Learning the deflated conjugate gradient method using gradient-based meta-solving

Sohei Arisaka[a,b], **Qianxiao Li**[b,c], **Osamu Imazeki**[a]

[a] *Kajima Corporation, Singapore 489690* s.arisaka@kajima.com.sg, imazeki@kajima.com

[b] *Department of Mathematics, National University of Singapore, Singapore 119076* qianxiao@nus.edu.sg

[c] *Institute for Functional Intelligent Materials, National University of Singapore, Singapore 117544*

1. Introduction

Large-scale sparse matrices are ubiquitous in science and engineering, such as computational fluid dynamics and structural analysis. Solving these systems efficiently is a fundamental task in numerical linear algebra, with iterative methods being a popular choice due to their scalability and robustness. For symmetric positive-definite (SPD) systems, the Conjugate Gradient (CG) method is widely used, although it can experience slow convergence when encountering difficult spectral properties [1]. The Deflated Conjugate Gradient (D-CG) method [2] addresses this by projecting the original system onto a *deflation subspace*, typically spanned by eigenvectors associated with the smallest eigenvalues. However, since CG convergence depends on the entire spectrum and right-hand side of the linear system, the conventional choice is not always optimal [3].

In this work, we propose learning a more effective deflation subspace using a neural network combined with the gradient-based meta-solving (GBMS) framework [4]. GBMS generalizes gradient-based meta-learning [5] to the domain of numerical solvers. Using GBMS, we train a neural network to generate a deflation subspace by directly incorporating the observed convergence behavior of the D-CG method. We also investigate non-intrusive gradient-based meta-solving (NI-GBMS) [6] for practical situations where the solver does not support automatic differentiation. Our experiments show that the learned deflation subspace substantially reduces the number of iterations compared to the conventional approach.

2. Methodology

2.1 Deflated Conjugate Gradient method [2]

We consider solving a SPD linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ using the D-CG method. Deflation modifies the iterative process by introducing a *deflation subspace* $\mathcal{S} \subset \mathbb{R}^n$, spanned by the columns of $\boldsymbol{Z} \in \mathbb{R}^{n\times k}$. Using the $\boldsymbol{A}$-orthogonal projection onto $\mathcal{S}$, denoted by $\boldsymbol{\pi}_{\boldsymbol{A}}(\mathcal{S})$, the solution $\boldsymbol{x}$ can be split into

$$\boldsymbol{x} = \boldsymbol{\pi}_{\boldsymbol{A}}(\mathcal{S})\boldsymbol{x} + \big(\boldsymbol{I} - \boldsymbol{\pi}_{\boldsymbol{A}}(\mathcal{S})\big)\boldsymbol{x}. \tag{1}$$

The first term $\boldsymbol{\pi}_{\boldsymbol{A}}(\mathcal{S})\boldsymbol{x}$ is computed by solving a reduced system $(\boldsymbol{Z}^T\boldsymbol{A}\boldsymbol{Z})\,\boldsymbol{z} = \boldsymbol{Z}^T\boldsymbol{b}$, which is inexpensive since $k \ll n$. The remaining component is obtained by solving the *deflated system*:

$$\big(\boldsymbol{I} - \boldsymbol{\pi}_{\boldsymbol{A}}(\mathcal{S})\big)^T\boldsymbol{A}\,\hat{\boldsymbol{x}} = \big(\boldsymbol{I} - \boldsymbol{\pi}_{\boldsymbol{A}}(\mathcal{S})\big)^T\boldsymbol{b}. \tag{2}$$

Because deflation removes the impact of problematic modes, the modified system converges faster than the original system [2].

A standard choice of $\boldsymbol{Z}$ is the eigenvectors corresponding to the smallest eigenvalues of $\boldsymbol{A}$. It aims to reduce the effective condition number κ_{eff}, the ratio of the largest to the smallest *nonzero* eigenvalues of the deflated matrix $(\boldsymbol{I} - \boldsymbol{\pi}_{\boldsymbol{A}}(\mathcal{S}))^T\boldsymbol{A}$, appearing in the following convergence bound:

$$\|\boldsymbol{e}^{(m)}\|_{\boldsymbol{A}} \le 2\left(\frac{\sqrt{\kappa_{\text{eff}}}-1}{\sqrt{\kappa_{\text{eff}}}+1}\right)^m \|\boldsymbol{e}^{(0)}\|_{\boldsymbol{A}}, \tag{3}$$

where $\boldsymbol{e}^{(m)} = \boldsymbol{x} - \boldsymbol{x}^{(m)}$ is the error at the m-th D-CG iteration. However, there are two overlooked points. First, removing the smallest eigenvalues may not be the best choice for reducing κ_{eff}. Second, although the condition number gives the upper bound of the error, it does not fully explain the convergence behavior of the CG method. The convergence depends on not only κ_{eff} but also the spectrum of the matrix and the right-hand side of the linear system [3], and in practice, it is much faster than the bound. Thus, there can be a choice of the deflation basis better than the conventional one.

