutional Neural Networks (CNN) or U-Nets. We found that using a U-Net architecture, with transformers instead of CNN for the subsampling step has shown good results.

3.3. Dataset and training. The training dataset is constructed according to Eq. (2.12), where $N_{\text{training}} = 1000$ states of linearization have been sampled, and $k = 100$ random vectors have been used for matrix-vector products.

3.4. Numerical Results. We chose to train a DNN for $r_{\text{train}} = 2000$, and compare the preconditioners obtained using different numbers of retained vectors: $r = 1000$, 1900 and 2000. For each of those, different values of μ have been chosen: either it is set to a fixed value, or it is set to the smallest eigenvalue provided by the DNN. For $r = 1000$, $\min_i \lambda^{(i)} \approx 160$, for $r = 1900$, $\min_i \lambda^{(i)} \approx 145$, and finally, for $r = 2000$, $\min_i \lambda^{(i)} \approx 2$. The matrices to inverse have their leading eigenvalues close to 20000, and show approximatively an exponential decay.

In order to compare numerical results, we generated the true states by advancing the model using a random number of time steps. Based on this true state, observations are generated by applying the observation operator and adding a sampled noise.

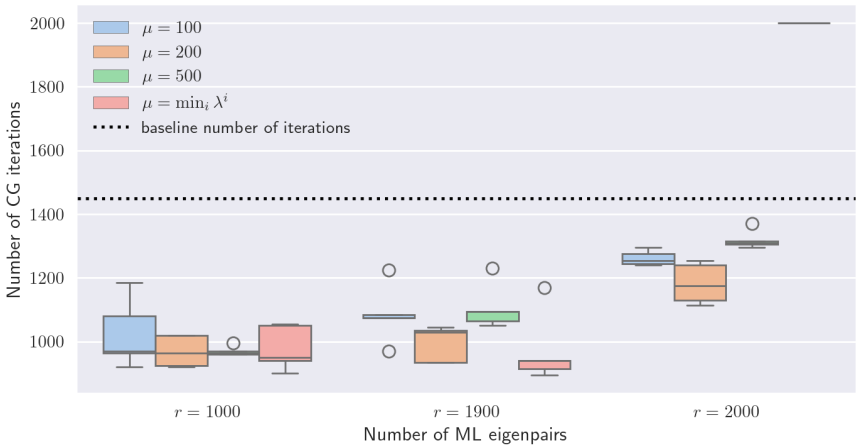

Figure 5. *Number of iterations needed to reach the norm threshold. Dotted line indicates the number of iterations for the unprecondition problem.*

Figure 5 shows the number of iterations needed to reach the threshold of 10^{-7}, or when 2000 iterations has been reached (whichever comes first) for the different preconditioners constructed using DNN.

Figure 6 shows the comparison of the L_2 norm of the residuals for the unpreconditioned problem, which acts as a baseline, and the preconditioned problem using a state-dependent preconditioner, depending on the number of retained vectors (denoted as the rank r, even though P_α is full-rank). It is worth noting that due to the form of the reconstruction Eq. (2.3), the individual contribution of each eigenpairs gets smaller and smaller, making them more and more difficult to approximate. This in turn might worsen the quality of the preconditioner because of the negative exponent α/ We can see this effect on the preconditioners built with $r = 2000$, the whole estimated spectrum. Some of the smallest eigenvalues are not well represented by the Neural Network, and this worsen the preconditioning effect of P_α, compared to $r = 1000$ or $r = 1900$, and the influence of the shift parameter μ is amplified. Indeed, μ helps mitigate this issue due to the approximation error of the Deep Neural Network, by forcing the resulting eigenvalues to be larger than 1, which acts as a lower bound for the eigenvalues of the original matrix.

This parameter is chosen whether with the fixed values $\mu = 100$, 200, 500, or automatically set to the minimum of the eigenvalues $\min_i \lambda^{(i)}$ used to construct the preconditioner. If the Deep Neural Network represents well the whole spectrum used, setting the shift to $\min_i \lambda^{(i)}$ is a sensible choice for performances, and the number of iterations required to reach the threshold is decreased by roughly 30%.

