Least-squares space-time formulation for advection-diffusion problem with efficient linear solver based on matrix compression

Marcin Łoś[(1)],
(space-time formulation, implementation)



Paulina Sepulveda-Salaz[(2)], Mateusz Dobija, Anna Paszyńska[(3)],
(space-time formulation) (SVD matrix compression)

Maciej Paszyński[(1)] (project management)

[(1)] AGH University of Science and Technology, Kraków, Poland
[(2)] Pontifical Catholic University of Valparaiso, Chile
[(3)] Jagiellonian University, Kraków, Poland

Introduction

- We consider advection-dominated diffusion transient problem formulated in space-time finite element method
- We employ B-splines for discretizations
- We propose a stabilization method based on least squares
- We generate and compress the resulting matrix using SVD algorithm applied for blocks of the global matrix (referred to as H-matrices)
- We show that the SVD compressed matrices can be processed by an iterative solver one order of magnitude faster than regular matrices

Space-time formulations

We start with a simple advection-diffusion problem:

$$\partial_t \phi = \varepsilon \Delta \phi - \beta \cdot \nabla u + f$$
$$u(x, 0) = u_0 \quad \text{for } x \in \Omega$$
$$u(x, t) = 0 \quad \text{for } (x, t) \in \partial\Omega \times [0, T]$$

For small $\varepsilon / \|\beta\|$ we may encounter instability (advection dominated)

Simple space-time formulation

- [Schafelner, Vassilevski, 2020] Numerical results for adaptive (negative norm) constrained first order system least squares formulations.
- [Voronin, Lee, Neumüller, Sepulveda, Vassilevski, 2018] Space-time discretizations using constrained first-order system least squares (CFOSLS)
- [Neumüller, 2013] ST Methods: Fast Solvers and Applications, thesis.
- [O. Steinbach, Yang. 2017] Comparison of algebraic multigrid methods for an adaptive space-time finite-element discretization of the heat equation in 3D and 4D.
- [Demkowicz, Nagaraj, Gopalakrishnan, Sepulveda 2017] A ST DPG method for the Schrödinger equation.
- [Gopalakrishnan, Sepulveda, 2019] A ST DPG method for acoustic waves in multiple dimensions
- [Wieners, Ernesti, 2019] A ST DPG method for acoustic waves
- [Führer, Karkulik, 2019] ST least-squares finite elements for parabolic equations

Simple space-time formulation

We can rewrite the equation as

$$\partial_t \phi + \mathrm{div}_x \underbrace{(-\varepsilon \nabla \phi + \beta \phi)}_{\mathcal{L}\phi} = f$$

or

$$\mathrm{div}_{x,t}\, \underline{\boldsymbol{\sigma}} = f$$

where $\underline{\boldsymbol{\sigma}} = (\mathcal{L}\phi, \phi)$ and $\mathcal{L}\phi = (\mathcal{L}_x \phi, \mathcal{L}_y \phi)$

Space-time formulation: find $\underline{\boldsymbol{\sigma}} \in H(\mathrm{div}, \Omega \times [0, T])$, $\phi \in V$

$$\begin{cases} \underline{\boldsymbol{\sigma}} - \begin{bmatrix} \mathcal{L}\phi \\ \phi \end{bmatrix} = 0 \\ \mathrm{div}_{x,t}\, \underline{\boldsymbol{\sigma}} = f \end{cases}$$

Simple space-time formulation

To simplify a bit, we can remove one unknown, as $\underline{\boldsymbol{\sigma}} = (\boldsymbol{\sigma}, \phi)$

$$\begin{cases} \partial_t \phi + \mathrm{div}_x \, \boldsymbol{\sigma} = f \\ \boldsymbol{\sigma} - \mathcal{L}\phi = 0 \end{cases}$$

We multiply the first equation by test function $\psi \in L^2(\Omega)$

$$(\partial_t \phi, \psi) + (\mathrm{div}_x \, \boldsymbol{\sigma}, \psi) = (f, \psi)$$

and the other by $\boldsymbol{\tau} = (\tau_x, \tau_y) \in L^2(\Omega) \times L^2(\Omega)$

$$(\boldsymbol{\sigma}, \boldsymbol{\tau}) - (\mathcal{L}\phi, \boldsymbol{\tau}) = 0$$

leaving us with the problem: find $\underline{\boldsymbol{\sigma}} \in H(\mathrm{div}, \Omega \times [0, T])$, $\phi \in V$

$$\begin{cases} (\partial_t \phi, \psi) + (\partial_x \sigma_x, \psi) + (\partial_y \sigma_y, \psi) = (f, \psi) \\ (\sigma_x, \tau_x) - (\mathcal{L}_x \phi, \tau_x) = 0 \\ (\sigma_y, \tau_y) - (\mathcal{L}_y \phi, \tau_y) = 0 \end{cases}$$