2.2 Gradient-based meta-solving [4]

We use the GBMS framework to learn the deflation basis $\boldsymbol{Z}$. Let τ be the task of solving a linear system, $\Phi(\tau;\boldsymbol{Z})$ denote the D-CG method that solves task τ using the deflation basis $\boldsymbol{Z}$, and $\Psi(\tau;\boldsymbol{\omega})$ be a neural network, called *meta-solver*, with weight $\boldsymbol{\omega}$ that produces $\boldsymbol{Z}_\tau$ for each task τ. We train Ψ to minimize

$$\mathbb{E}_{\tau\sim P}\Big[\mathcal{L}\big(\Phi(\tau;\Psi(\tau;\boldsymbol{\omega}))\big)\Big], \tag{4}$$

where P is a task distribution and $\mathcal{L}$ measures the D-CG performance. A key advantage of GBMS is that it can incorporate the actual behavior of Φ, which is missing in the solver-independent training [7]. Rather than minimizing just the residual norm or error, we target the number of iterations needed to reach a given tolerance δ, denoted by $\mathcal{L}_\delta$. Since $\mathcal{L}_\delta$ is discrete, we use the differentiable surrogate loss $\tilde{\mathcal{L}}_\delta$:

$$\tilde{\mathcal{L}}_\delta \;=\; \sum_{m=0}^{\mathcal{L}_\delta} \operatorname{sigmoid}\Big(\|\boldsymbol{r}^{(m)}\|/\|\boldsymbol{b}\| - \delta\Big), \tag{5}$$

where $\boldsymbol{r}^{(m)}$ is the residual at the m-th D-CG iteration. Because $\tilde{\mathcal{L}}_\delta$ is differentiable, we can train Ψ via backpropagation, provided the solver Φ is compatible with automatic differentiation.

2.3 Non-intrusive gradient-based meta-solving [6]

We also consider the scenario where Φ is not automatic-differentiable (e.g., a legacy solver in C++ or Fortran). In this setting, we employ the NI-GBMS algorithm [6], in which $\nabla\Phi$ is approximated using the control variate forward gradient:

$$\boldsymbol{h}_{\mathbf{v}} = (\nabla\Phi \cdot \mathbf{v})\mathbf{v} - (\nabla\hat{\Phi} \cdot \mathbf{v})\mathbf{v} + \nabla\hat{\Phi} \tag{6}$$

where $\mathbf{v}$ is a random Rademacher vector and $\hat{\Phi}$ is a neural network imitating Φ. The first term is known as the forward gradient [8], which is an unbiased estimator of $\nabla\Phi$ and can be approximated by the finite difference even when Φ is not automatic-differentiable. The control variate forward gradient $\boldsymbol{h}_{\mathbf{v}}$ reduces the variance of the forward gradient using $(\nabla\hat{\Phi} \cdot \mathbf{v})\mathbf{v}$ as a control variate, while preserving unbiasedness. During training of the meta-solver Ψ, we simultaneously train the surrogate model $\hat{\Phi}$ to match the forward gradients of the actual solver Φ.

3. Numerical experiment

3.1 Problem setting

We demonstrate the proposed method on a dam break problem simulated by the Moving Particle Semi-implicit (MPS) method [9]. MPS is a meshless technique for incompressible flow simulations and requires solving a pressure Poisson equation at each time step. It is represented by SPD system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ encodes particle interactions and boundary conditions, $\boldsymbol{x} \in \mathbb{R}^n$ represents the pressure, and $\boldsymbol{b} \in \mathbb{R}^n$ is the source term from deviations in particle number density. Here, $\boldsymbol{A}$ and $\boldsymbol{b}$ change at every time step in response to fluid motion.

To accelerate solving the pressure Poisson equation, we apply the proposed method with a graph neural network (GNN) as our meta-solver Ψ. We treat $\boldsymbol{A}$ as the graph adjacency matrix and $\boldsymbol{b}$ as the node feature. We adopt the GNN architecture from [10], originally developed for predicting solutions of linear systems, but modify its output dimension from $n \times 1$ to $n \times k$ to produce a deflation basis $\boldsymbol{Z}$. For the surrogate model $\hat{\Phi}$ in the NI-GBMS approach, we employ the same GNN architecture but include $\boldsymbol{Z}$ as additional node features. We simulate the dam break problem for 1 second with 2,500 time steps, each producing one linear system of size $n = 992$. Although the size of the linear system is relatively small, it has varying spectral properties due to the evolving particle configuration. These tasks are split into training, validation, and test sets with a ratio of 8:1:1. The meta-solver Ψ is trained on the training set using the GBMS or NI-GBMS algorithm, and its performance is evaluated on the test set.

