EPFL

The Sylvester Equation

Computational Linear Algebra Miniproject

Matthias Zeller

June 6, 2021

Notation

- Vectors are denoted by lowercase bold letters, e.g. $\mathbf{x}$
- Scalars are denoted by lowercase non-bold letters, e.g. x
- Matrices are denotes by capital non-bold letters, e.g. A
- Columns of a matrix $A \in \mathbb{R}^{m\times n}$ are denoted by the lowercased matrix name in roman alphabet: $A = \begin{bmatrix}\mathbf{a}_1 & \dots & \mathbf{a}_n\end{bmatrix}, \mathbf{a}_i \in \mathbb{R}^m$

1 Problem formulation

The Sylvester equation (1) consists in given coefficient matrices $A \in \mathbb{R}^{m\times m}$, $B \in \mathbb{R}^{n\times n}$, $C \in \mathbb{R}^{m\times n}$ and an unknown matrix $X \in \mathbb{R}^{m\times n}$:

$$AX - XB = C \tag{1}$$

2 Linear system reformulation

We will show that an equivalent formulation of (1) is the equation of a linear system:

$$\mathcal{A}\mathbf{x} = \mathbf{c}\,, \tag{2}$$

where $\mathcal{A} \in \mathbb{R}^{mn\times mn}$, $\mathbf{x}, \mathbf{c} \in \mathbb{R}^{mn}$.

We use the vectorization transformation vec : $\mathbb{R}^{m\times n} \to \mathbb{R}^{mn}$, which can be seen as stacking column vectors of a matrix:

$$\text{vec}(A_{m\times n}) = \text{vec}\left(\begin{bmatrix}\mathbf{a}_1 & \dots & \mathbf{a}_n\end{bmatrix}\right) = \begin{bmatrix}\mathbf{a}_1 \\ \vdots \\ \mathbf{a}_n\end{bmatrix} \in \mathbb{R}^{mn\times 1}\,,$$

and we notice that we can link the vectorization operation with a Kronecker product, using partitionning:

$$
\begin{aligned}
\operatorname{vec}(AX) &= \operatorname{vec}\left(A\begin{bmatrix}\mathbf{x}_1 & \dots & \mathbf{x}_n\end{bmatrix}\right) = \operatorname{vec}\left(\begin{bmatrix}A\mathbf{x}_1 & \dots & A\mathbf{x}_n\end{bmatrix}\right) \\
&= \begin{bmatrix}A\mathbf{x}_1 \\ \vdots \\ A\mathbf{x}_n\end{bmatrix} = \begin{bmatrix}A & \mathbb{0}_{m\times m} & \dots & \mathbb{0}_{m\times m} \\ \mathbb{0}_{m\times m} & A & \dots & \mathbb{0}_{m\times m} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{0}_{m\times m} & \mathbb{0}_{m\times m} & \dots & A\end{bmatrix}\begin{bmatrix}\mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n\end{bmatrix} \\
&= (\mathbb{I}_n \otimes A)\operatorname{vec}(X)\,, \qquad (3)
\end{aligned}
$$

where $\mathbb{0}_{m\times m}$ denotes an $m \times m$ matrix full of zeros, $\mathbb{I}_n$ denotes the $n \times n$ identity matrix used to build the block diagonal matrix, and $\otimes$ denotes the Kronecker product. Analogously, we have the relationship:

$$
\begin{aligned}
\operatorname{vec}(XB) &= \operatorname{vec}\left(\begin{bmatrix}X\mathbf{b}_1 & \dots & X\mathbf{b}_n\end{bmatrix}\right) = \operatorname{vec}\left(\begin{bmatrix}(\sum_{i=1}^n b_{i1}\mathbf{x}_i) & \dots & (\sum_{i=1}^n b_{in}\mathbf{x}_i)\end{bmatrix}\right) \\
&= \begin{bmatrix}(\sum_{i=1}^n b_{i1}\mathbf{x}_i) \\ \vdots \\ (\sum_{i=1}^n b_{in}\mathbf{x}_i)\end{bmatrix} = \begin{bmatrix}(\sum_{i=1}^n b_{i1}\mathbb{I}_m\mathbf{x}_i) \\ \vdots \\ (\sum_{i=1}^n b_{in}\mathbb{I}_m\mathbf{x}_i)\end{bmatrix} = \begin{bmatrix}b_{11}\mathbb{I}_m & \dots & b_{n1}\mathbb{I}_m \\ \vdots & \ddots & \vdots \\ b_{1n}\mathbb{I}_m & \dots & b_{nn}\mathbb{I}_m\end{bmatrix}\begin{bmatrix}\mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n\end{bmatrix} \\
&= \left(B^\top \otimes \mathbb{I}_m\right)\operatorname{vec}(X)\,. \qquad (4)
\end{aligned}
$$

