# Computation of matrix functions with fully automatic Schur-Parlett and Rational Krylov methods.

Roberto Zanotto

## Matrix Functions

$\mathrm{sqrt}(A), \exp(A), f(A)$ ... where $A$ is a square matrix.

Useful generalization of scalar function $f : \mathbb{C} \to \mathbb{C}$

Examples of applications:

› Stiff differential equations;

› Nuclear magnetic resonance;

› Control theory.

Mathematical definition of $f(A)$ can be obtained through polynomial interpolation.

Matrix polynomials:

$$p(x) = x^3 + 4x - 7, \qquad p(A) = A^3 + 4A - 7I$$

Interesting property: value of $p(A)$ is fully determined by the values of $p$ on the spectrum of $A$:

$$p(A) = q(A) \Longleftrightarrow p^{(j)}(\lambda) = q^{(j)}(\lambda), \qquad \forall \lambda \in \Lambda(A)$$

By extending this property to matrix functions, we get a definition of $f(A)$:

$$f(A) \coloneqq p(A)$$

where $p$ takes the same values of $f$ on the spectrum of $A$, i.e.

$$p^{(j)}(\lambda) = f^{(j)}(\lambda), \qquad \forall \lambda \in \Lambda(A)$$

## Computation of Matrix Functions

Hermite interpolation with Vandermonde matrix is unstable and the resulting method is slow.

There are specialized methods (scaling and squaring for $\mathrm{expm}$ and $\mathrm{logm}$, Björk-Hammarling for $\mathrm{sqrtm}$).

We want an algorithm that is both:

› **Generic** – working for any function $f$ and matrix $A$;

› **Automatic** – requiring no other information form the user (such as derivatives of $f$ or other data structures).

## Automatic Differentiation

**Dual numbers** can be used to compute derivatives in an automatic, efficient and stable manner:

$\boldsymbol{x} + \boldsymbol{y\varepsilon}$ with $\varepsilon \neq 0,\ \varepsilon^2 = 0$

$$f(x + y\varepsilon) = f(x) + f'(x)y\varepsilon \quad \Rightarrow \quad f'(x) = \mathrm{Eps}(f(x+\varepsilon))$$

We use the Julia language (a “new Matlab”), whose type-system allows for pain-free use of dual numbers on user-defined functions.

Dual numbers can be generalized for higher-order derivatives (TaylorSeries.jl).

## Automatic Differentiation – example

$f(x) = (x-3)^8$

```
function f(x)
        x -= 3
        for i = 1:3
                x *= x
        end
        return x
end
```

We can compute $f'(5)$ with

$f(5+\varepsilon)$:

$(5+\varepsilon)-3 = 2+\varepsilon$

$(2+\varepsilon)(2+\varepsilon) = 4+4\varepsilon$

$(4+4\varepsilon)(4+4\varepsilon) = 16+32\varepsilon$

$(16+32\varepsilon)(16+32\varepsilon) = 256+1024\varepsilon$

The result $f(5+\varepsilon) = 256+1024\varepsilon$ means $f(5)=256$ and $f'(5) = 1024$

