Graph Neural Network-based Preconditioners for Optimizing GMRES

Aleix Nieto Juscafresa

Department of Information Technology
Division of Systems and Control

May 30, 2024

Supervisor: Paul Häusner
Subject reviewer: Jens Sjölund

UPPSALA
UNIVERSITET

Contents

Linear systems of equations

For a given matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ and vector $\boldsymbol{b} \in \mathbb{R}^n$, we consider the problem of finding a vector $\boldsymbol{x} \in \mathbb{R}^n$, such that

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{1}$$

where n is large and $\boldsymbol{A}$ is sparse.

Countless fields demand solving (1). For example:

- Optimization
- Signal processing and machine learning
- Computational biology and physics
- Economics

Problem

Solve $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ efficiently!

How are we trying to solve the problem?

If $\boldsymbol{A}$ is nonsingular, the unique solution to the linear system is:

$$\boldsymbol{x}^* = \boldsymbol{A}^{-1}\boldsymbol{b}.$$

Classical solving methods to find the unique solution $\boldsymbol{x}^* \in \mathbb{R}^n$:

Direct methods

- Conventional way of solving: $\boldsymbol{x}^* = \boldsymbol{A}^{-1}\boldsymbol{b}$.
- Involves matrix inversion or factorization.

Iterative methods

- Repeatedly refine an initial guess to approximate solutions.
- Efficient where approximations are sufficient.

Sparse matrices

Definition

A matrix with most elements being zero, usually $\mathcal{O}(n)$, where special techniques can be utilized to take advantage of the large number of zero elements and their locations.

- **Examples:**
  - Finite element method (FEM) for PDEs.
- **Relation to graphs:**
  - Rows/columns as nodes, nonzero entries as edges.
- **Storage schemes:**
  - Coordinate list (COO).
  - Compressed sparse row (CSR).
  - Compressed sparse column (CSC).

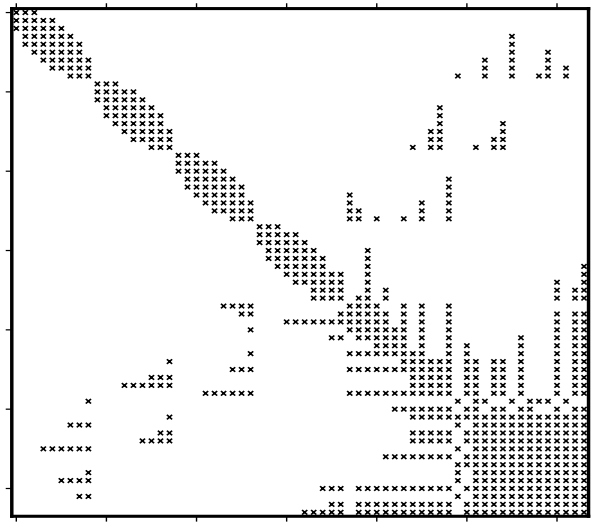


Figure 1: Sparse matrix from a Poisson PDE problem.

Scalability issues with direct solvers

Direct solvers and the problem with $\boldsymbol{A}^{-1}$

- **Complexity**: Computing the inverse of a matrix $\boldsymbol{A}^{-1}$ via direct methods (e.g., Gauss-Jordan elimination) has a complexity of $\mathcal{O}(n^3)$.
- **Fill-in effect**: Factorization can increase the number of non-zero elements, raising memory and computational costs.

Direct solvers are accurate and reliable but become impractically slow for very large, sparse systems.

Solution

Iterative methods.

Benefits of iterative methods for large and sparse matrices

Role of sparsity in iterative methods

- **General complexity**: For sparse matrices, the complexity of iterative methods like GMRES is typically $\approx \mathcal{O}(n^2)$, much lower than direct methods.
- **Storage**: The matrix $\boldsymbol{A}$ does not to be stored explicitly. We only know $\boldsymbol{A}$ by matrix-vector products.

Preconditioning

The reliability of iterative solvers depends heavily on the conditioning of the linear system.

Solution

Preconditioning.

Improve the spectral properties of the equation system by solving $\boldsymbol{PAx} = \boldsymbol{Pb}$ instead for a suitable matrix $\boldsymbol{P}$.

- Preconditioners improve solver efficiency by approximating the inverse of matrix $\boldsymbol{A}$ with the matrix $\boldsymbol{P}$.
- Depends on matrix properties: symmetry, positive definiteness, diagonal dominance,...

Challenges with traditional preconditioners

- **Traditional preconditioners**:
  - Reliance on heuristics.
  - Limited effectiveness in new domains.
- **Emerging paradigm**:
  - Data-driven heuristics using neural networks.[1]
  - Potential for improved performance.

Thesis motivation

Leverage GNNs for optimized preconditioning strategies.

[1] Paul Häusner/Ozan Öktem/Jens Sjölund: Neural incomplete factorization: learning preconditioners for the conjugate gradient method, 2023.

Approach

- Combine data-driven methods with sparse matrix theory.
- Represent coefficient matrix as a graph.
- Train a GNN to predict a suitable preconditioner.

Research questions:

- *Can we accelerate the GMRES algorithm by creating a preconditioner parameterized by a neural network that is trained against data?*
- *Will this approach outperform established preconditioning techniques like incomplete LU or Jacobi?*
- *How does this trained preconditioner perform in problems arising from different contexts?*

Krylov subspace

- Introduced in the 1950s by Hestenes, Stiefel, Lanczos.