Simple space-time formulation – discrete level

We discretize the following system

$$\begin{cases} (\partial_t \phi, \psi) + (\partial_x \sigma_x, \psi) + (\partial_y \sigma_y, \psi) = (f, \psi) \\ (\sigma_x, \tau_x) - (\mathcal{L}_x \phi, \tau_x) = 0 \\ (\sigma_y, \tau_y) - (\mathcal{L}_y \phi, \tau_y) = 0 \end{cases}$$

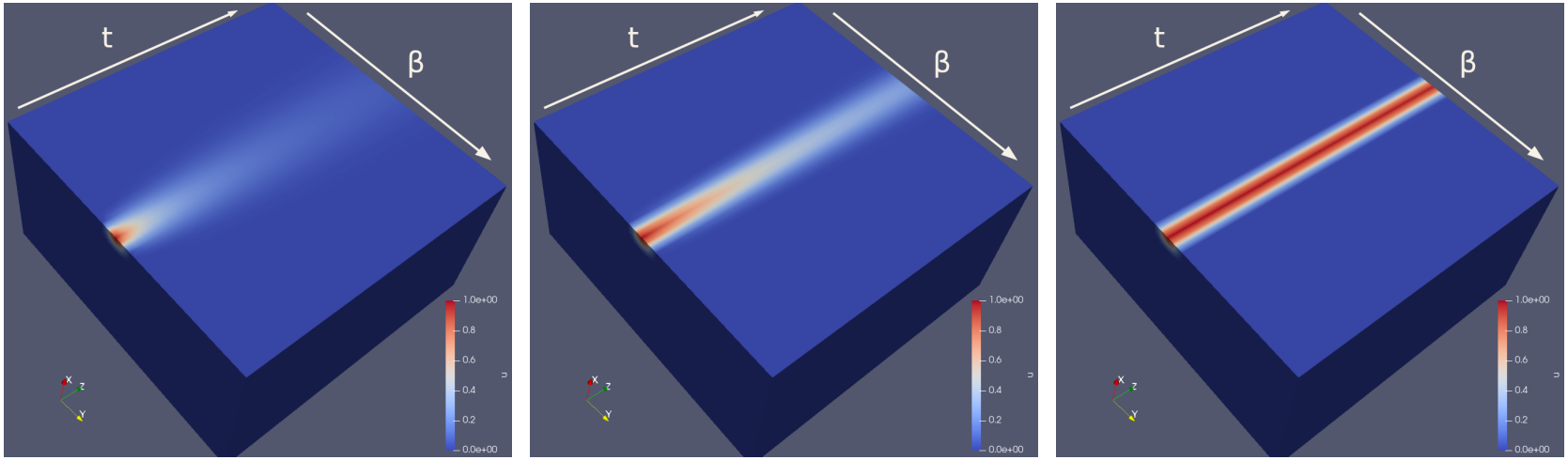
using quadratic B-splines and get the following structure of the discrete system

$$\begin{bmatrix} A_t & A_x & A_y \\ L_x & M & 0 \\ L_y & 0 & M \end{bmatrix} \begin{bmatrix} \phi \\ \sigma_x \\ \sigma_y \end{bmatrix} = \begin{bmatrix} f \\ 0 \\ 0 \end{bmatrix}$$

where

- $M \sim u, v \mapsto (u, v)$ (mass matrix)
- $A_\gamma \sim u, v \mapsto (\partial_\gamma u, v)$
- $L_\gamma \sim u, v \mapsto (\mathcal{L}_\gamma u, v)$