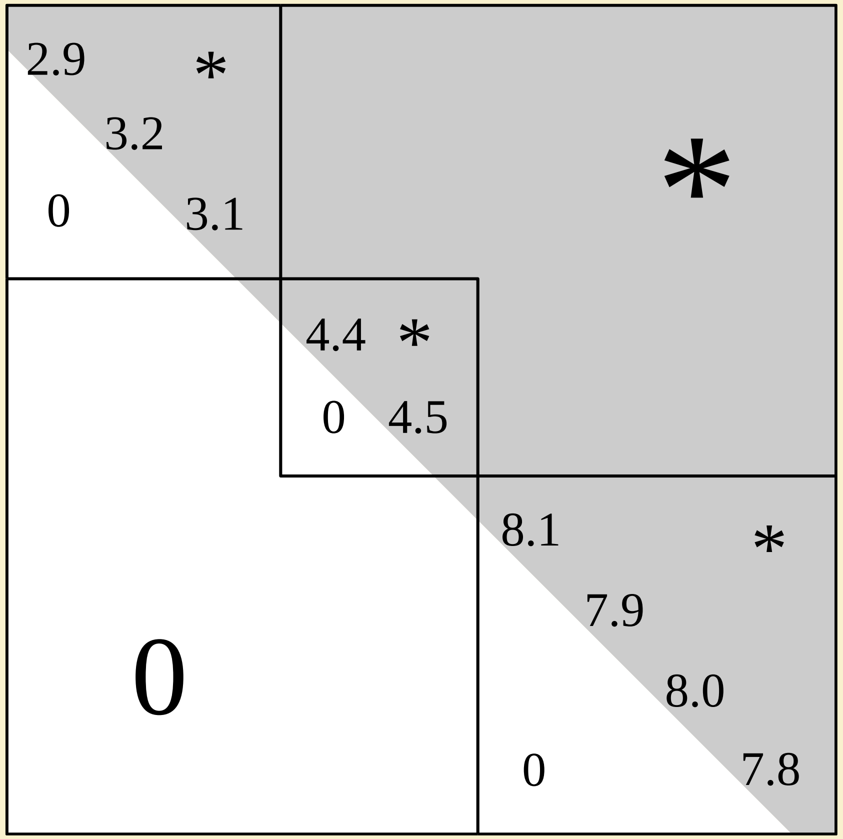
# Schur-Parlett – dense matrix functions

Proposed by Higham and Davies in 2003.

$A = QTQ^* \Rightarrow f(A) = Qf(T)Q^*$

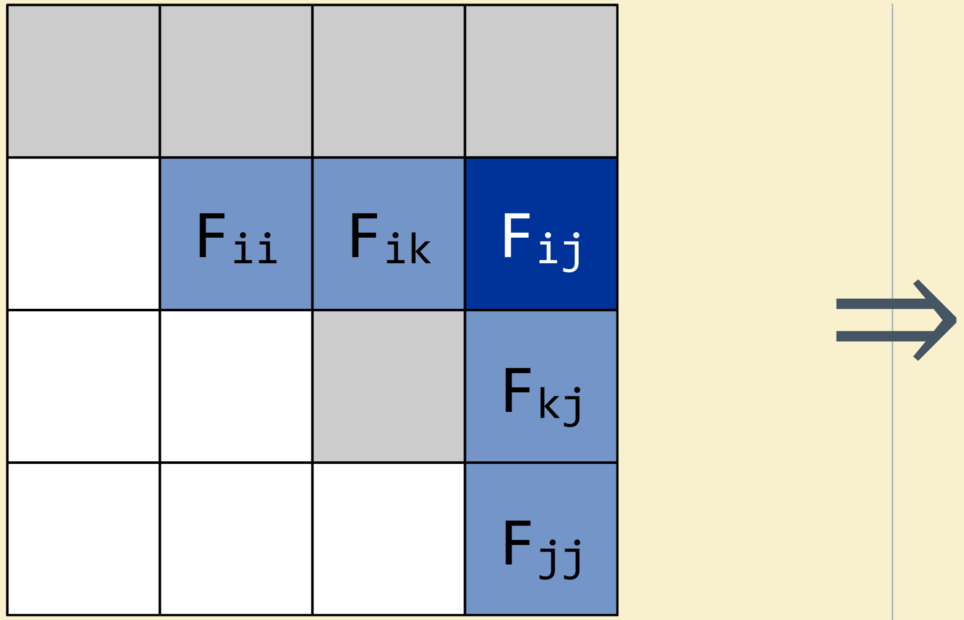
Need to compute $F = f(T)$:

1. Group eigenvalues in blocks by proximity;
2. Compute $f(T_{ii})$ of diagonal blocks with Taylor;
3. Use the Parlett recurrence to reconstruct the upper part.