Definition:

For a matrix $\boldsymbol{A}$ and a vector $\boldsymbol{b}$, the k-th Krylov subspace, denoted as $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{b})$, is defined as

$$\text{span}\{\boldsymbol{b}, \boldsymbol{A}\boldsymbol{b}, \ldots, \boldsymbol{A}^{k-1}\boldsymbol{b}\}.$$

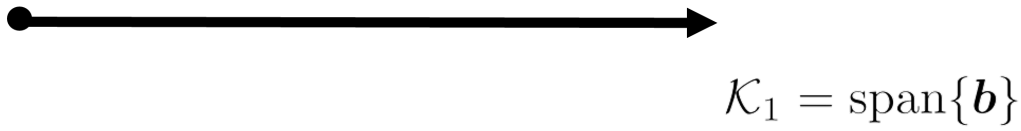
Introduction to the method: GMRES

The GMRES[2] is a Krylov subspace method that computes at the k-th step, the best least squares solution from the Krylov subspace $\mathcal{K}_k(A, b)$.

[2] Yousef Saad: Iterative methods for sparse linear systems, 2003.

Introduction to the method: GMRES

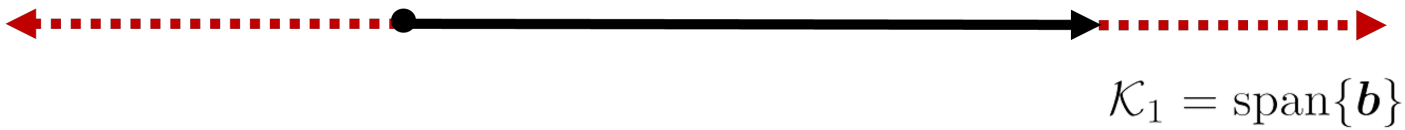
The GMRES[2] is a Krylov subspace method that computes at the k-th step, the best least squares solution from the Krylov subspace $\mathcal{K}_k(A, b)$.



[2] Yousef Saad: Iterative methods for sparse linear systems, 2003.

Introduction to the method: GMRES

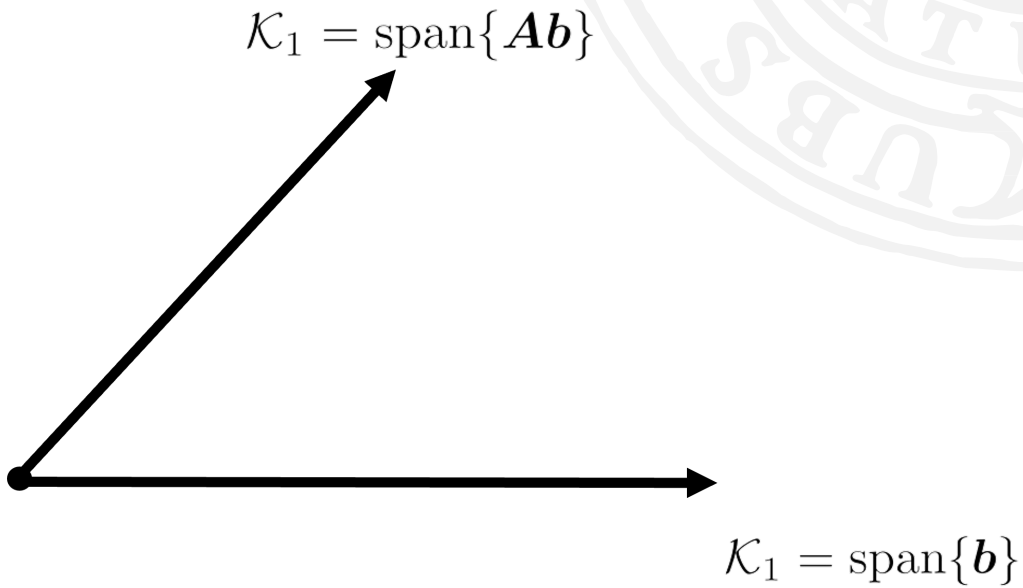
The GMRES[2] is a Krylov subspace method that computes at the k-th step, the best least squares solution from the Krylov subspace $\mathcal{K}_k(A, b)$



[2] Yousef Saad: Iterative methods for sparse linear systems, 2003.