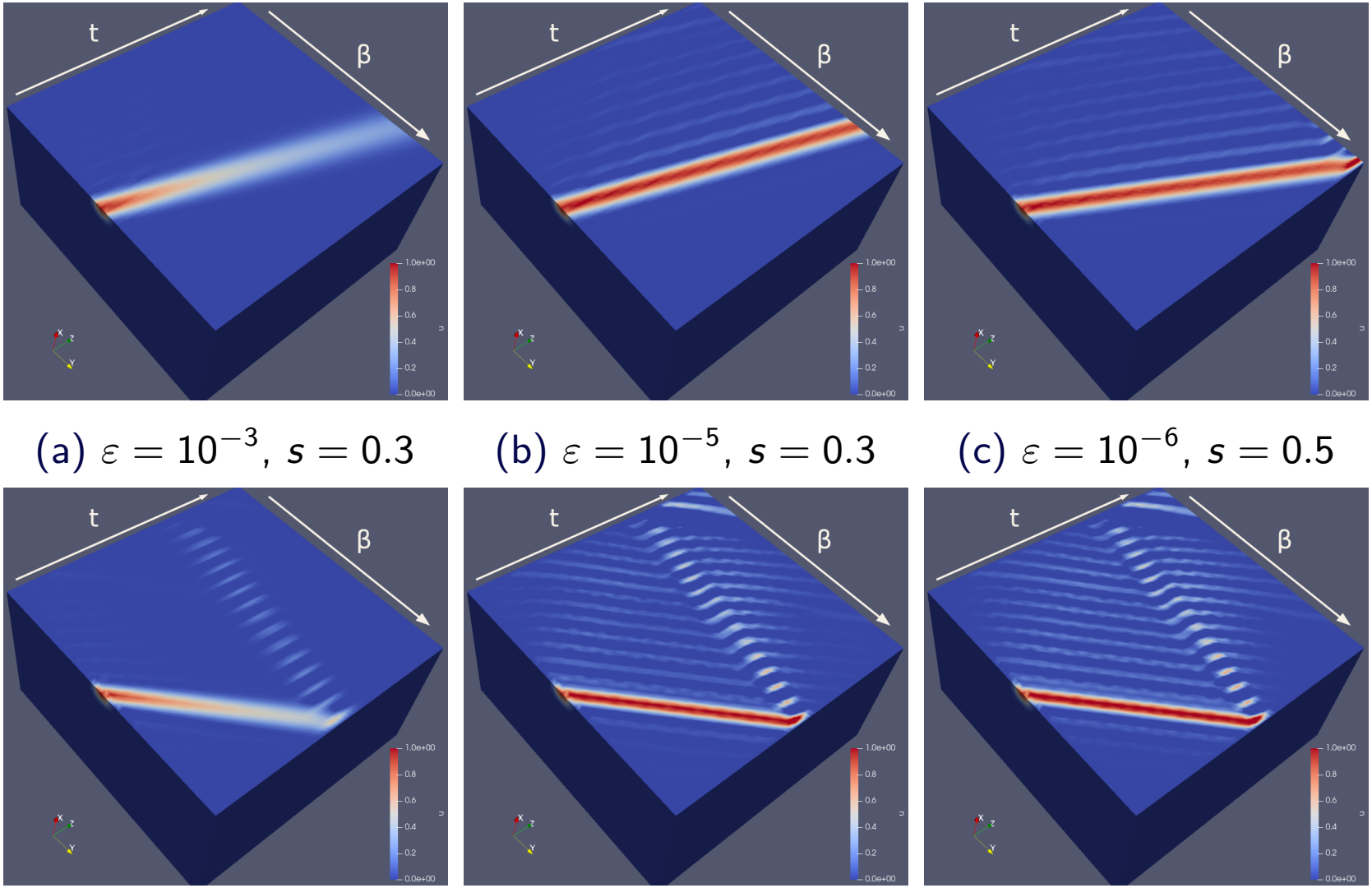
Example results – pure diffusion



(a) $\varepsilon = 5 \times 10^{-3}$ (b) $\varepsilon = 10^{-3}$ (c) $\varepsilon = 10^{-5}$

Figure: Results for $\beta = 0$, $T = 1$, $32 \times 32 \times 32$ mesh with quadratic B-splines

Example results – diffusion and advection



(a) $\varepsilon = 10^{-3}$, $s = 0.3$ (b) $\varepsilon = 10^{-5}$, $s = 0.3$ (c) $\varepsilon = 10^{-6}$, $s = 0.5$

(d) $\varepsilon = 10^{-3}$, $s = 1$ (e) $\varepsilon = 10^{-5}$, $s = 1$ (f) $\varepsilon = 10^{-6}$, $s = 1$

Figure: Results for $\beta = (0, s)$, $T = 1$, $32 \times 32 \times 32$ mesh with quadratic B-splines

Mixed formulation

Idea: constrained least-squares problem

$$\textbf{min } J(\underline{\boldsymbol{\sigma}}, \phi) = \frac{1}{2} \left\| \underline{\boldsymbol{\sigma}} - \begin{bmatrix} \mathcal{L}\phi \\ \phi \end{bmatrix} \right\|^2 = \frac{1}{2} \left\| \boldsymbol{\sigma} - \mathcal{L}\phi \right\|^2 + \frac{1}{2} \left\| \sigma_* - \phi \right\|^2$$

$$\textbf{subject to } \operatorname{div}_{x,t} \underline{\boldsymbol{\sigma}} = f$$

where $\underline{\boldsymbol{\sigma}} \in H(\operatorname{div}, \Omega \times [0, T])$, $\phi \in V$ and $\underline{\boldsymbol{\sigma}} = (\boldsymbol{\sigma}, \sigma_*)$

