1 Data-driven GMRES

By following the same steps as in GCROT (Generalized Conjugate Residual with inner Orthogonalization adn outer Truncation), an efficient update rule can be developed for the data-driven GMRES iteration (defined by Eq. 3 of the main text). In this section, we provide more details on the derivation of this update rule.

Notation and prelimnaries:

1. The system of equations being solved will be denoted by $Af = b$, with $f$ being the unknown vector being solved for. We also denote by $D$ the size of the system of equations i.e. $A \in \mathbb{C}^{D \times D}$ and $f, b \in \mathbb{C}^D$. It will be assumed that $A$ is invertible.

2. Given the vectors $v_1, v_2, \ldots, v_N$ with which GMRES has to be accelerated (which are assumed to be linearly independent, but not necessarily orthogonal), we will denote by $V$ the matrix that is formed with these vectors as its columns. Note that $V \in \mathbb{C}^{D \times N}$ and $\text{span}(v_1, v_2, \ldots, v_N) = \text{range}(V)$.

3. $K_n(A, b)$ will denote the Krylov subspace of dimensionality $n$ that is generated by the matrix $A$ and the vector $b$: $K_n(A, b) = \text{span}(b, Ab, A^2b, \ldots, A^{n-1}b)$.

4. $\tilde{A}$ and $\tilde{b}$ are defined by:

\begin{align}
\tilde{A} &= P_\perp(Av_1, Av_2, \ldots, Av_N)A \\
\tilde{b} &= P_\perp(Av_1, Av_2, \ldots, Av_N)b
\end{align}

where $P_\perp(Av_1, Av_2, \ldots, Av_N)$ is the operator projecting a vector out of $\text{span}(Av_1, Av_2, \ldots, Av_N)$. For convenience, we will denote this operator by just $P_\perp$. We note that, in general, $\tilde{A}$ is not sparse even if $A$ is sparse, but for small $N$ multiplication of $\tilde{A}$ with a vector can be computed efficiently by first multiplying the vector by $A$, followed by projecting the resulting vector out of $\text{span}(Av_1, Av_2, \ldots, Av_N)$.

5. To conveniently work with $P_\perp$, we perform an incomplete QR decomposition on the matrix $AV$ to obtain an orthogonal matrix $C \in \mathbb{C}^{D \times N}$ and an upper triangular matrix $R \in \mathbb{C}^{N \times N}$: $AV = CR$. It then immediately follows that $P_\perp = I - CC^\dagger$. Moreover, it is also convenient to precompute and store $R^{-1}$ (Note that if $A$ is invertible, and $v_1, v_2, \ldots, v_N$ are linearly independent then $R$ is invertible).

Arnoldi iteration: The $i^{th}$ iteration of data-driven GMRES approximates the solution to $Af = b$ with $f_i$, where $f_i$ is given by:

\begin{equation}
    f_i = \arg\min_{f \in \text{range}(V) \oplus K_i(\tilde{A}, \tilde{b})} ||Af - b||^2
\end{equation}
One of the key ingredients of the GMRES iteration is the Arnoldi iteration which generates an orthonormal basis for the Krylov subspace $\mathcal{K}_{i+1}(A, b)$ from the orthonormal basis for the Krylov subspace $\mathcal{K}_i(A, b)$. Denoting the orthonormal basis for $\mathcal{K}_i(A, b)$ by \( \{q_1, q_2 \ldots q_i\} \), note that span\( \{q_1, q_2 \ldots q_i, Aq_i\} = \mathcal{K}_{i+1}(A, b) \). Therefore, \( q_{i+1} \) can be computed by orthonormalizing \( \tilde{A} q_i \) against \( \{q_1, q_2 \ldots q_i\} \):

\[
q_{i+1} = \frac{v_{i+1}}{\|v_{i+1}\|}, \text{ where } v_{i+1} = \tilde{A} q_i - \sum_{j=1}^{i} (q_j^\top \tilde{A} q_i) q_j
\] (3)

In our implementation, we assume \( q_1 = \tilde{b}/\|\tilde{b}\| \), and use Eq. 3 to generate \( q_2, q_3 \ldots \) and so on. Note that \( q_i \perp \text{span}(Av_1, Av_2 \ldots Av_N) \) \( \forall i \), or equivalently \( C^\dagger Q_i = 0 \) \( \forall i \). Denoting by \( Q_i \) the matrix formed with the vectors \( q_1, q_2 \ldots q_i \) as its columns \( (Q_i \in \mathbb{C}^{D \times i}) \), the Arnoldi iteration can be expressed as the following relationship between \( Q_{i+1} \) and \( Q_i \):

\[
\tilde{A} Q_i = Q_{i+1} H_{i,i+1} \implies A Q_i = Q_{i+1} H_{i,i+1} + C C^\dagger A Q_i
\] (4)

where \( H_{i,i+1} \in \mathbb{C}^{(i+1)\times i} \) is an upper Hessenberg matrix defined by:

\[
H_{i,i+1} = \begin{bmatrix}
q_1^\top \tilde{A} q_1 & q_1^\top \tilde{A} q_2 & q_1^\top \tilde{A} q_3 & \cdots & q_1^\top \tilde{A} q_i \\
\|v_2\| & q_2^\top \tilde{A} q_2 & q_2^\top \tilde{A} q_3 & \cdots & q_2^\top \tilde{A} q_i \\
0 & \|v_3\| & q_3^\top \tilde{A} q_3 & \cdots & q_3^\top \tilde{A} q_i \\
0 & 0 & \|v_4\| & \cdots & q_4^\top \tilde{A} q_i \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & \|v_{i+1}\|
\end{bmatrix}
\] (5)