Introduction to the method: GMRES

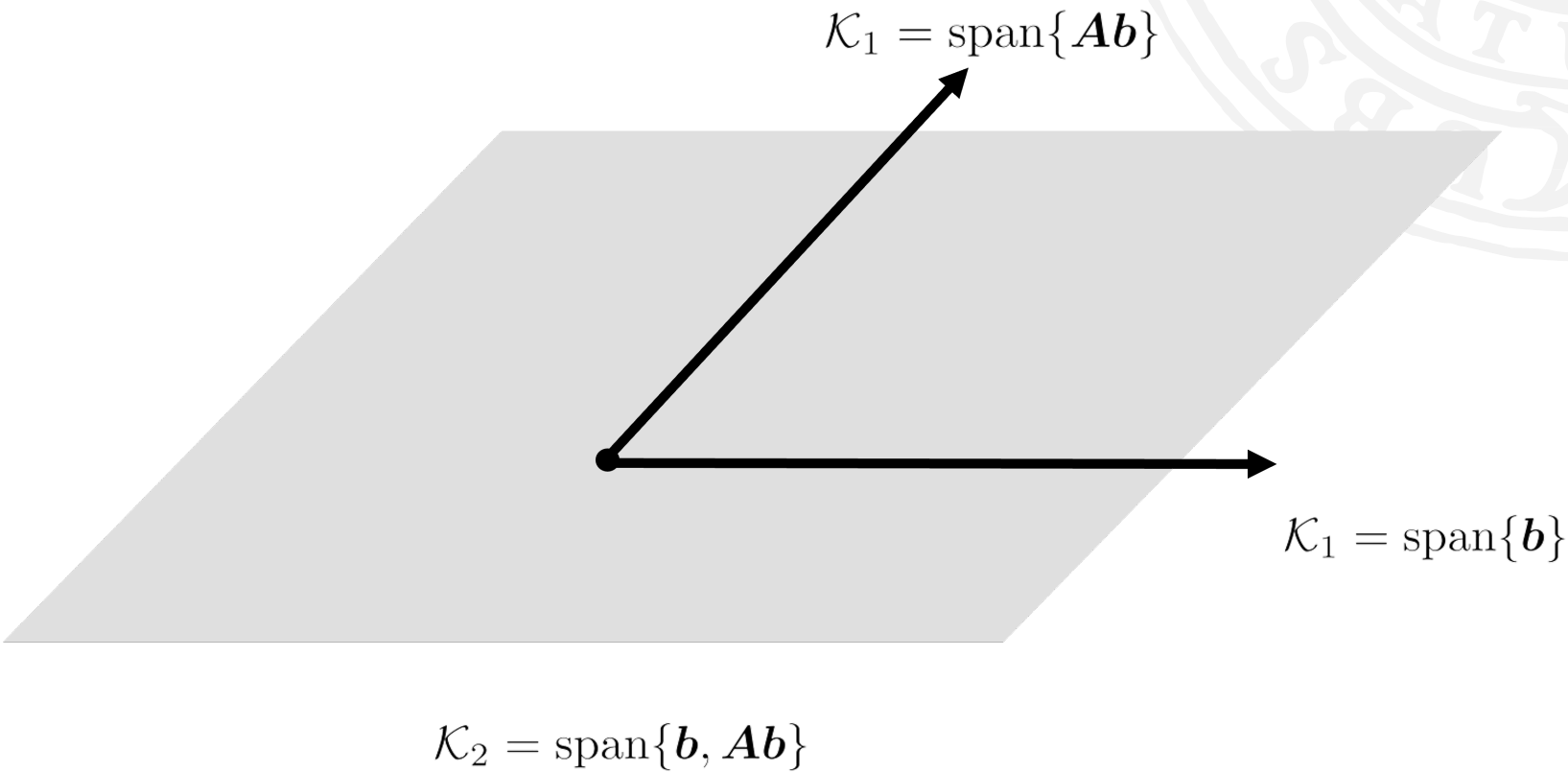
The GMRES[2] is a Krylov subspace method that computes at the k-th step, the best least squares solution from the Krylov subspace $\mathcal{K}_k(A, b)$



[2] Yousef Saad: Iterative methods for sparse linear systems, 2003.

Introduction to the method: GMRES

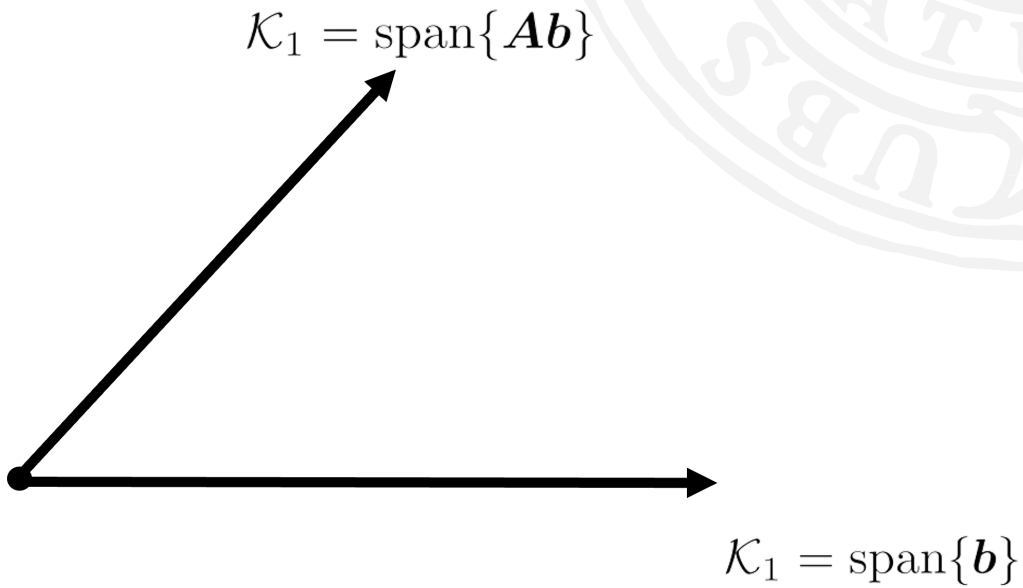
The GMRES[2] is a Krylov subspace method that computes at the k-th step, the best least squares solution from the Krylov subspace $\mathcal{K}_k(A, b)$



[2] Yousef Saad: Iterative methods for sparse linear systems, 2003.