We can now apply vertorization transformation in both sides of equation (1), and use the two relationships (3), (4) and the fact that vec is a linear transformation:

$$
\begin{aligned}
&\operatorname{vec}(AX - XB) = \operatorname{vec}(C) \\
\iff &\operatorname{vec}(AX) - \operatorname{vec}(XB) = \operatorname{vec}(C) \\
\iff &\left(\mathbb{I}_n \otimes A - B^\top \otimes \mathbb{I}_m\right)\operatorname{vec}(X) = \operatorname{vec}(C) \\
\iff &\mathcal{A}\mathbf{x} = \mathbf{c}\,,
\end{aligned}
$$

with $\mathcal{A} := \mathbb{I}_n \otimes A - B^\top \otimes \mathbb{I}_m$, $\mathbf{x} := \operatorname{vec}(X)$, $\mathbf{c} := \operatorname{vec}(C)$.

3 Uniqueness of solution

We now show that equation (1) has a unique solution if and only if $\sigma(A) \cap \sigma(B) = \varnothing$, where $\sigma(\cdot)$ denotes the spectrum of a matrix.

To do this, we need to relate eigenpairs of $\mathcal{A}$ to eigenpairs of A and B. But first, we will need the mixed product property, i.e. to relate matrix product with Kronecker product:

$$
\begin{aligned}
\left((D_{l\times m} \otimes E_{n\times k})\,(F_{m\times p} \otimes H_{k\times q})\right)_{ij} &= \begin{bmatrix}d_{11}E & \dots & d_{1m}E \\ \vdots & \ddots & \vdots \\ d_{l1}E & \dots & d_{lm}E\end{bmatrix}_{i:}\begin{bmatrix}f_{11}G & \dots & f_{1m}G \\ \vdots & \ddots & \vdots \\ f_{l1}G & \dots & f_{lm}G\end{bmatrix}_{:j} \\
&= \sum_{t=1}^{m}(d_{it}E)\cdot(f_{tj}G) = \left(\sum_{t=1}^{m} d_{it}f_{tj}\right)(EG) \\
&= (DF)_{ij}\cdot(EG) \\
&= (DF \otimes EG)_{ij}\,,
\end{aligned}
$$

which shows that $(D \otimes E)(F \otimes G) = (DF) \otimes (EG)$. We denote $(\lambda, \mathbf{v})$ an eigenpair of A and $(\mu, \mathbf{u})$ an eigenpair of $B^\top$.

$$\begin{aligned}\mathcal{A}(\mathbf{u}\otimes\mathbf{v}) &= (\mathbb{I}_n\otimes A)(\mathbf{u}\otimes\mathbf{v}) - (B^\top\otimes\mathbb{I}_m)(\mathbf{u}\otimes\mathbf{v})\\ &= (\mathbb{I}_n\mathbf{u})\otimes(A\mathbf{v}) - (B^\top\mathbf{u})\otimes(\mathbb{I}_m\mathbf{v})\\ &= \mathbf{u}\otimes(\lambda\mathbf{v}) - (\mu\mathbf{u})\otimes\mathbf{v}\\ &= (\lambda-\mu)(\mathbf{u}\otimes\mathbf{v})\end{aligned}$$

i.e., the vector $\mathbf{u}\otimes\mathbf{v}$ is an eigenvector of $\mathcal{A}$ associated to the eigenvalue $\lambda-\mu$. We can now derive the property for uniqueness of solution:

$$\begin{aligned}AX - XB = C \text{ has a unique solution } &\iff \mathcal{A}\mathbf{x} = \mathbf{c} \text{ has a unique solution}\\ &\iff \mathcal{A} \text{ is invertible}\\ &\iff 0 \text{ is not an eigenvalue of } \mathcal{A}\\ &\iff \lambda-\mu\neq 0,\ \forall\,\lambda\in\sigma(A),\ \mu\in\sigma(B^\top)\\ &\iff \sigma(A)\cap\sigma(B^\top) = \varnothing\\ &\iff \sigma(A)\cap\sigma(B) = \varnothing\,, \qquad (5)\end{aligned}$$

in the last step, we used the fact that eigenvalues of the transpose of a matrix are the same as the eigenvalues of the untransposed matrix.