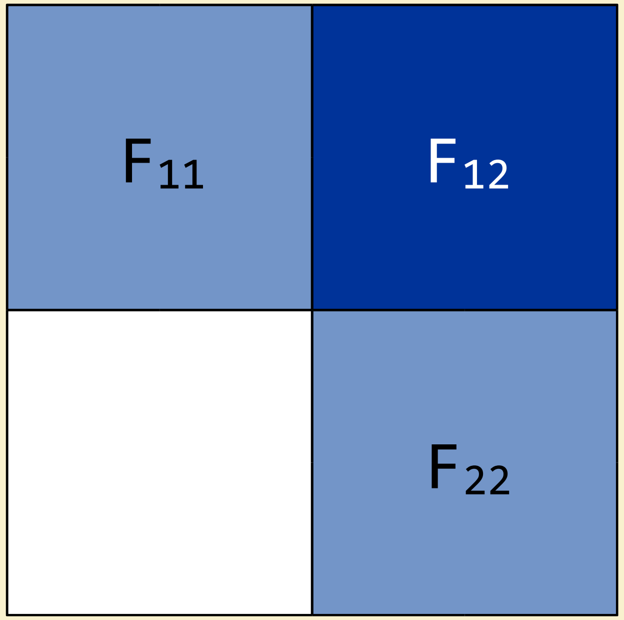


# Schur-Parlett – Parlett recurrence

Parlett recurrence made recursive and cache-oblivious.



$$T_{ii}F_{ij} - F_{ij}T_{jj} = F_{ii}T_{ij} - T_{ij}F_{jj} + \sum_{k=i+1}^{j-1} (F_{ik}T_{kj} - T_{ik}F_{kj})$$



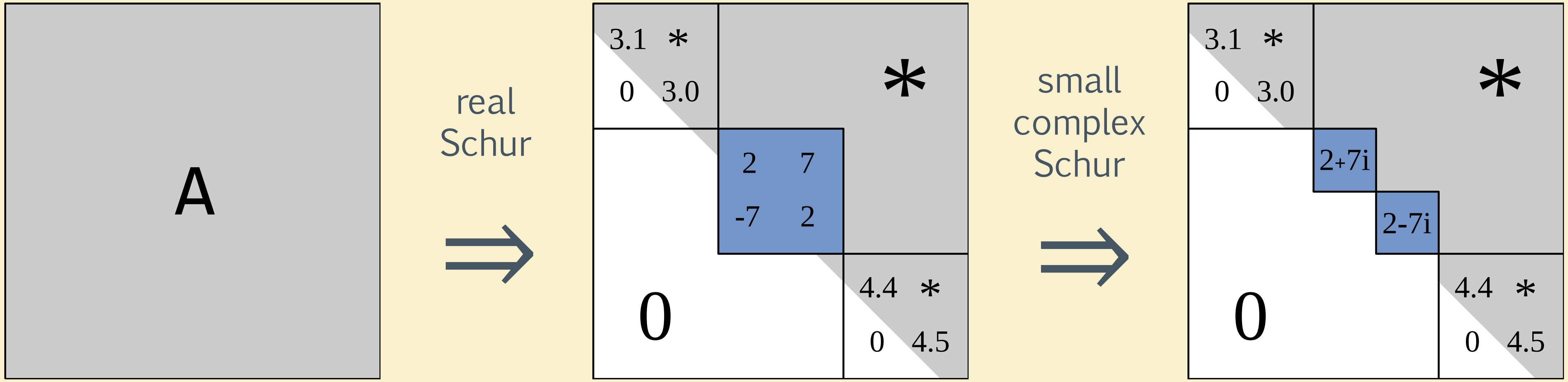
$$T_{11}F_{12} - F_{12}T_{22} = F_{11}T_{12} - T_{12}F_{22}$$