Introduction to the method: GMRES

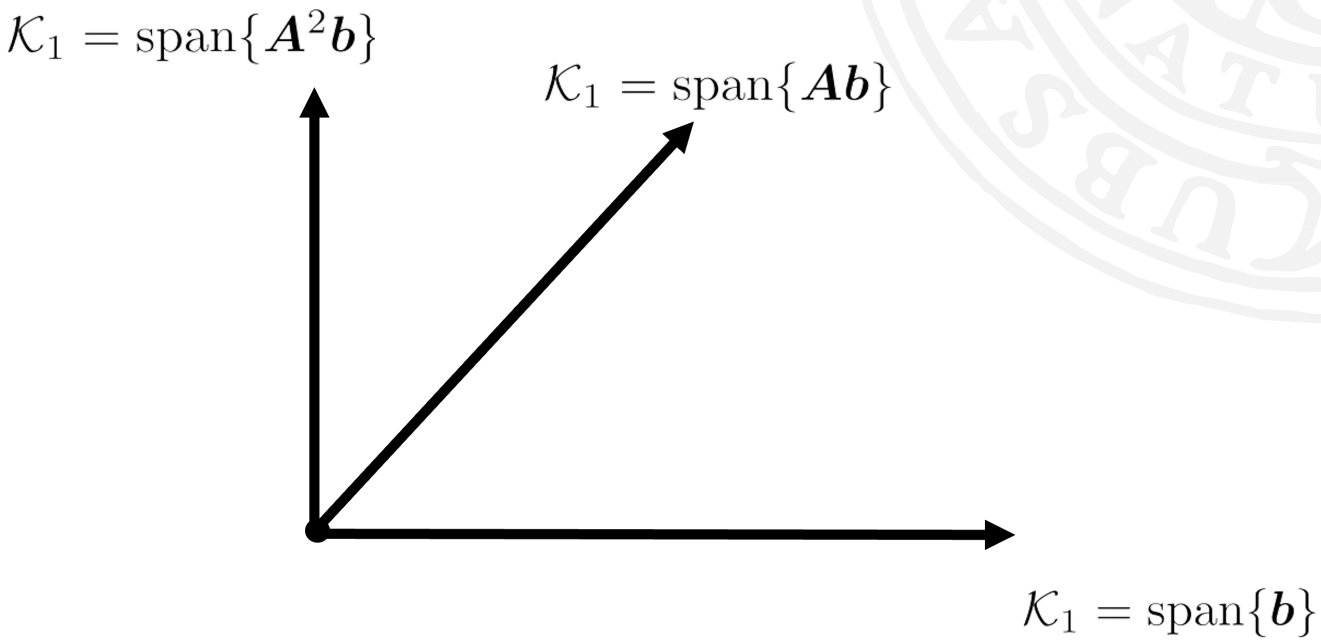
The GMRES[2] is a Krylov subspace method that computes at the k-th step, the best least squares solution from the Krylov subspace $\mathcal{K}_k(A, b)$



[2] Yousef Saad: Iterative methods for sparse linear systems, 2003.

Introduction to the method: GMRES

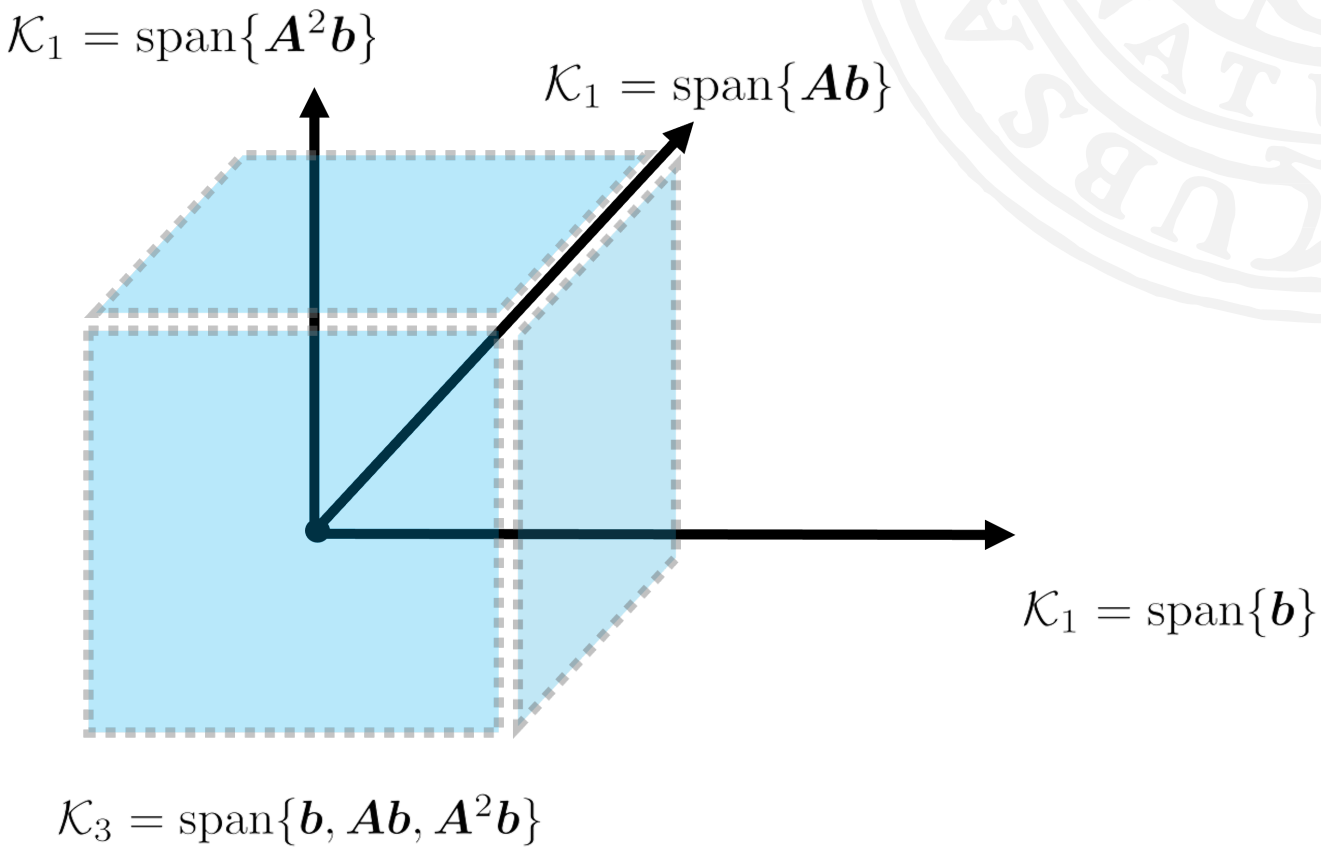
The GMRES[2] is a Krylov subspace method that computes at the k-th step, the best least squares solution from the Krylov subspace $\mathcal{K}_k(A, b)$



[2] Yousef Saad: Iterative methods for sparse linear systems, 2003.