4 Special case — Lyapunov equation

A special case of (1) is when $B = -A^\top$ and C is symmetric, which yields the Lyapnuov equation:

$$AX + XA^\top = C, \quad C = C^\top\,. \qquad (6)$$

We will show that X is symmetric under the condition that the solution is unique, and we will give a counter-example if uniqueness does not hold.

4.1 Symmetry of X

In this special case, we can rewrite $\mathcal{A} = \mathbb{I}_n\otimes A + A\otimes\mathbb{I}_n$. We use again properties (3) and (4) to express:

$$\begin{aligned}\operatorname{vec}(X^\top A^\top) &= (A\otimes\mathbb{I}_n)\operatorname{vec}(X^\top)\\ \operatorname{vec}(AX^\top) &= (\mathbb{I}_n\otimes A)\operatorname{vec}(X^\top)\,.\end{aligned}$$

We can now exploit the fact that C is symmetric in (6):

$$\begin{aligned}C = C^\top &\iff AX + XA^\top = X^\top A^\top + AX^\top\\ &\iff \mathcal{A}\operatorname{vec}(X) = \operatorname{vec}(X^\top A^\top) + \operatorname{vec}(AX^\top)\\ &\iff \mathcal{A}\operatorname{vec}(X) = (A\otimes\mathbb{I}_n + \mathbb{I}_n\otimes A)\operatorname{vec}(X^\top)\\ &\iff \mathcal{A}\operatorname{vec}(X) = \mathcal{A}\operatorname{vec}(X^\top) \qquad (7)\end{aligned}$$

We must now invoke uniqueness of the solution X, which is equivalent to $\mathcal{A}$ being invertible. This implies, from equation (7), that $\operatorname{vec}(X) = \operatorname{vec}(X^\top) \iff X = X^\top$. Thus, in this special case, uniqueness of solution implies symmetry of X.

4.2 Counter-example

We now show that non-uniqueness of the solution of (6) does not guarantee symmetry of X. The non-uniqueness of the solution is equivalent to have A and $B = -A^\top$ sharing at least one eigenvalue (see the previous section about uniqueness of the solution and eigenvalues). Given that:

$$\lambda \in \sigma(-A^\top) \iff -\lambda \in \sigma(A)\,,$$

it is sufficient to craft a matrix A having eigenvalues λ and $-\lambda$. For simplicity, let us work in two dimensions with A being diagonal:

$$X := \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad A := \begin{bmatrix} \lambda & 0 \\ 0 & -\lambda \end{bmatrix}.$$

Equation (6) now reads as:

$$AX + XA^\top = \begin{bmatrix} a\lambda & b\lambda \\ -c\lambda & -d\lambda \end{bmatrix} + \begin{bmatrix} a\lambda & -b\lambda \\ c\lambda & -d\lambda \end{bmatrix} = \begin{bmatrix} 2a\lambda & 0 \\ 0 & -2d\lambda \end{bmatrix} = C\,,$$

which is indeed symmetric. This holds for any arbitrary matrix X. In particular, this holds for any matrix X with $b \neq c$ (i.e., X is non-symmetric). So we found a counter-example for which there are infinitely many non-symmetric solutions.

4.3 Optional Question — Positive (semi-)definiteness

We show that if C is negative semi-definite and the eigenvalues of A have real negative part, then X is positive semi-definite.

4.3.1 Proof using closed-form solution

We will first assume the closed-form solution of equation (6):

$$X := -\int_0^\infty e^{A^\top u} C e^{Au} du \tag{8}$$

We will show that this is a solution. Note that this solution must be unique, since we assume that eigenvalues of A have negative real part, thus A has no two eigenvalues λ, $\mu \in \sigma(A)$ such that $\lambda = -\mu$. Using the easily-provable derivative of the matrix exponential $\frac{d}{du} e^{Au} = Ae^{Au} = e^{Au}A$, we can show that $\frac{d}{du}(e^{A^\top u} C e^{Au}) = A^\top e^{A^\top u} C e^{Au} + e^{A^\top u} C e^{Au} A$. Plugging equation 8 into 7 yields

$$AX + XA^\top = -\int_0^\infty \left(Ae^{A^\top u} C e^{Au} + e^{A^\top u} C e^{Au} A^\top \right) du = -\int_0^\infty \frac{d}{du}\left(e^{A^\top u} C e^{Au} \right) du = \left[e^{A^\top u} C e^{Au} \right]_\infty^0$$

In order to evaluate the term where $u \to \infty$, we need to consider the Jordan form of A. Let $J := \mathrm{diag}(J_1, \ldots, J_k) = P^{-1}AP$ be the Jordan form of A, with k jordan blocks J_i. Since $\exp(Au) = P\exp(Ju)P^{-1}$ and $\exp(Ju) = \mathrm{diag}(\exp(J_1 u), \ldots, \exp(J_k u))$, we can evaluate each $\exp(J_i u)$ individually.