Conclusion and perspectives. In this work, we focused on the problem of data-driven preconditioning of non-sparse parameterized matrices. In a Data Assimilation context, more specifically in the incremental formulation of 4D-Var, the inner loops refer to the iterations of Conjugate Gradient to solve a high-dimensional linear system which depends on the point of linearization. In order to

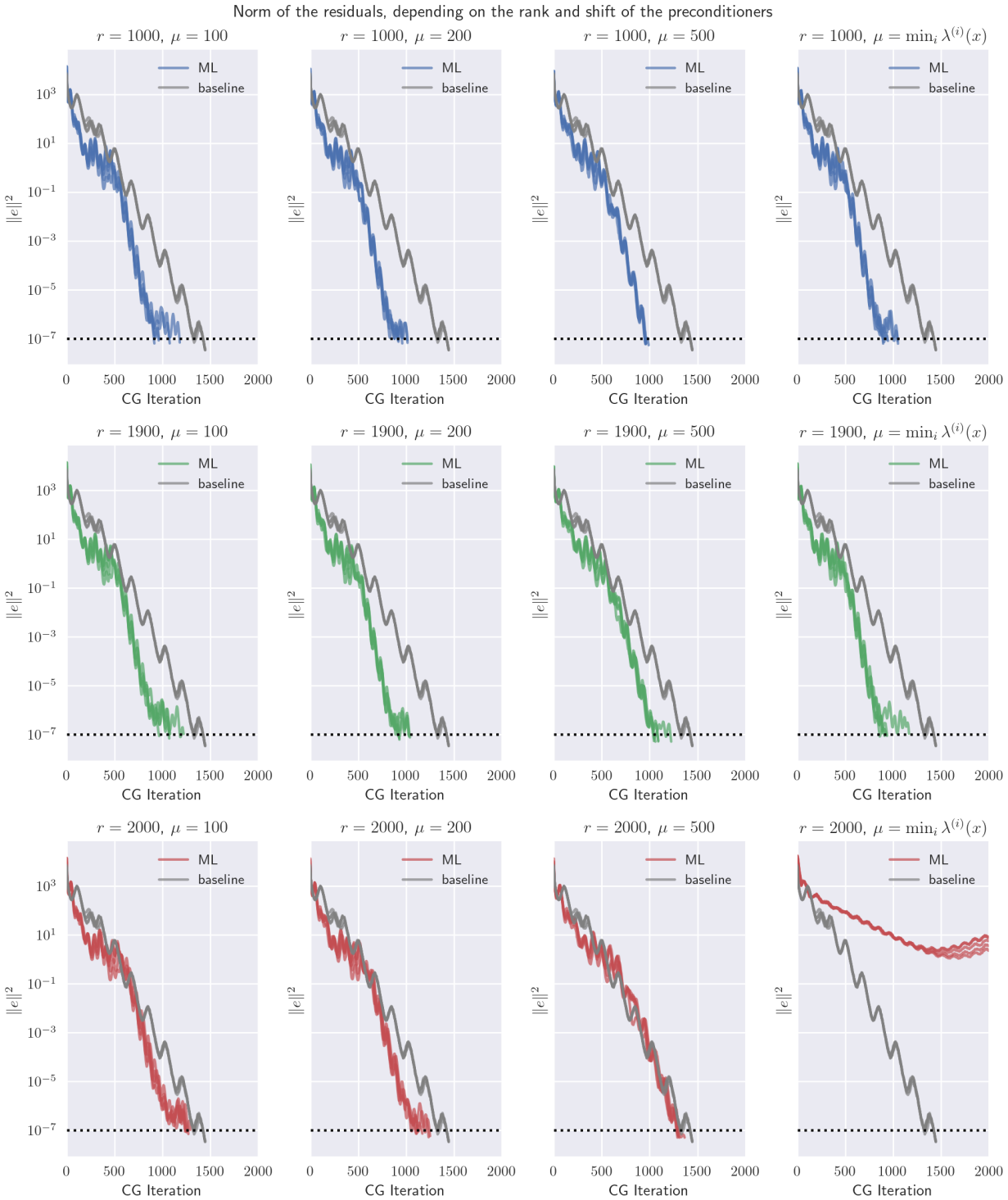

Figure 6. *Norm of the residuals depending on the CG iteration, number of retained vectors r and shift parameter μ*