3.2 Results

Table 1 compares three meta-solvers: $\Psi_{\mathbf{0}}$, Ψ_{eig}, and Ψ_{nn}. $\Psi_{\mathbf{0}}$ always outputs the zero matrix, corresponding to the standard CG method without deflation. Ψ_{eig} is a conventional non-learning baseline that selects eigenvectors corresponding to the smallest eigenvalues of $\boldsymbol{A}$. Ψ_{nn} is our proposed GNN-based meta-solver, trained using GBMS (where Φ is differentiable) or NI-GBMS (where Φ is not).

The proposed Ψ_{nn} significantly reduces iteration counts compared to both the standard CG $\Psi_{\mathbf{0}}$ and the eigenvector-based baseline Ψ_{eig}. In the GBMS case, for a tolerance of $\delta = 10^{-4}$, Ψ_{nn} with $k = 3$ achieves a 6.3× reduction in iteration count relative to Ψ_{eig}. For $\delta = 10^{-6}$, the reduction is about 1.9×. These results confirm that relying solely on the smallest eigenvalues may be suboptimal, and a learned deflation basis can reduce the number of iterations significantly. In the NI-GBMS case, Ψ_{nn} with $k = 2$ achieves a 2.7× reduction in iterations for $\delta = 10^{-4}$ and 1.6× for $\delta = 10^{-6}$. Although the reduction is smaller than the GBMS case due to the approximated gradient, the NI-GBMS approach still outperforms the eigenvector-based deflation by a large margin. The results demonstrate the effectiveness of learning the deflation subspace in both automatic-differentiable and non-automatic-differentiable settings.

Table 1: Number of iterations required by the D-CG method to reach tolerance δ.

(a) Iterations to reach $\delta = 10^{-4}$.

Ψ	Training	$k=1$	$k=2$	$k=3$
$\Psi_{\mathbf{0}}$	–	14.38	14.38	14.38
Ψ_{eig}	–	13.28	12.62	11.94
Ψ_{nn}	GBMS	3.44	3.01	1.91
	NI-GBMS	4.66	4.59	6.61

(b) Iterations to reach $\delta = 10^{-6}$.

Ψ	Training	$k=1$	$k=2$	$k=3$
$\Psi_{\mathbf{0}}$	–	21.56	21.56	21.56
Ψ_{eig}	–	20.13	19.08	18.08
Ψ_{nn}	GBMS	10.87	10.45	9.60
	NI-GBMS	12.15	11.96	13.67

4. Conclusion

We present a novel approach for learning the deflation subspace of the D-CG method via gradient-based meta-solving. By training a graph neural network using the GBMS framework, we capture both system-specific and convergence-related characteristics, yielding substantially faster convergence compared to the conventional choice of deflating with eigenvectors of the smallest eigenvalues. Our numerical experiments demonstrate the effectiveness of this technique in both automatic-differentiable and non-automatic-differentiable settings. Future work includes more detailed performance analysis considering wall-clock time, designing more advanced network architectures, and extending the approach to larger-scale industrial simulations.

Acknowledgments

S. Arisaka and O. Imazeki are supported by Kajima Corporation, Japan. Q. Li is supported by the National Research Foundation, Singapore, under the NRF fellowship (NRF-NRFF13-2021-0005).

References

[1] Jörg Liesen and Zdenek Strakos. *Krylov Subspace Methods: Principles and Analysis*. Oxford University Press, 2012.

[2] Y Saad, M Yeung, J Erhel, and F Guyomarc'h. A Deflated Version of the Conjugate Gradient Algorithm. *SIAM Journal on Scientific Computing*, 21(5):1909–1926, 2000.

[3] Erin Carson, Jörg Liesen, and Zdeněk Strakoš. Towards understanding CG and GMRES through examples. *Linear algebra and its applications*, 692:241–291, 1 July 2024.

[4] Sohei Arisaka and Qianxiao Li. Principled Acceleration of Iterative Numerical Methods Using Machine Learning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 1041–1059. PMLR, 2023.

[5] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 2017.

[6] Sohei Arisaka and Qianxiao Li. Accelerating Legacy Numerical Solvers by Non-intrusive Gradient-based Meta-solving. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 1689–1708. PMLR, 2024.