If the block J_i correspond to the eigenvalue $\lambda_i(A) \in \mathbb{R}$ with multiplicity m, then $J_i \in \mathbb{R}^{m \times m}$. We can decompose $J_i = \lambda_i \mathbb{I}_m + N$ into a diagonal matrix $\lambda_i \mathbb{I}_m$ and a nilpotent matrix N of order m:

$$J_i = \begin{bmatrix} \lambda_i & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_i \end{bmatrix} = \begin{bmatrix} \lambda_i & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \lambda_i \end{bmatrix} + \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & 0 \end{bmatrix} = \lambda_i \mathbb{I}_m + N$$

Since $\lambda_i \mathbb{I}_m$ and N commute, $\exp(J_i) = \exp(\lambda_i \mathbb{I}_m + N) = \exp(\lambda_i \mathbb{I}_m)\exp(N) = \exp(\lambda_i)\exp(N)$. Thus, $\exp(J_i u) = \exp(\lambda_i u)\exp(Nu)$, and one can easily see that $\exp(Nu)$ is made of polynomials of the form $u^p/p!$ by observing the pattern of the powers $(Nu)^p$:

$$\exp(Nu) = \begin{bmatrix} 1 & u & \dots & \frac{u^{m-1}}{(m-1)!} \\ & 1 & & \frac{u^{m-2}}{(m-2)!} \\ & & \ddots & \vdots \\ & & & 1 \end{bmatrix}$$

If we rewrite $\exp(Au)C\exp(A^\top u) = P\exp(Ju)\tilde{C}\exp(J^\top u)P^\top$, with $\tilde{C} = P^{-1}CP^{-\top}$ and assume that all eigenvalues of A are real, one can thus easily see that each element of this matrix is a linear combination of functions of the form $e^{\lambda_i u}u^p$. Since $\lambda_i < 0$, we thus have $\lim_{u\to\infty} e^{A^\top u}Ce^{Au} = 0$.

If A has some complex eigenvalues, the proof is a bit more involved, but analogous. The Jordan blocks J_i are themselves block diagonal matrices of size $2m \times 2m$ corresponding to eigenvalue $\lambda_i(A) = a_i + jb_i \in \mathbb{C}$ of multiplicity m, with 2×2 blocks F_i:

$$F_i := \begin{bmatrix} a_i & b_i \\ -b_i & a_i \end{bmatrix}$$

Evaluating each $\exp(J_i)$ individually thus boils down to evaluating each of those m F_i blocks. We decompose into $F_i = a_i\mathbb{I}_2 + b_iG$, where G is an antidiagonal matrix with elements $(-1, 1)$. We note that $G^2 = -\mathbb{I}_2$, which allows us to evaluate:

$$\begin{aligned} \exp(F_i) &= \exp(a_i\mathbb{I}_2 + b_iG) = \exp(a_i\mathbb{I}_2)\exp(b_iG) \\ &= \exp(a_i)\cdot\sum_{k=0}^{\infty}\frac{(b_iG)^k}{k!} = \exp(a_i)\cdot\sum_{k=0}^{\infty}\left[\frac{(b_iG)^{2k}}{(2k)!} + \frac{(b_iG)^{2k+1}}{(2k+1)!}\right] \\ &= \exp(a_i)\cdot\sum_{k=0}^{\infty}(-1)^k\left[\frac{(b_i)^{2k}}{(2k)!}\mathbb{I}_2 + \frac{(b_i)^{2k+1}}{(2k+1)!}G\right] \\ &= e^{a_i}\left(\mathbb{I}_2\cos b_i + G\sin b_i\right) \end{aligned}$$

Thus, each element of the matrix $\exp(Au)C\exp(A^\top u) = P\exp(Ju)\tilde{C}\exp(J^\top u)P^\top$ will contain exponentials e^{a_i}, showing that $\lim_{u\to\infty} e^{A^\top u}Ce^{Au} = 0$.