Lagrange multipliers:

$$G(\underline{\boldsymbol{\sigma}}, \phi, \lambda) = \frac{1}{2} \left\| \boldsymbol{\sigma} - \mathcal{L}\phi \right\|^2 + \frac{1}{2} \left\| \sigma_* - \phi \right\|^2 + (\operatorname{div}_{x,t} \underline{\boldsymbol{\sigma}} - f, \lambda)$$

Mixed formulation

Problem: find $(\underline{\boldsymbol{\sigma}}, \phi, \lambda) \in \mathbf{W}$ such that for all $(\underline{\boldsymbol{\tau}}, \omega, \mu) \in \mathbf{W}$

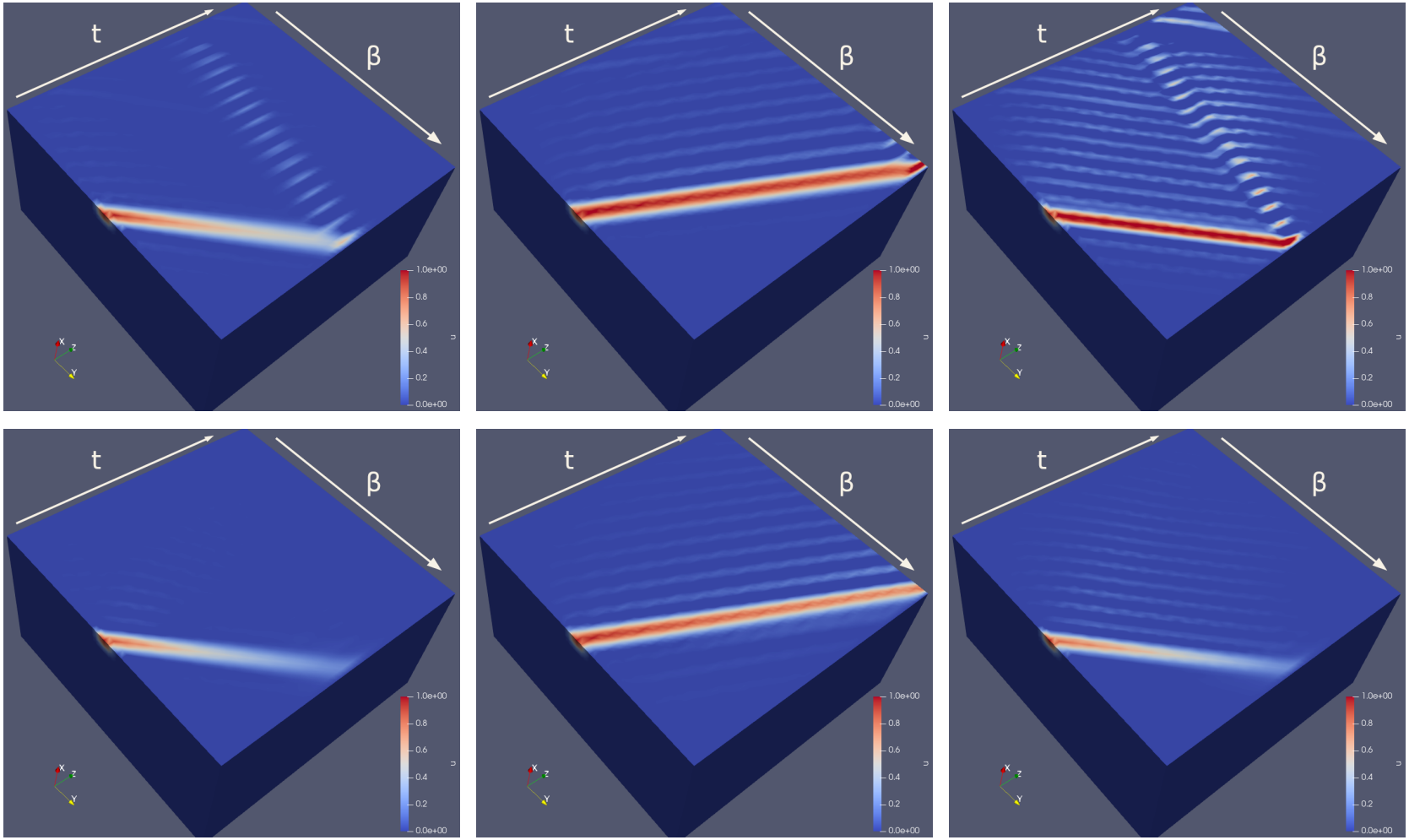
$$
\begin{aligned}
&(\boldsymbol{\sigma} - \mathcal{L}\phi, \boldsymbol{\tau} - \mathcal{L}\omega) + (\sigma_* - \phi, \tau_* - \omega) \\
&+ (\mathrm{div}_{\mathrm{x,t}}\, \underline{\boldsymbol{\tau}}, \lambda) + (\mathrm{div}_{\mathrm{x,t}}\, \underline{\boldsymbol{\sigma}} - f, \mu) = 0
\end{aligned}
$$

where $\mathbf{W} = H(\mathrm{div}, \Omega \times [0, T]) \times V \times L^2(\Omega \times [0, T])$