[7] Jian Luo, Jie Wang, Hong Wang, Huanshuo Dong, Zijie Geng, Hanzhu Chen, and Yufei Kuang. Neural Krylov Iteration for Accelerating Linear System Solving. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 6 November 2024.

[8] Atılım Güneş Baydin, Barak A Pearlmutter, Don Syme, Frank Wood, and Philip Torr. Gradients without Backpropagation. *arXiv [cs.LG]*, 17 February 2022.

[9] Gen Li, Guangtao Duan, Xiaoxing Liu, and Zidi Wang. Moving particle semi-implicit method, 1 January 2023.

[10] Jie Chen. Graph neural preconditioners for iterative solutions of sparse linear systems. *arXiv [math.NA]*, 2 June 2024.

Appendix A. Algorithms

Algorithm 1: GBMS

Input: $(P, \mathcal{T})$: task space, Ψ: meta-solver, Φ: differentiable solver, S: stopping criterion, α: learning rate
Result: ω: optimized meta-solver parameter

while *S is not satisfied* **do**
- $\tau \sim P$; // sample task τ from P
- $\theta_\tau \leftarrow \Psi(\tau; \omega)$; // generate θ_τ by Ψ with ω
- $\hat{u} \leftarrow \Phi(\tau; \theta_\tau)$; // solve task τ using Φ with θ_τ
- $\omega \leftarrow \omega - \alpha \nabla_\omega L_\tau(\hat{u})$; // update ω using gradient descent

Algorithm 2: NI-GBMS

Input: $(P, \mathcal{T})$: task space, Ψ: meta-solver with weight $\boldsymbol{\omega}$, f: legacy solver, $\hat{f}$: surrogate model with weight $\boldsymbol{\phi}$, $\mathcal{L}$: loss function, Opt: optimizer for Ψ, $\hat{\mathrm{Opt}}$: optimizer for $\hat{f}$, $\hat{P}$: distribution of $\mathbf{v}$, ϵ: positive scalar, S: stopping criterion
Result: ω: optimized meta-solver parameter

while *S is not satisfied* **do**
- $\tau \sim P$, $\mathbf{v} \sim \hat{P}$; // Sample task τ and random vector $\mathbf{v}$ from P and $\hat{P}$
- /* **Forward computation** */
- $\boldsymbol{\theta} \leftarrow \Psi(\tau; \boldsymbol{\omega})$; // Generate solver parameter $\boldsymbol{\theta}$ by Ψ
- $y \leftarrow f(\boldsymbol{\theta})$, $y^+ \leftarrow f(\boldsymbol{\theta} + \epsilon \mathbf{v})$; // Solve task τ twice using $\boldsymbol{\theta}$ and $\boldsymbol{\theta} + \epsilon \mathbf{v}$
- $d \leftarrow \frac{y^+ - y}{\epsilon}$; // Compute the directional derivative of f
- $\hat{y}, \hat{d} \leftarrow \mathrm{ForwardAD}(\hat{f}, \boldsymbol{\theta}, \mathbf{v})$; // Compute output and directional derivative of $\hat{f}$ using forward mode AD
- $L \leftarrow \mathcal{L}(y)$, $\hat{L} \leftarrow (d - \hat{d})^2$; // Compute main loss L and surrogate model loss $\hat{L}$
- /* **Backward computation** */
- $\boldsymbol{h} \leftarrow d\mathbf{v} - \hat{d}\mathbf{v} + \nabla_{\boldsymbol{\theta}} \hat{f}(\boldsymbol{\theta})$; // Compute control variate forward gradient
- $\nabla_{\boldsymbol{\omega}} L \leftarrow \mathrm{ModifiedBackprop}(L, \boldsymbol{\omega}, \boldsymbol{h})$; // Compute $\nabla_{\boldsymbol{\omega}} L$ with modified backpropagation
- $\nabla_{\boldsymbol{\phi}} \hat{L} \leftarrow \mathrm{Backprop}(\hat{L}, \boldsymbol{\phi})$; // Compute $\nabla_{\boldsymbol{\phi}} \hat{L}$ by backpropagation
- /* **Update** */
- $\boldsymbol{\omega} \leftarrow \mathrm{Opt}(\boldsymbol{\omega}, \nabla_{\boldsymbol{\omega}} L)$, $\boldsymbol{\phi} \leftarrow \hat{\mathrm{Opt}}(\boldsymbol{\phi}, \nabla_{\boldsymbol{\phi}} \hat{L})$; // Update $\boldsymbol{\omega}$ and $\boldsymbol{\phi}$ using optimizers