Thus, we can show that (8) is indeed a solution:

$$AX + XA^\top = \left[e^{A^\top u}Ce^{Au}\right]_\infty^0 = e^{\mathbb{0}_n}Ce^{\mathbb{0}_n} - \lim_{u\to\infty} e^{A^\top u}Ce^{Au} = \mathbb{I}_nC\mathbb{I}_n - \mathbb{0}_n = C$$

We must now derive positive semi-definiteness of X from equation (8). One can easily see that $e^{A^\top u}Ce^{Au}$ is negative semi-definite by evaluating the quadratic form $\mathbf{x}^\top e^{A^\top u}Ce^{Au}\mathbf{x} = \mathbf{y}^\top C\mathbf{y}$, with $\mathbf{y} := e^{Au}\mathbf{x}$. Since e^{Au} is invertible and C is negative semi-definite, $e^{A^\top u}Ce^{Au}$ is negative semi-definite as well. An integral of a negative semi-definite matrix is negative semi-definite as well. The minus sign makes it a positive semi-definite matrix, so X is positive semi-definite.

One can guarantee positive definiteness of X if A has eigenvalues with real negative parts, and if C is negative definite (since an integral of a negative definite matrix is negative definite, the minus sign makes it a positive definite matrix).

4.3.2 Alternative (weaker) proof

Let me present an alternative, but weaker, proof of the optional question. If we further *assume diagonalizability of A*, we can explicitly prove positive semi-definiteness of X without assuming any *a priori* expression.

Let $\mathbf{v} \in \mathbb{C}^n$ be a vector and let $\mathbf{v} = \mathbf{v}_R + j\mathbf{v}_I$ be its decomposition into the real and imaginary parts so that $\mathbf{v}_R, \mathbf{v}_I \in \mathbb{R}^n$. We first evaluate the quadratic form:

$$\begin{aligned}\mathbf{v}^* C\mathbf{v} &= (\mathbf{v}_R^\top - j\mathbf{v}_I^\top)C(\mathbf{v}_R + j\mathbf{v}_I) \\ &= \mathbf{v}_R^\top C\mathbf{v}_R + j\mathbf{v}_R^\top C\mathbf{v}_I - j\mathbf{v}_I^\top C\mathbf{v}_R + \mathbf{v}_I^\top C\mathbf{v}_I \\ &= \underbrace{\mathbf{v}_R^\top C\mathbf{v}_R}_{\leq 0} + \underbrace{\mathbf{v}_I^\top C\mathbf{v}_I}_{\leq 0} \leq 0\end{aligned}$$

From line 2 to line 3, we use the fact that $\mathbf{v}_R^\top C\mathbf{v}_I = \mathbf{v}_I^\top C\mathbf{v}_R$ since C is symmetric, whereas the last statement uses the fact that C is negative semi-definite.

We will now evaluate a quadratic form on the left-hand side of equation (6).

Since all eigenvalues of A have a negative real part, A has no two eigenvalues $\lambda, \mu \in \sigma(A)$ such that $\lambda = -\mu$, and thus there is a unique symmetric solution X. We can thus orthogonally diagonalize it: $X = Q\Lambda Q^\top$ and rewrite equation (6):

$$\tilde{A}\Lambda + \Lambda\tilde{A}^\top = \tilde{C} \tag{9}$$

with $\tilde{A} := Q^\top AQ$, $\tilde{C} := Q^\top CQ$. Since those are similarity transformations, $\sigma(\tilde{A}) = \sigma(A)$ and $\sigma(\tilde{C}) = \sigma(C)$. In particular, eigenvalues of $\tilde{A}$ also have negative real part, and $\tilde{C}$ is negative semi-definite as well.

Let $(\lambda, \mathbf{v})$ be an eigenpair of $\tilde{A}^\top$. Note that $\mathbf{v}^\star$ is a left eigenvector of $\tilde{A}$ associated to the eigenvalue $\bar{\lambda}$, since $\tilde{A}^\top\mathbf{v} = \lambda\mathbf{v} \iff \mathbf{v}^\star(\tilde{A}^\top)^\star = \lambda^\star\mathbf{v}^\star \iff \mathbf{v}^\star\tilde{A} = \bar{\lambda}\mathbf{v}^\star$. We write from equation (9),

$$\begin{aligned}0 \geq \mathbf{v}^\star\tilde{C}\mathbf{v} &= \mathbf{v}^\star\tilde{A}\Lambda\mathbf{v} + \mathbf{v}^\star\Lambda\tilde{A}^\top\mathbf{v} \\ &= (\bar{\lambda}\mathbf{v}^\star)\Lambda\mathbf{v} + \mathbf{v}^\star\Lambda(\lambda\mathbf{v}) \\ &= (\bar{\lambda} + \lambda)(\mathbf{v}^\star\Lambda\mathbf{v})\end{aligned}$$

As eigenvalues of $\tilde{A}$ have a negative real part, $(\bar{\lambda} + \lambda) = 2\,\mathrm{Re}(\lambda)$ is negative. Dividing by $(\bar{\lambda} + \lambda)$ in the above equation implies:

$$0 \leq \mathbf{v}^\star\Lambda\mathbf{v} = \sum_{i=1}^{n} \lambda_i |v_i|^2$$

where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ are the eigenvalues of X. This holds for any left eigenvector $\mathbf{v}$ of $\tilde{A}$. If we assume that A is diagonalizable, $\tilde{A}$ is diagonalizable as well, and any vector $\mathbf{w} \in \mathbb{C}^n$ can be expanded in the eigenbasis of $\tilde{A}$. The quadratic form $\mathbf{w}^\star\Lambda\mathbf{w}$ would then always be positive, which makes the matrix X positive semi-definite.