~3x speedup for n = 2500

**Problem:** conjugated eigenvalues with big imaginary part must go in different blocks, even with real $A$.

**Original solution:** do everything in complex arithmetic.

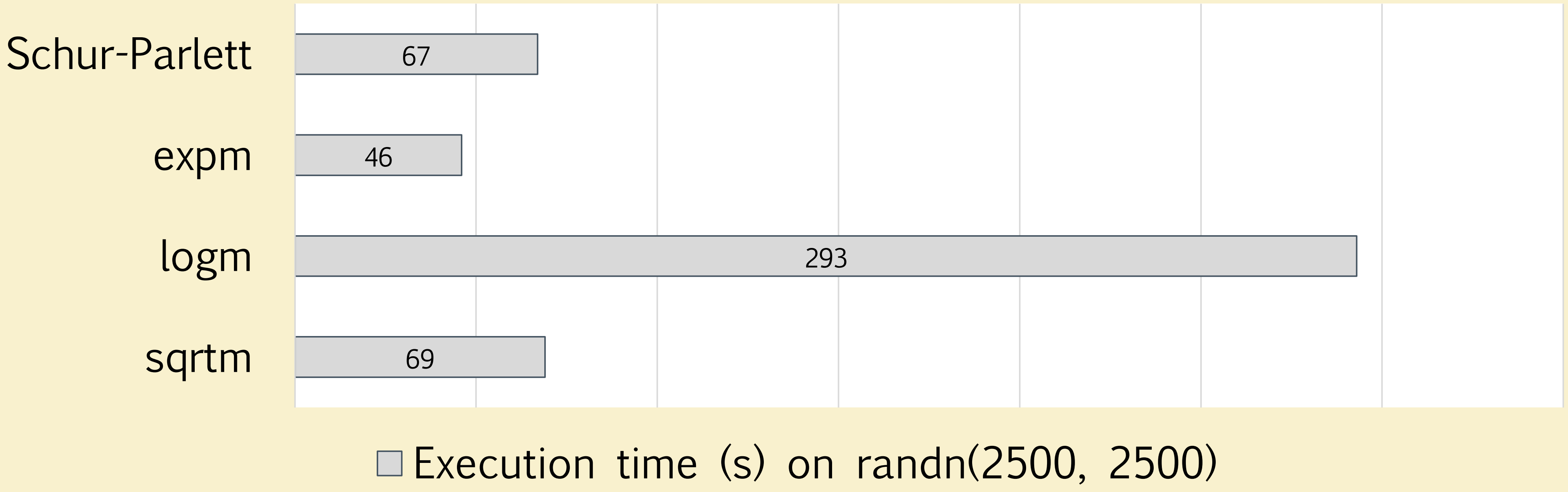
**Our solution:** complex Schur factorization can be "delayed" and done on small blocks:



Also allows for Parlett recurrence with real Sylvester equations, ~2x speedup for n = 2500 on whole Schur-Parlett.

## Schur-Parlett – performance results

The whole procedure typically spends ~2/3 of the time doing $A$'s Schur decomposition (varies depending on the eigenvalues' distribution).



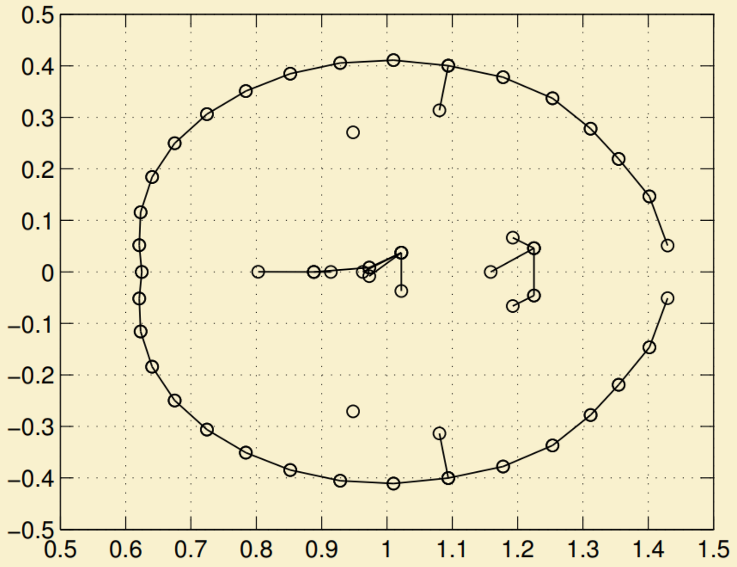
# Schur-Parlett – numerical accuracy

| | Specialized method | Schur-Parlett |
|---|---|---|
| exp | 1.0e-14 | 1.5e-13 |
| log | 4.0e-14 | 4.1e-14 |
| sqrt | 4.3e-14 | 4.5e-14 |

Relative error versus Matlab's Symbolic Toolbox (on 50x50 random matrix).

Boils down to how well the eigenvalues can be clustered.