Introduction to the method: GMRES

The GMRES[2] is a Krylov subspace method that computes at the k-th step, the best least squares solution from the Krylov subspace $\mathcal{K}_k(A, b)$



[2] Yousef Saad: Iterative methods for sparse linear systems, 2003.

GMRES assumptions

Until now we did not assume any special structure of $\boldsymbol{A} \rightarrow$ GMRES works for arbitrary matrices.

Assumptions for matrix $\boldsymbol{A}$

- Invertible.
- Sparse.
- Diagonal elements different from 0.

Recall

The objective of the thesis is to learn a data-driven preconditioner for GMRES.

Learned preconditioner - learning problem

Final goal:

Learn a parameterized mapping $f_{\boldsymbol{\theta}} : \mathbb{F}^{n\times n} \to \mathbb{F}^{n\times n}$, that takes a *full rank* matrix $\boldsymbol{A}$ and predicts a suitable preconditioner.

- *But how, do we **enforce invertibility** on the output of the mapping?*

Inspired by incomplete LU factorization we first learn another mapping

$$\begin{aligned} \Lambda_{\boldsymbol{\theta}} : \mathbb{F}^{n\times n} &\to \mathcal{L}_n^* \times \mathcal{U}_n^* \\ \boldsymbol{A} &\mapsto (L_{\boldsymbol{\theta}}(\boldsymbol{A}), U_{\boldsymbol{\theta}}(\boldsymbol{A})) \end{aligned}$$

that takes a full rank matrix $\boldsymbol{A} \in \mathbb{F}^{n\times n}$ as input and outputs two matrices $L_{\boldsymbol{\theta}}(\boldsymbol{A})$ and $U_{\boldsymbol{\theta}}(\boldsymbol{A})$ within the sets of lower and upper triangular matrices of order n with **nonzero diagonal elements** denoted by $\mathcal{L}_n^*$ and $\mathcal{U}_n^*$.

Learned preconditioner - learning problem

The function $\Lambda_{\boldsymbol{\theta}}$ generates a factorization of the preconditioning matrix:

$$f_{\boldsymbol{\theta}}(\boldsymbol{A}) = L_{\boldsymbol{\theta}}(\boldsymbol{A})U_{\boldsymbol{\theta}}(\boldsymbol{A}).$$

$f_{\boldsymbol{\theta}}(\boldsymbol{A})$ is full rank

$L_{\boldsymbol{\theta}}(\boldsymbol{A})$ and $U_{\boldsymbol{\theta}}(\boldsymbol{A})$ are invertible and the product of two invertible matrices is also invertible and an invertible matrix is full rank.

The inverse of $f_{\boldsymbol{\theta}}$ is therefore computationally efficient since as applying forward-backward substitution needs just $\mathcal{O}(n^2)$ operations[2].

[2] G.H. Golub/C.F. Van Loan: Matrix computations, (Johns Hopkins studies in the mathematical sciences), 2013.

Learned preconditioner - optimization problem

To learn a suitable approximation model we assume that the matrices of interest are generated by a matrix-valued random variable $\mathbb{A}$, with an unknown distribution that describes the underlying class of problems.

Our objective is to mimic the sparse factorization methods with no fill-ins. We train a function to approximate a sparse factor

$$\hat{\theta} \in \arg\min \mathbb{E}_{\boldsymbol{A}\sim\mathbb{A}} \left[||L_{\theta}(\boldsymbol{A})U_{\theta}(\boldsymbol{A}) - A||_F\right]$$

$$\text{s.t.} \begin{cases} L_{\theta}(\boldsymbol{A})_{ij} = 0 \text{ if } \boldsymbol{A}_{ij} = 0, \\ U_{\theta}(\boldsymbol{A})_{ij} = 0 \text{ if } \boldsymbol{A}_{ij} = 0. \end{cases}$$

Learned preconditioner - optimization problem

A practical issue with $||L_\theta(\boldsymbol{A})U_\theta(\boldsymbol{A}) - \boldsymbol{A}||_F$ is that computing its expectation $\mathbb{E}_{\boldsymbol{A}\sim\mathbb{A}}$ is impossible due to the unavailability of the distribution of $\mathbb{A}$. However, we do have access to training data $\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_m \sim \mathbb{A}$. Therefore, we opt for its empirical version:

$$\hat{\theta} \in \arg\min \frac{1}{m} \sum_{k=1}^{m} ||L_\theta(\boldsymbol{A}_k)U_\theta(\boldsymbol{A}_k) - \boldsymbol{A}_k||_F.$$

To enhance computational efficiency we circumvent computing the matrix-matrix multiplication $L_\theta(\boldsymbol{A})U_\theta(\boldsymbol{A})$ in the objective, by approximating the Frobenius norm using Hutchinson's trace estimator as