5 Implementation of Sylvester equation solver

In order to solve equation 1, we will use the Bartels–Stewart algorithm, which consists in three phases. First, we compute the Schur form of A and B:

$$A = URU^\top, \quad B = VSV^\top ,$$

and compute $D = U^\top CV$. Second, we solve the transformed equation

$$RZ - ZS = D ,$$

with Z the unknown, exploiting the triangular structure of the system. Finally, we compute the solution by mapping Z back to the original coordinate system:

$$X = UZV^\top .$$

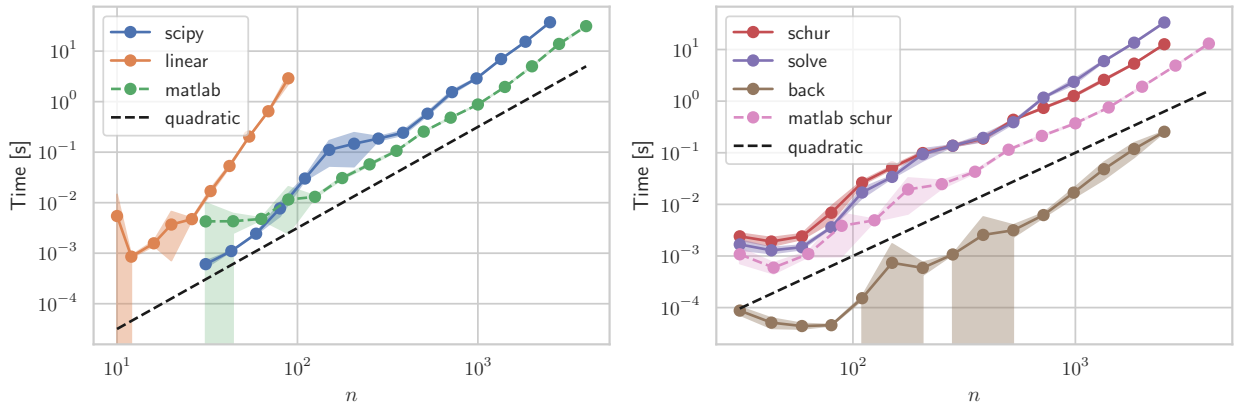

Figure 1: Basic benchmark timings on square $n \times n$ random matrices. Dots correspond to the mean over multiple runs, shaded area correspond to $\pm$ standard deviation. Black dashed lines represent a quadratic relationship. **Left**: Comparison of two solvers, `scipy.linalg.solve_sylvester` and `numpy.linalg.solve` (which solves the linear system of Equation (2)). Matrices are provided as-is, i.e. not transformed to their Schur form. Timings for the MATLAB function `sylvester` are included for comparison. **Right**: time decomposition of the three phases, i.e. (i) the schur decomposition and transformation of the equation, (ii) solving the triangular system, and (iii) map coordinates back to the original system. Timings for the MATLAB function `schur` are included for comparison.

In order to solve the triangular system (phase 2), we will use and compare three solvers: the function `scipy.linalg.solve_sylvester` from the well-known `scipy` library that is built on top of `numpy`, the function `numpy.linalg.solve` which we use to solve the linear system described in Equation 2, and the `rtrgsyl` function, which implements the algorithm presented in [1].

However, note that `rtrgsyl` relies on other solvers: it recursively breaks computations into smaller ones, until a user-defined matrix size is reached, and another solver is used.

Also, note that scipy's solver also relies on the Bartel-Steward algorithm [1]. Thus, if we provide matrices that are already in Schur form, it will still attempt to compute their Schur form, which may be a loss of resources (but this cannot be disabled).

All benchmarks are performed over several runs, the number of runs is a function of the matrix size (typically 50-100 runs for $n < 100$ and 2-5 runs for $n > 1000$), the function looks like a staircase when plotting number of runs (linear scale) against matrix size (log scale). Details can be found in notebooks.