improve the rate of convergence of Conjugate Gradient, we propose to use Deep Neural Networks to get an approximation of the largest eigenpairs of the matrix to inverse, and then use those to precondition the linear system.

We applied this method to an academic assimilation system of moderate size. Based on the image-like structure of the state vector, we used an architecture based on U-Nets to construct a surrogate. We have shown that using this preconditioner, we could reduced the number of matrix-

vector products required to reach a convergence threshold. Compared to traditional preconditioning methods, training such a neural network can be partly done in a almost non-intrusive way. Once trained, this can be used as a first-level preconditioner, and thus traditional randomized methods can be applied to improve furthermore the convergence rates.

Acknowledgement. This work has been funded within the France Relance Economic plan, and has been jointly done between Eviden and Inria.

REFERENCES

[Ackmann et al., 2021] Ackmann, J., Düben, P., Palmer, T., and Smolarkiewicz, P. (2021). Machine-Learned Preconditioners for Linear Solvers in Geophysical Fluid Flows. In *EGU General Assembly Conference*, pages EGU21–5507.

[Arcucci et al., 2021] Arcucci, R., Zhu, J., Hu, S., and Guo, Y.-K. (2021). Deep Data Assimilation: Integrating Deep Learning with Data Assimilation. *Applied Sciences*, 11(3):1114.

[Benner et al., 2018] Benner, P., Qiu, Y., and Stoll, M. (2018). Low-rank computation of posterior covariance matrices in Bayesian inverse problems. *SIAM/ASA Journal on Uncertainty Quantification*, 6(2):965–989.

[Bonavita et al., 2018] Bonavita, M., Lean, P., and Holm, E. (2018). Nonlinear effects in 4D-Var. *Nonlinear Processes in Geophysics*, 25(3):713–729.

[Boudier et al., 2020] Boudier, P., Fillion, A., Gratton, S., and Gürol, S. (2020). DAN – An optimal Data Assimilation framework based on machine learning Recurrent Networks. *arXiv:2010.09694 [cs, eess]*.

[Cheng et al., 2023] Cheng, S., Quilodran-Casas, C., Ouala, S., Farchi, A., Liu, C., Tandeo, P., Fablet, R., Lucor, D., Iooss, B., Brajard, J., Xiao, D., Janjic, T., Ding, W., Guo, Y., Carrassi, A., Bocquet, M., and Arcucci, R. (2023). Machine learning with data assimilation and uncertainty quantification for dynamical systems: A review.

[Daužickaitė et al., 2021] Daužickaitė, I., Lawless, A. S., Scott, J. A., and van Leeuwen, P. J. (2021). Randomised preconditioning for the forcing formulation of weak constraint 4D-Var. *Quarterly Journal of the Royal Meteorological Society*, 147(740):3719–3734.

[Dubois et al., 2020] Dubois, P., Gomez, T., Planckaert, L., and Perret, L. (2020). Data-driven predictions of the Lorenz system. *Physica D: Nonlinear Phenomena*, 408:132495.

[Fablet et al., 2020] Fablet, R., Drumetz, L., and Rousseau, F. (2020). Joint learning of variational representations and solvers for inverse problems with partially-observed data. *arXiv:2006.03653 [cs, eess, stat]*.

[Freitag, 2020] Freitag, M. A. (2020). Numerical linear algebra in data assimilation. *GAMM-Mitteilungen*, 43(3):e202000014.