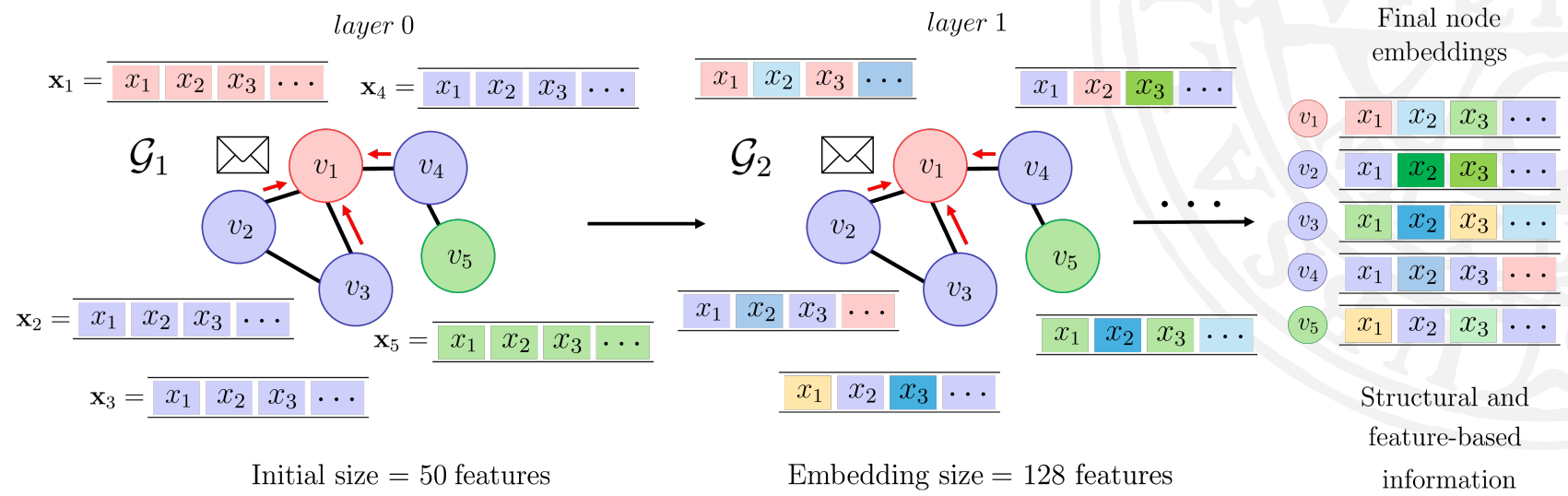
$$||\boldsymbol{B}||_F^2 = \text{trace}(\boldsymbol{B}^\top \boldsymbol{B}) \approx \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{w}_i^\top \boldsymbol{B}^\top \boldsymbol{w}_i,$$

where w_i are iid normally distributed random variables. Now, we obtain an unbiased estimator of the loss which requires only matrix-vector products.

Model architecture

But how do we parameterize $\Lambda_{\boldsymbol{\theta}} : \mathbb{F}^{n \times n} \rightarrow \mathcal{L}_n^* \times \mathcal{U}_n^*$? We use a graph neural network (GNN) as computational backend.

GNN: example


layer 0
layer 1
Final node embeddings
Structural and feature-based information
Initial size = 50 features
Embedding size = 128 features
AGGREGATE
UPDATE
AGGREGATE
UPDATE

Model architecture

- Every matrix $\boldsymbol{A}$ can be interpreted as a weighted digraph (Coates graph).

$$\boldsymbol{A} = \begin{bmatrix} 1 & -1 & 0 & 2 \\ 1 & -1 & -1 & 0 \\ 2 & 0 & 2 & -1 \\ -1 & 0 & 0 & 0 \end{bmatrix}$$

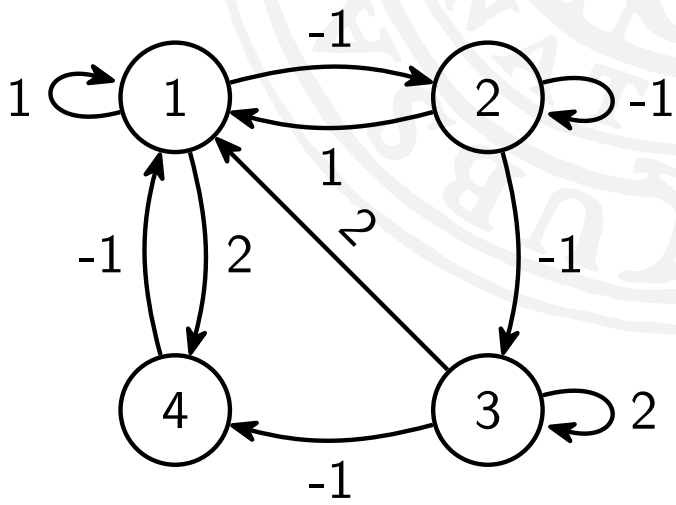


Figure 2: Coates digraph of matrix $\boldsymbol{A}$.

- We use these insights to construct our GNN architecture.
- GNNs are a natural choice to parameterize functions on sparse matrices.

Model architecture

$$\boldsymbol{A} = \begin{bmatrix} 1 & -1 & 0 & 2 \\ 1 & -1 & -1 & 0 \\ 2 & 0 & 2 & -1 \\ -1 & 0 & 0 & 0 \end{bmatrix} \longrightarrow \mathcal{E}_{ij}^{(0)} = \begin{bmatrix} 1 & 1 & 0 \\ -1 & -1 & 1 \\ 2 & 2 & 1 \\ 1 & 1 & -1 \\ -1 & -1 & 0 \\ -1 & -1 & 1 \\ 2 & 2 & -1 \\ 2 & 2 & 0 \\ -1 & -1 & 1 \\ -1 & -1 & -1 \end{bmatrix}$$