Equivalently:

$$
\begin{array}{llll}
(\phi, \omega) + (\mathcal{L}\phi, \mathcal{L}\omega) - (\sigma_*, \omega) & - (\boldsymbol{\sigma}, \mathcal{L}\omega) & & = 0 \\
- (\phi, \tau_*) & + (\sigma_*, \tau_*) & + (\lambda, \partial_t \tau_*) & = 0 \\
- (\mathcal{L}\phi, \boldsymbol{\tau}) & + (\boldsymbol{\sigma}, \boldsymbol{\tau}) & + (\lambda, \mathrm{div}_{\mathrm{x}}\, \boldsymbol{\tau}) & = 0 \\
& + (\partial_t \sigma_*, \mu) + (\mathrm{div}_{\mathrm{x}}\, \boldsymbol{\sigma}, \mu) & & = (f, \mu)
\end{array}
$$

Example results with stabilization



(a) $\varepsilon = 10^{-3}$, $s = 1$ (b) $\varepsilon = 10^{-5}$, $s = 0.5$ (c) $\varepsilon = 10^{-6}$, $s = 1$

Figure: Results for $\beta = (0, s)$, $T = 1$, $32 \times 32 \times 32$ mesh with quadratic B-splines (top: no stabilization, bottom: stabilization)

Mixed formulation matrix

Discretization leads to

$$\begin{bmatrix} M+K & -M & -L_x^T & -L_y^T & 0 \\ -M & M & 0 & 0 & A_t^T \\ -L_x & 0 & M & 0 & A_x^T \\ -L_y & 0 & 0 & M & A_y^T \\ 0 & A_t & A_x & A_y & 0 \end{bmatrix} \begin{bmatrix} \phi \\ \sigma_* \\ \sigma_x \\ \sigma_y \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ f \end{bmatrix}$$

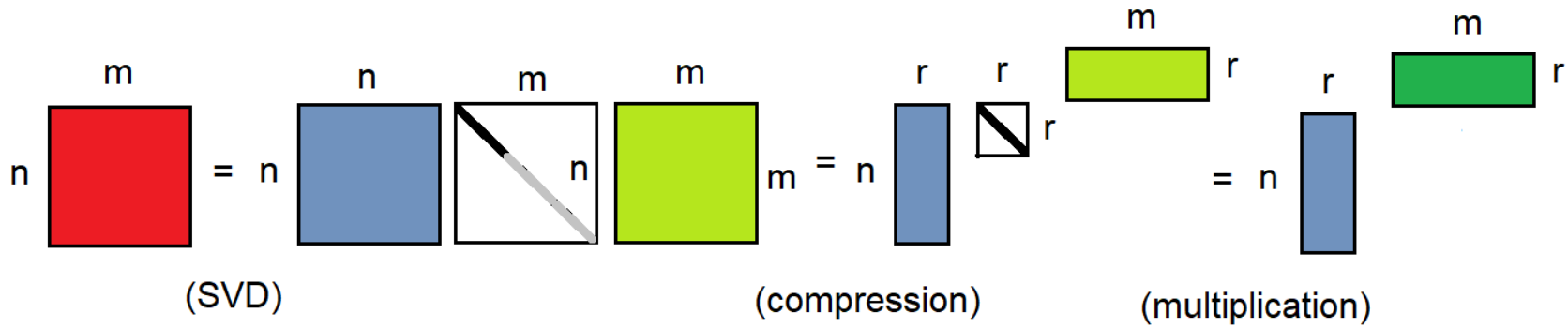
where

- $M \sim u, v \mapsto (u, v)$ (mass matrix)
- $A_\gamma \sim u, v \mapsto (\partial_\gamma u, v)$
- $L_\gamma \sim u, v \mapsto (\mathcal{L}_\gamma u, v)$
- $K \sim u, v \sim (\mathcal{L}u, \mathcal{L}v)$

Similar structure as in DPG (saddle-point system)

How to solve it efficiently?