5.1 Basic benchmarks

Before evaluating the performance of our recursive algorithm and analyzing the optimal block size, we compare the solvers we use for small systems alone. For completeness, we also include the timings for the corresponding MATLAB functions. Note that, for the solvers, the tested matrix sizes are much bigger than what will be used later, but it enables to assess the asymptotic complexity.

Note that we will test matrix sizes $n \times n$ much bigger than what we will use later, but the computation time for small matrices is hard to assess.

The left subplot of Figure 1 compares the scipy's solver of the Sylvester equation and the solver for linear systems. The scipy's solver is significantly more efficient. Note that the linear solver is extremely memory inefficient, as it needs to build the matrix $\mathcal{A} \in \mathbb{R}^{mn \times mn}$.

The right subplot of Figure 1 compares the timings of the three phases of the Bartel-Stewart algorithm, using scipy's solver (which thus gets matrices in their Schur form). The schur decomposition takes as much time as solving the triangular system, for $10 < n < 1000$. This suggests that there is no significant performance issue about scipy's solver attempting to compute Schur form again.

[1] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.solve_sylvester.html`

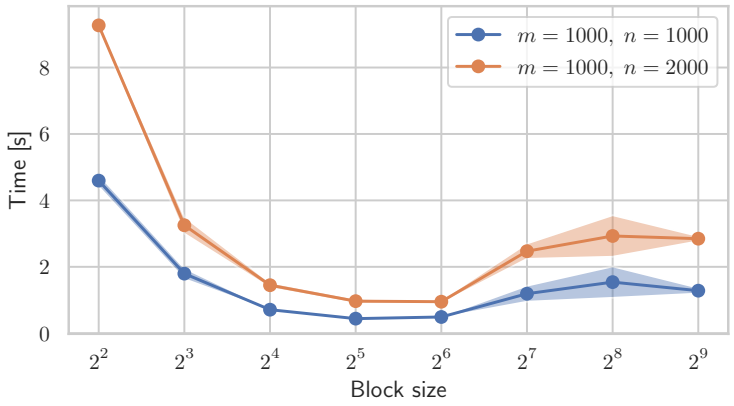

Figure 2: Evaluation of the optimal block size of `rtrgsyl` with random matrices over multiple runs. Dots correspond to mean, shaded area to ± standard deviation. The y axis reports the time of phase two of the Bartel-Stewart algorithm, i.e. Schur decomposition and back transformation are ignored. The solver `scipy.linalg.solve_sylvester` is used when `rtrgsyl` reaches the block size. Refer to the notebook `2-Benchmark-blks.ipynb` to see the implementation.

5.2 Selecting an optimal block size

We know analyze how the recursion stopping criterion of `rtrgsyl` affects computation time. We fix the matrix sizes, while varying the block size over a grid. In this section, we will ignore the timings of phase 1 and 3 of Bartel-Stewart, since those are unrelated to the optimal block size. We use scipy's solver when the recursion stops.

We deduce from Figure 2 that the optimal block size lies between 32 and 64. Note that the inefficiency for very small block sizes (4-8) is expected. First, the algorithm `rtrgsyl` is implemented in Python, a high-level language, and thus recursions are slow. On the other hand, scipy's solver is implemented in C, a low-level language. Second, scipy's solver is expected to be optimized for large systems, and we do not benefit from these optimizations with very small systems. Last, each recursion requires to put pieces back together after solving them individually, and we loose the speedup provided by vectorization of `numpy` array operations (again, implemented in a low-level language).

5.3 Comparison: with vs without recursion

The long-waited moment has come! We will compare `rtrgsyl` with scipy's solver against scipy's solver alone. `rtrgsyl` requires matrices in their Schur form, whereas scipy's solver doesn't. But in order to have a fair comparison, we will also provide scipy's solver matrices in their Schur form, and we will compare timings of phase two only (since phase 1 and 3 are the same).

Figure 3 depicts the comparison. The `rtrgsyl` implementation is significantly faster than scipy's solver alone, especially for big matrices ($n > 1000$) where the slopes on log-log scale diverge.

5.4 Scalability

In this section, we explore the performance in function of the number of threads. We test different matrix sizes ($n \times n$), with the optimal block size 64 for `rtrgsyl`. We only take into account the time for phase two, i.e. matrices are provided in their Schur form to the algorithms. Figure 4 displays the results. Surprisingly, there is no difference in computational time when the number of threads changes. Note that the CPU usage per core was carefully monitored while performing the benchmarks, and when k threads were selected, those k cores were indeed used at almost 100%.