Model architecture

$$\mathcal{E}_{ij}^{(0)} = \begin{bmatrix} 1 & 1 & 0 \\ -1 & -1 & 1 \\ 2 & 2 & 1 \\ 1 & 1 & -1 \\ -1 & -1 & 0 \\ -1 & -1 & 1 \\ 2 & 2 & -1 \\ 2 & 2 & 0 \\ -1 & -1 & 1 \\ -1 & -1 & -1 \end{bmatrix} \xrightarrow[L\text{-layer GNN}]{} \mathcal{E}_{ij}^{(L)} = \begin{bmatrix} e_{1,1} & e_{1,2} & e_{1,3} \\ e_{2,1} & e_{2,2} & e_{2,3} \\ e_{3,1} & e_{3,2} & e_{3,3} \\ e_{4,1} & e_{4,2} & e_{4,3} \\ e_{5,1} & e_{5,2} & e_{5,3} \\ e_{6,1} & e_{6,2} & e_{6,3} \\ e_{7,1} & e_{7,2} & e_{7,3} \\ e_{8,1} & e_{8,2} & e_{8,3} \\ e_{9,1} & e_{9,2} & e_{9,3} \\ e_{10,1} & e_{10,2} & e_{10,3} \end{bmatrix}$$

Model architecture

$$\mathcal{E}_{ij}^{(L)} = \begin{bmatrix} e_{1,1} & e_{1,2} & e_{1,3} \\ e_{2,1} & e_{2,2} & e_{2,3} \\ e_{3,1} & e_{3,2} & e_{3,3} \\ e_{4,1} & e_{4,2} & e_{4,3} \\ e_{5,1} & e_{5,2} & e_{5,3} \\ e_{6,1} & e_{6,2} & e_{6,3} \\ e_{7,1} & e_{7,2} & e_{7,3} \\ e_{8,1} & e_{8,2} & e_{8,3} \\ e_{9,1} & e_{9,2} & e_{9,3} \\ e_{10,1} & e_{10,2} & e_{10,3} \end{bmatrix} \longrightarrow \begin{matrix} \boldsymbol{e_{1,1}} & \boldsymbol{e_{1,2}} \\ e_{2,1} & \boldsymbol{e_{2,2}} \\ e_{3,1} & \boldsymbol{e_{3,2}} \\ \boldsymbol{e_{4,1}} & e_{4,2} \\ \boldsymbol{e_{5,1}} & \boldsymbol{e_{5,2}} \\ e_{6,1} & \boldsymbol{e_{6,2}} \\ \boldsymbol{e_{7,1}} & e_{7,2} \\ \boldsymbol{e_{8,1}} & \boldsymbol{e_{8,2}} \\ e_{9,1} & \boldsymbol{e_{9,2}} \\ \underbrace{\boldsymbol{e_{10,1}}}_{\text{edge feature of } L} & \underbrace{e_{10,2}}_{\text{edge feature of } U} \end{matrix}$$

Datasets

Currently two problem settings: synthetic and Poisson PDE.

Synthetic matrices ($n = 1000$)

Randomized matrix with 99.5% of sparsity.

PDE matrices ($n = 2500$)

- Perturbed Poisson PDE with $> 99.5\%$ of sparsity.
- Perturbed Poisson PDE + randomized matrix with $> 97.3\%$ of sparsity.

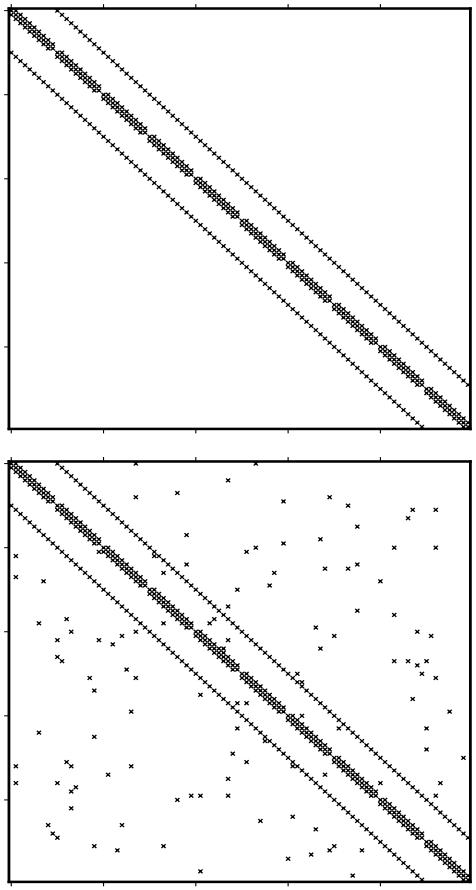


Figure 3: PDE matrices.

Implementation details

GMRES implementation

- Both its (left) preconditioned and nonpreconditioned without restarting.
- Only matrix-vector and vector-vector products.
- Leveraging matrix sparsity in CSR and COO format.

Learned preconditioner implementation

- Data-driven preconditioner trained on 1000 problems.
- Validated on 100 samples.

Baseline preconditioners

The learned preconditioner will be compared against Jacobi, Gauss-Seidel, symmetric Gauss-Seidel, and ILU(0) preconditioners.

Results for synthetic data (GMRES convergence)

Preconditioner	Iterations	P-time	GMRES-time	Total time
None	810.73	-	36.9770	36.9770
Jacobi	481.85	**0.0003**	11.8322	11.832
Gauss-Seidel	676.00	0.0005	22.1245	22.1250
Sym. Gauss-Seidel	294.43	0.0010	5.3966	5.3977
ILU(0)	**221.64**	0.0012	**3.3709**	**3.3721**
Learned	254.01	0.0055	4.4950	4.5005

Table 1: Mean results for the synthetic dataset on 100 test instances.