Matrix compression with Singular Value Decomposition

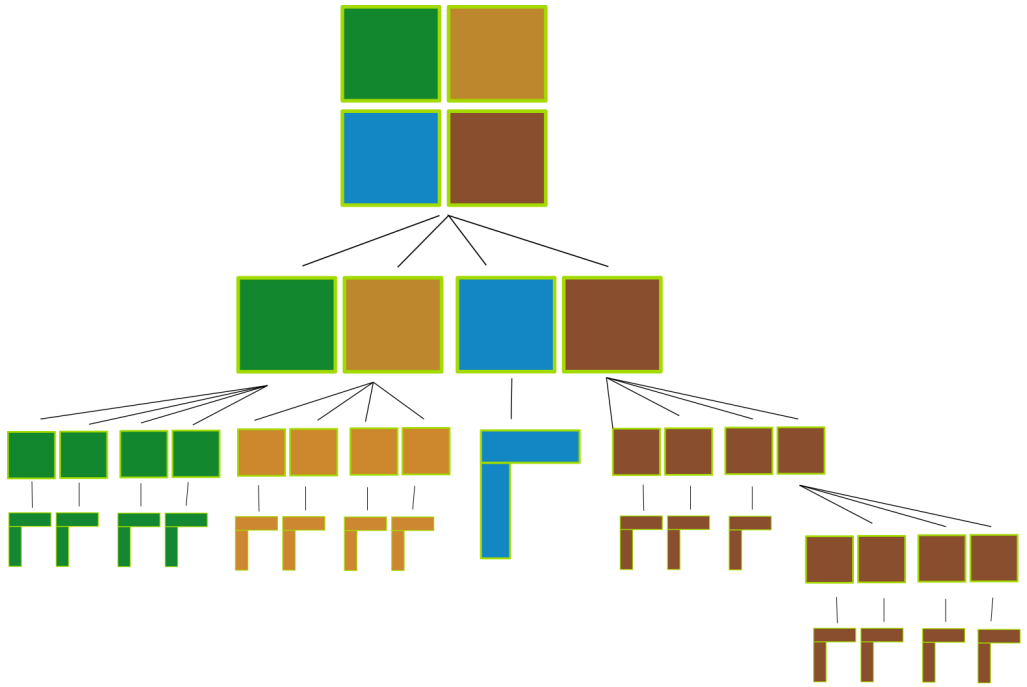


$$A = UDV, \quad [U, D, V] = SVD(B),$$
$$U \in \mathcal{M}^{n \times n}, D - \text{diagonal } m \times n, V \in \mathcal{M}^{m \times m}$$

Truncated SVD

The entries of D (singular values) are sorted in descending order.
The diagonal values less than the compression threshold δ are removed together with corresponding columns of U and rows of V.
The rank of the compressed matrix $s = \max\{i : d_{ii} > \delta\}$.
Optimal rank s approximation wrt Frobenius norm (Eckart–Young)

Hierarchical compression



In each node, the decision about storing the block in SVD form or dividing the block into submatrices depends on the so-called admissibility condition of the block

Recursive compression: *node* = **compress_matrix**(A, δ, b)

Require: $A \in \mathcal{M}^{m \times n}$, δ *compression threshold*, b *maximum rank*

1: $v \leftarrow create\_tree()$
2: $A_{11} \leftarrow A(1 : \frac{m}{2}, 1 : \frac{n}{2})$
3: $A_{12} \leftarrow A(1 : \frac{m}{2}, \frac{n}{2} + 1 : n)$
4: $A_{21} \leftarrow A(\frac{m}{2} + 1 : m, 1 : \frac{n}{2})$
5: $A_{22} \leftarrow A(\frac{m}{2} + 1 : m, \frac{n}{2} : n)$
6: $n_1 \leftarrow \text{process\_matrix}(A_{11}, \delta, b)$
7: $n_2 \leftarrow \text{process\_matrix}(A_{12}, \delta, b)$
8: $n_3 \leftarrow \text{process\_matrix}(A_{21}, \delta, b)$
9: $n_4 \leftarrow \text{process\_matrix}(A_{22}, \delta, b)$
10: $v.sons \leftarrow [n_1, n_2, n_3, n_4]$

Processing of a block: $node = \textbf{process\_matrix}(A, \delta, b)$

Require: $A \in \mathcal{M}^{m \times n}$, δ *compression threshold*, b *maximum rank*

1: **if** $A = 0$ **then**
2: **create new node** v; $v.rank \leftarrow 0$; $v.size \leftarrow size(A)$;
return v;
3: **end if**
4: $[U, D, V] \leftarrow SVD(A)$; $\sigma \leftarrow diag(D)$;
5: $rank \leftarrow card\left(\{i : \sigma_i > \delta\}\right)$;
6: **if** $rank < b$ **then**
7: **create new node** v; $v.rank \leftarrow rank$;
8: $v.singularvalues \leftarrow \sigma(1 : rank)$;
9: $v.U \leftarrow U(*, 1 : rank)$;
10: $v.V \leftarrow D(1 : rank, 1 : rank) * V(1 : rank, *)$;
11: $v.sons \leftarrow \emptyset$; $v.size \leftarrow size(A)$;
12: **return** v;
13: **else**
14: **return** $compress\_matrix(A, \delta, b)$;
15: **end if**

Practical considerations

SVD is expensive: standard LAPACK subroutine dgesvd has time complexity $\mathcal{O}(N^3)$ for a square matrix
(C. F. Van Loan G. H. Golub. Matrix Computations. Johns Hopkins University Press, London, 1996.)