[Gottwald and Reich, 2021] Gottwald, G. A. and Reich, S. (2021). Combining machine learning and data assimilation to forecast dynamical systems from noisy partial observations. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(10):101103.

[Gratton et al., 2007] Gratton, S., Lawless, A. S., and Nichols, N. K. (2007). Approximate Gauss–Newton Methods for Nonlinear Least Squares Problems. *SIAM Journal on Optimization*, 18(1):106–132.

[Gratton and Titley-Peloquin, 2018] Gratton, S. and Titley-Peloquin, D. (2018). Improved Bounds for Small-Sample Estimation. *SIAM Journal on Matrix Analysis and Applications*, 39(2):922–931.

[Grote and Huckle, 1997] Grote, M. J. and Huckle, T. (1997). Parallel Preconditioning with Sparse Approximate Inverses. *SIAM Journal on Scientific Computing*, 18(3):838–853.

[Gudmundsson et al., 1995] Gudmundsson, T., Kenney, C. S., and Laub, A. J. (1995). Small-Sample Statistical Estimates for Matrix Norms. *SIAM Journal on Matrix Analysis and Applications*, 16(3):17.

[Gürol et al., 2014] Gürol, S., Weaver, A. T., Moore, A. M., Piacentini, A., Arango, H. G., and Gratton, S. (2014). **B** -preconditioned minimization algorithms for variational data assimilation with the dual formulation: **B** -preconditioned minimization algorithms. *Quarterly Journal of the Royal Meteorological Society*, 140(679):539–556.

[Haben et al., 2011] Haben, S., Lawless, A., and Nichols, N. (2011). Conditioning and preconditioning of the variational data assimilation problem. *Computers & Fluids*, 46(1):252–256.

[Häusner et al., 2023] Häusner, P., Öktem, O., and Sjölund, J. (2023). Neural incomplete factorization: Learning preconditioners for the conjugate gradient method.

[Indyk et al., 2019] Indyk, P., Vakilian, A., and Yuan, Y. (2019). Learning-Based Low-Rank Approximations.

[Luna et al., 2021] Luna, K., Klymko, K., and Blaschke, J. P. (2021). Accelerating GMRES with Deep Learning in Real-Time.

[Ménétrier and Auligné, 2015] Ménétrier, B. and Auligné, T. (2015). An Overlooked Issue of Variational Data Assimilation. *Monthly Weather Review*, 143(10):3925–3930.

[Peyron et al., 2021] Peyron, M., Fillion, A., Gürol, S., Marchais, V., Gratton, S., Boudier, P., and Goret, G. (2021). Latent Space Data Assimilation by using Deep Learning. *arXiv:2104.00430 [cs, math]*.

[Sappl et al., 2019] Sappl, J., Seiler, L., Harders, M., and Rauch, W. (2019). *Deep Learning of Preconditioners for Conjugate Gradient Solvers in Urban Water Related Problems*.

[Spantini et al., 2015] Spantini, A., Solonen, A., Cui, T., Martin, J., Tenorio, L., and Marzouk, Y. (2015). Optimal low-rank approximations of Bayesian linear inverse problems. *arXiv:1407.3463 [math, stat]*.

[Tabeart et al., 2021] Tabeart, J. M., Dance, S. L., Lawless, A. S., Nichols, N. K., and Waller, J. A. (2021). New bounds on the condition number of the Hessian of the preconditioned variational data assimilation problem.

[Tang et al., 2022] Tang, Z., Zhang, H., and Chen, J. (2022). Graph Neural Networks for Selection of Preconditioners and Krylov Solvers. In *NeurIPS 2022 Workshop: New Frontiers in Graph Learning*.

[Tshimanga et al., 2008] Tshimanga, J., Gratton, S., Weaver, A. T., and Sartenaer, A. (2008). Limited-memory preconditioners, with application to incremental four-dimensional variational data assimilation. *Quarterly Journal of the Royal Meteorological Society*, 134(632):751–769.