Results for PDE data

Preconditioner	Iterations	P-time	GMRES time	total time	Success
None	904.00	-	60.2389	60.2389	**100%**
Jacobi	481.85	**0.0007**	48.6407	48.6414	**100%**
Gauss-Seidel	no convergence				0%
Sym. Gauss-Seidel	334.00	0.0016	10.7933	10.7949	**100%**
ILU(0)	360.50	0.0023	13.6240	13.6263	**100%**
Learned	**319.50**	0.0112	**9.6079**	**9.6192**	**100%**

Table 2: Mean results for the Poisson (v1) dataset on 10 test instances.

Preconditioner	Iterations	P-time	GMRES time	total time	Success
None	no convergence				0%
Jacobi	partial convergence				10%
Gauss-Seidel	no convergence				0%
Sym. Gauss-Seidel	494.90	**0.0042**	22.2406	22.2448	**100%**
ILU(0)	partial convergence				40%
Learned	**469.50**	0.0702	**20.1612**	**20.2314**	**100%**

Table 3: Mean results for the Poisson (v2) dataset on 10 test instances.

Results for PDE data

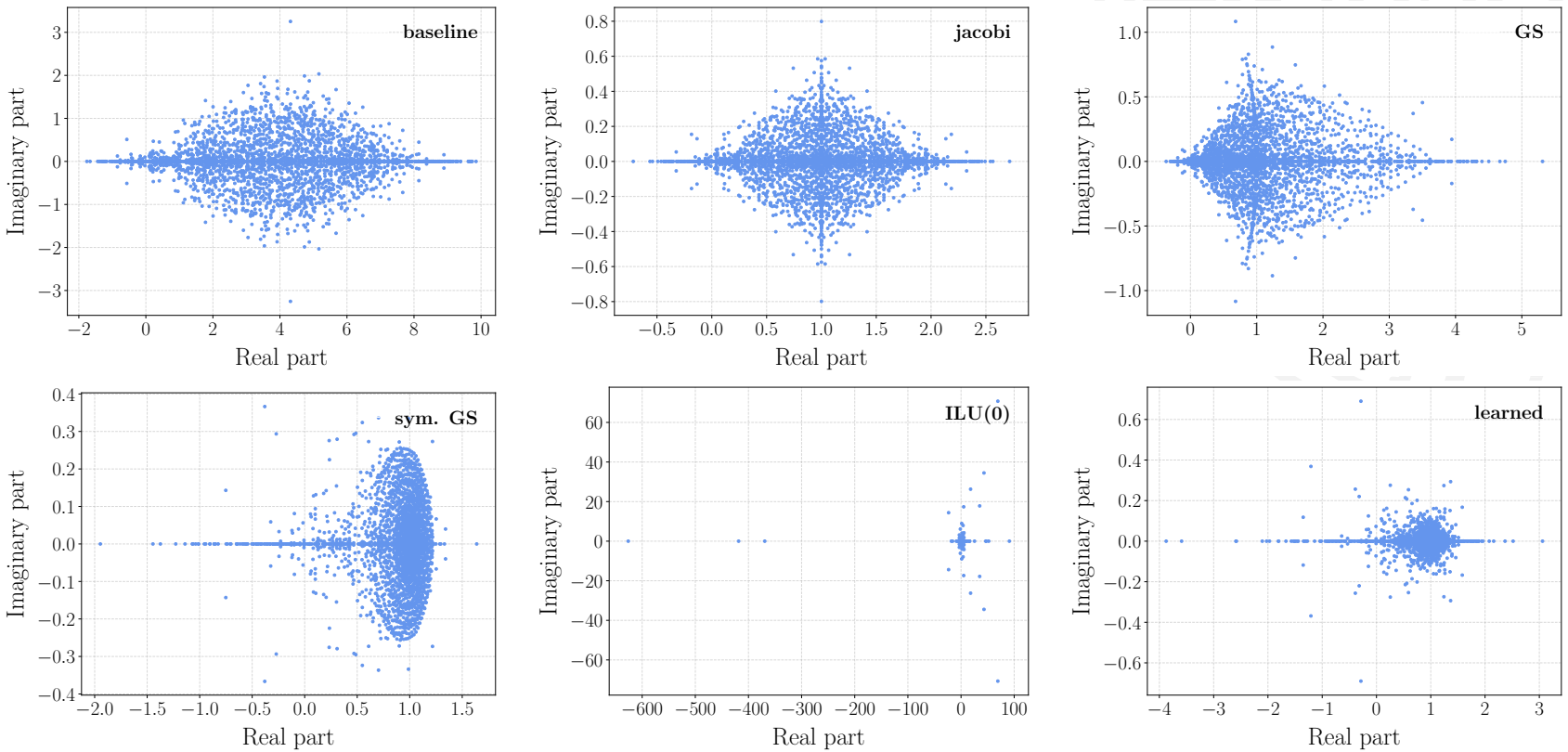


Figure 4: Eigenvalue distribution for a single instance of the Poisson (v1) dataset. All instances converged except the system preconditioned by Gauss-Seidel.

Conclusion

- In the synthetic dataset, the learned preconditioner shows promise in reducing iterations but requires further optimization to match the overall efficiency of ILU(0).
- More epochs and bigger training size reducing $\rightarrow$ loss function reduction $\rightarrow$ faster convergence $\rightarrow$ suitable loss function
- In the Poisson datasets, the learned precondition has proven to be the best in speed of convergence. Although the improvement in convergence was not substantial.
- The learned preconditioner has demonstrated superior consistency when clustering eigenvalues.

Future work

- Remove the constraint of the diagonal elements being nonzero.
- Track the GMRES performance while training.
- Apply to new problem domains such as IPMs.
- Allow additional fill-ins either heuristically or learned.