We avoid computing SVD and use other splitting criteria

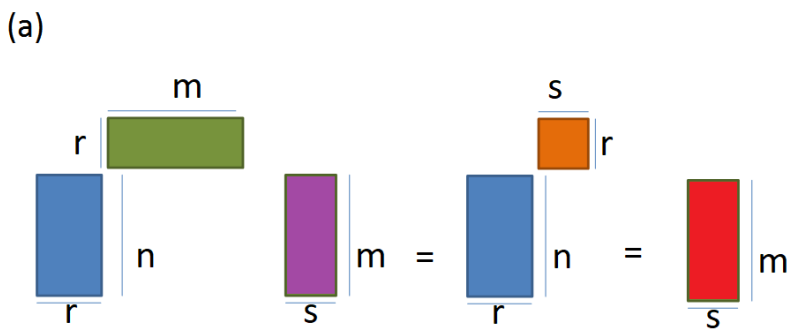
- maximum size
- tree depth
- density of nonzero entries
- ...

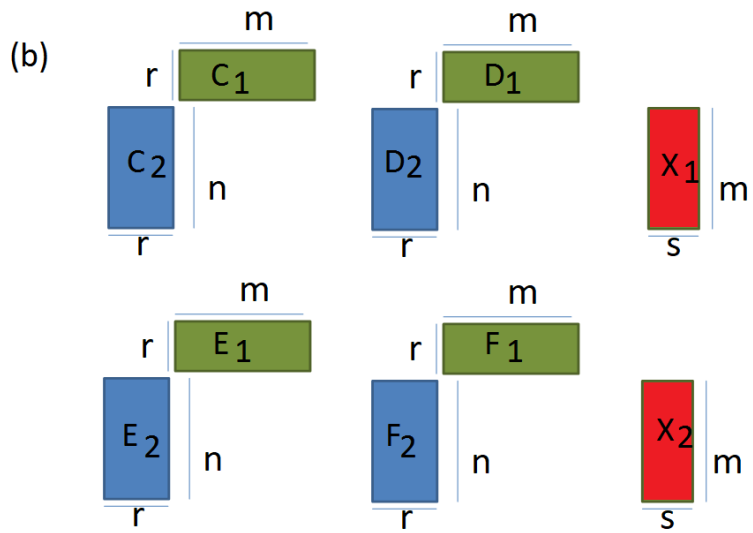
If matrix comes from FEM discretization, we have a lot of information about its structure

Matrix vector multiplication

How to multiply a vector by a compressed matrix?

- **leaf case** (left): computational cost of matrix-vector multiplication with compressed matrix and s vectors is $\mathcal{O}(rms + rns)$, when $n = m = N \gg r$ it reduces to $\mathcal{O}(Nrs)$
- **recursive step** (right): multiplication of a matrix compressed into four SVD blocks by the vector partitioned into two blocks, following $\begin{bmatrix} C_2 * (C_1 * X_1) + D_2 * (D_1 * X_2) \\ E_2 * (E_1 * X_1) + F_2 * (F_1 * X_2) \end{bmatrix}$. The computational cost is $\mathcal{O}(Nrs)$.





Matrix vector multiplication: $Y =$ **matrix_vector_mult**(v, X)

Require: node v, X *vectors to multiply*

1: **if** $v.sons = \emptyset$ **then**
2: **if** $v.rank = 0$ **then**
3: **return** $zeros(size(A).rows)$;
4: **end if**
5: **return** $v.U * (v.V * X)$;
6: **end if**
7: $rows \leftarrow size(X).rows$;
8: $X_1 \leftarrow X(1 : \frac{rows}{2}, *)$; $X_2 \leftarrow X(\frac{rows}{2} + 1 : rows, *)$;
9: $Y_1^{(1)} \leftarrow \text{matrix\_vector\_mult}(v.sons(1), X_1)$;
10: $Y_1^{(2)} \leftarrow \text{matrix\_vector\_mult}(v.sons(2), X_2)$;
11: $Y_2^{(1)} \leftarrow \text{matrix\_vector\_mult}(v.sons(3), X_1)$;
12: $Y_2^{(2)} \leftarrow \text{matrix\_vector\_mult}(v.sons(4), X_2)$;
13: **return** $\begin{bmatrix} Y_1^{(1)} + Y_1^{(2)} \\ Y_2^{(1)} + Y_2^{(2)} \end{bmatrix}$;