**Calculating \( f_i \):** Consider now solving the optimization problem in Eq. 2. Since the optimization variable \( f \) is in the space range\((V) \oplus \mathcal{K}_i(A, b)\), it can be expressed as:

\[
f = VR^{-1}x + Q_i y
\] (6)

where \( x \in \mathbb{C}^N \) and \( y \in \mathbb{C}^i \). Thus, it follows that:

\[
\|Af - b\|^2 = \|AVR^{-1}x + A Q_i y - b\|^2 = \left\| C \begin{bmatrix} I & C^\dagger A Q_i \\ 0 & H_{i,i+1} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} - b \right\|^2
\] (7)

wherein we have used \( AV = CR \) and Eq. 4. Note that since \( C \) and \( Q_{i+1} \) are independently orthogonal matrix, and \( C^\dagger Q_i = 0 \), it follows that \( [C \begin{bmatrix} I & C^\dagger A Q_i \\ 0 & H_{i,i+1} \end{bmatrix}] \) is an orthogonal matrix. Eq. 7 can now be further simplified to:

\[
\|Af - b\|^2 = \left\| \begin{bmatrix} I & C^\dagger A Q_i \\ 0 & H_{i,i+1} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} C^\dagger b \\ Q_{i+1}^\dagger b \end{bmatrix} \right\|^2 + \|(I - CC^\dagger - Q_{i+1} Q_{i+1}^\dagger) b\|^2
\] (8)

Therefore, \( f_i = VR^{-1}x_i + Q_i y_i \), where

\[
x_i, y_i = \arg\min_{x,y} \left\| \begin{bmatrix} I & C^\dagger A Q_i \\ 0 & H_{i,i+1} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} C^\dagger b \\ Q_{i+1}^\dagger b \end{bmatrix} \right\|^2
\] (9)

We have thus reduced the problem of calculating \( f_i \), which was a constrained least squares problem, to an unconstrained least squares problem (Eq. 9) of size \( i + N \), which can be solved numerically (e.g. using QR factorization).
Figure 1: Performance of the different preconditioners described in section 2 on the evaluation dataset. (a) Jacobi preconditioner, (b) Gauss-Siedel preconditioner, (c) preconditioner from ref. [17] of main text (d) Symmetric over-relaxation (SOR) preconditioner for different relaxation parameter $\omega$ and (e) Incomplete LU preconditioner for different drop tolerances.

2 Benchmarks for data-free preconditioners

Here we present the results of applying some data-free preconditioners on the simulation problem. Given a left preconditioner $P_L$ and a right preconditioner $P_R$, the system of equations being solved is transformed from $Af = b$ to $A'f' = b'$ where:

$$A' = P_L A P_R, \quad b' = P_L b \quad \text{and} \quad f' = P_R^{-1} f$$

We study the following four preconditioners:

1. **Jacobi preconditioner**: The Jacobi preconditioner [1] is given by:

$$P_L = D(A)^{-1} \quad \text{and} \quad P_R = I$$

where $D(A)$ is a diagonal matrix formed from the diagonal entries of the matrix $A$. The performance of Jacobi preconditioner on the evaluation dataset is shown in Fig. 1(a).

2. **Gauss-Siedel preconditioner**: The Gauss-Siedel preconditioner is given by:

$$P_L = [D(A) + L(A)]^{-1} \quad \text{and} \quad P_R = I$$

where $L(A)$ is a strictly lower-triangular matrix formed by the elements of $A$ below the main diagonal. Note that application of this preconditioner requires the solution a lower triangular system of equations. The performance of the Gauss-Siedel preconditioner on the evaluation dataset is shown in Fig. 1(b).

3. **Preconditioner from ref. [17]**: This preconditioner is specific to Maxwell’s equations. $P_R$ and $P_L$ are diagonal matrices constructed from the grid spacing (including the complex stretching due to PMLs) in the simulation domain. Details of this preconditioner can be found in ref. [17] for main text. The performance of this preconditioner on the evaluation dataset is shown in Fig. 1(c).
4. **Symmetric over-relaxation (SOR) preconditioner:** The SOR preconditioner [1] is given by:

\[ P_L = [D(A) + \omega L(A)]^{-1} \text{ and } P_R = I \]  

where \( L(A) \) is a strictly lower-triangular matrix formed by the elements of \( A \) below the main diagonal and \( \omega \) is a tunable parameter (referred to as the relaxation parameter) in the preconditioner which can be between 0 to 2. Note that the SOR preconditioner for \( \omega = 1 \) is identical to the Gauss-Siedel preconditioner. Additionally, application of the SOR preconditioner requires the solution of a lower triangular system of equations. The performance of the SOR preconditioner on the evaluation dataset is shown in Fig. 1(d) — we see that the best performance of SOR preconditioner on our dataset is achieved for \( \omega = 1.25 \).

5. **Incomplete LU:** This preconditioner seeks an upper and lower triangular matrix, \( U \) and \( L \) such that the product \( LU \) is approximately equal to the matrix \( A \) [1]. The preconditioner is then given by:

\[ P_L = U^{-1} L^{-1} \text{ and } P_R = I \]  

The deviation of the \( LU \) from \( A \) is controlled with a drop tolerance parameter — a larger drop tolerance implies a faster computation of \( L \) and \( U \) but a worse approximation to \( A \) and therefore a less useful preconditioner. The performance of incomplete LU preconditioner on the evaluation dataset is shown in Fig. 1(e) for drop tolerances of 0.1, 0.01 and 0.001.

**References**

[1] Yousef Saad. *Iterative methods for sparse linear systems*, volume 82. siam, 2003.