It is probably the case that scipy's solver is surprisingly not optimized for multi-threading. However, the fact that the selected number of threads are used at 100% without improving efficiency remains unexplained. Since `rtrgsyl` uses scipy's solver in this case, it is consistent to also observe no improvement when increasing

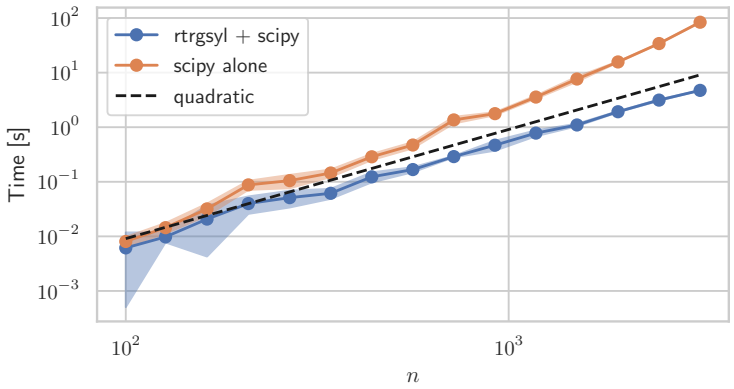

Figure 3: Comparison of `rtrgsyl` with scipy's solver against scipy's solver alone, for different matrix sizes ($n \times n$). Matrices are random, each case is solved several times. Dots represent the mean timing of phase two, shaded area represents $\pm$ standard deviation. For `rtrgsyl`, the block size is set to 64. The dashed black line has a quadratic relationship $y \propto x^2$ to ease plot interpretation. Refer to the notebook `3-Benchmark-rtrgsyl.ipynb` to see the implementation.

the number of threads. Note that this scalability benchmark was also performed on another computer and another operating system (but with the same code), similar results were obtained.

References

[1] Isak Jonsson and Bo Kågström. "Recursive blocked algorithms for solving triangular systems—Part I: one-sided and coupled Sylvester-type matrix equations". In: *ACM Transactions on Mathematical Software* 28.4 (Dec. 2002), pp. 392–415. ISSN: 0098-3500. DOI: 10.1145/592843.592845. URL: https://doi.org/10.1145/592843.592845 (visited on 06/05/2021).

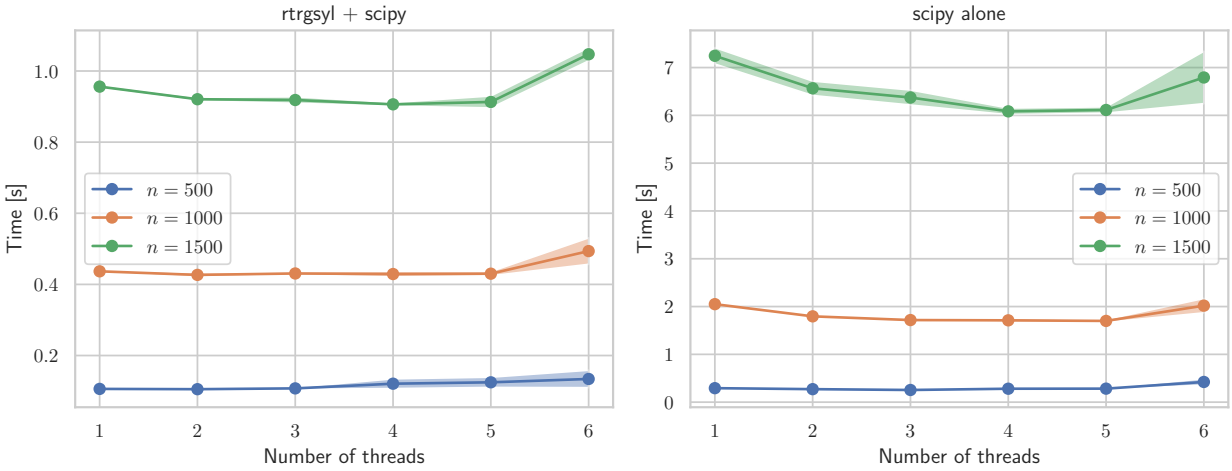

Figure 4: Evaluation of the performance of algorithms in function of the number of threads. Random matrices are used, each case is computed several times. Dots represent mean time of phase two, shaded area represent $\pm$ standard deviation. Three different matrix sizes $n \times n$ are used: $n = 500, 1000, 1500$. **Left**: `rtrgsyl` algorithm using scipy's solver, with block size 64. **Right**: scipy's solver alone. Refer to the notebook `4-Benchmark-scalability.ipynb` to see the implementation.