Matrix-vector multiplication complexity

Crucial assumption: most of the compression tree is shallow

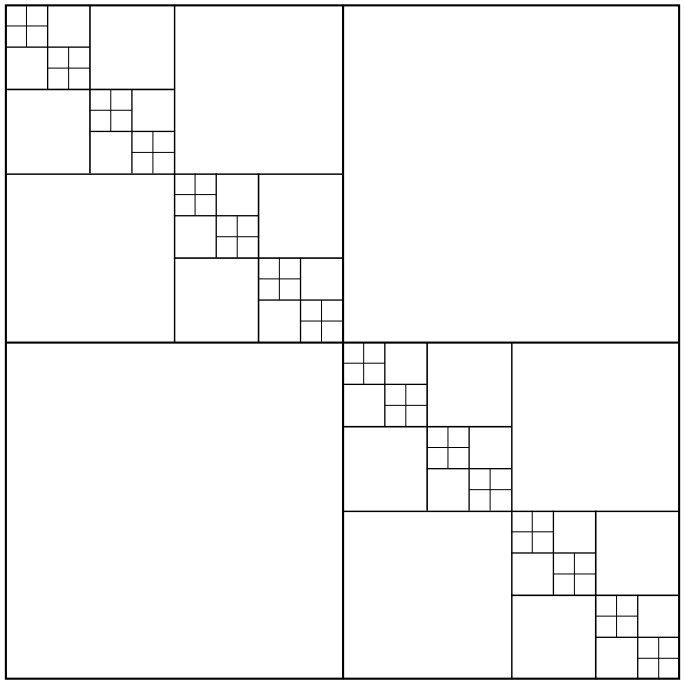
Tree built uniformly until fixed size N_0 – poor performance

$$C(N) = \underbrace{4C(N/2)}_{\text{multiplication}} + \underbrace{\mathcal{O}(N)}_{\text{addition}}, \quad C(N_0) = \mathcal{O}(N_0 rs)$$

$$\Rightarrow C(N) = \mathcal{O}(N^2)$$

But: sparsity $\Rightarrow$ large empty/simple regions $\Rightarrow$ large blocks

Matrix-vector multiplication complexity – example



At each step: 2 leaves, 2 interior nodes

$$C(N) = \underbrace{2C(N/2) + 2\mathcal{O}(Nrs/2)}_{\text{multiplication}} + \underbrace{\mathcal{O}(N)}_{\text{addition}}, \quad C(N_0) = \mathcal{O}(N_0 rs)$$

$$\Rightarrow C(N) = \mathcal{O}(N \log N)$$

Compressed matrices of space-time formulation

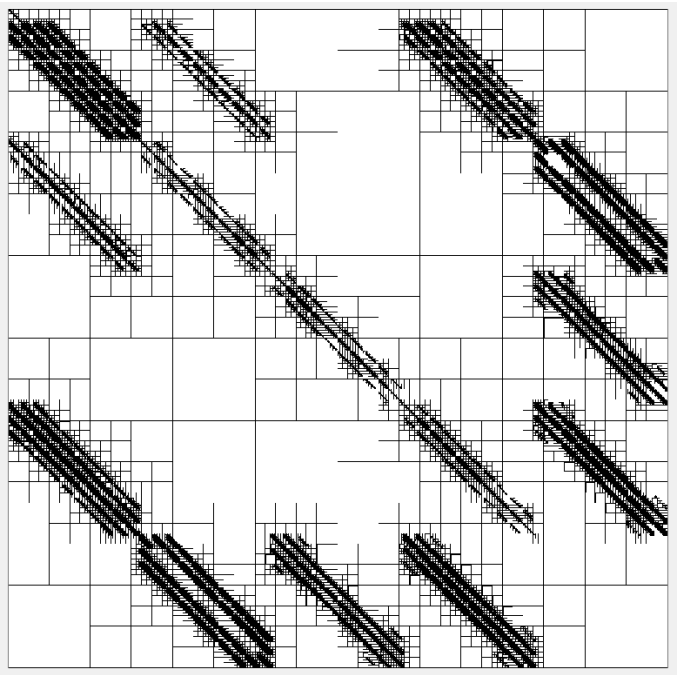


Figure: Compression of space-time $8 \times 8 \times 8$ matrix.

matrix	iterations	FLOPS	FLOPS per iteration
original	10	25,112,040	2,511,204
compressed	7	2,307,760	329,680

Conclusions

- Residual minimization methods can be successfully applied for space-time formulations
- Hierarchical compression of matrices with SVD algorithm plugged into the GMRES iterative solver can speed up the solution process one order of magnitude
- Future work
  - exploring other space-time formulations
  - investigation of compression error
  - choice of singular value threshold and max rank
  - admissibility condition