Can behave badly (7e-4) with "snake" eigenvalues (as shown in original paper, experiment 4)



## Code comparison for the matrix cube root $A^{1/3}$

### Matlab

```
function r = cuberoot(x, k)
     c = 1.0;
     for i = 0:k-1
          c = c*(1/3 - i);
     end
     r = c*x^(1/3 - k);
end

funm(A, @cuberoot)
```

### Our Julia implementation

```
schurparlett((x)->x^(1/3), A)
```

# Rational Krylov – sparse matrix functions

(as described by Güttel in 2013)

Goal: compute $f(A)b$ for a sparse A.

Approximate $f(A)b$ with $r_m(A)b$ where $r_m = p_{m-1}/q_{m-1}$

Denominator is factored as $q_{m-1}(z) = \prod_{j=1}^{m-1}(1 - z/\xi_j)$
with poles $\xi_j$ provided by the user.

Rational Krylov space is defined as:

$$Q_m(A,b) = q_{m-1}(A)^{-1} span\{b, Ab, \dots, A^{m-1}b\}$$

Obtained with Ruhe's rational Arnoldi algorithm:

$$v_1 = b/||b||, \quad v_{j+1} = orthonormalize((I - A/\xi_j)^{-1}Av_j)$$

$r_m(A)b$ is computed by projecting $A$ into the Krylov space:

$$A_m = V^*AV, \quad r_m(A)b = Vf(A_m)V^*b$$

## Rational Krylov – approximation accuracy

Depends on two factors:

- How well $f$ can be approximated on $A$'s spectrum by a rational function, hopefully with low degree (ill-posed problem);
- How well we choose the poles $\xi_j$.

We use the AAA algorithm for rational approximation to find good poles automatically.

## AAA algorithm for rational approximation

Proposed by Nakatsukasa, Sète, Trefethen in 2017.

Input: function samples (real or complex).
Output: rational barycentric function of type $(m-1, m-1)$:

$$r(z) = \frac{n(z)}{d(z)} = \sum_{j=1}^{m} \frac{w_j f_j}{z - z_j} \Big/ \sum_{j=1}^{m} \frac{w_j}{z - z_j}$$

$r(z_j) = \infty/\infty$ but $\lim_{z \to z_j} r(z) = f_j$

Support/interpolation points $(z_j)$ are chosen incrementally from samples in a greedy way, to avoid instabilities:

next $z_j$ is chosen where $|f(z_j) - r(z_j)|$ is maximized.

After a new support point is found, weights are recomputed to minimize the approximation error:

$$f(z) \approx \frac{n(z)}{d(z)} \rightarrow minimize||f(z)d(z) - n(z)||, \qquad z \in Z^{(m)}$$

Is a least squares problem solvable with SVD:

$$minimize||Aw||, \qquad ||w|| = 1$$

When approximation error is small, we are done.

Poles can be then retrieved by solving a generalized eigenvalue problem, with accuracy up to machine precision.

## Rational Krylov + AAA

1. Sample $f$ on the 0-centered disk of radius $\mathrm{norm}(A)$;
2. Find the poles of $f$ with AAA;
3. Use the poles for the rational Arnoldi approximation.

Computational costs:

Rational Krylov: $m$ times sparse linear system + orthogonalization: $O(m(L + mN))$

AAA: $m$ times SVD: $O(m(m^2 M))$

$\pi$

# Rational Krylov + AAA, Experiment 1

Matrices are from the SuiteSparse Matrix Collection.

$\exp(A)b$ with Krylov is compared with dense expm:

| Problem type | name | cond(A) | cond (exp, A) | # poles | error | # poles | error |
|---|---|---|---|---|---|---|---|
| 2D/3D problem | jagmesh3 | 1168 | 7.0e0 | 9 | 1.5e-6 | 15 | 6.0e-14 |
| Fluid dynamics | sherman4 | 2178 | 1.2e2 | 11 | 6.6e-6 | 19 | 6.7e-10 |
| Structural problem | can_1072 | 2.0e34 | 3.4e1 | 11 | 7.3e-6 | 23 | 1.7e-14 |
| Directed graph | SmaGri | Inf | 7.8e1 | 9 | 2.6e-7 | 21 | 7.3e-13 |

# Rational Krylov + AAA, Experiment 2

sqrt$(A)b$ (positive definite $A$) against dense sqrtm:

| Problem type | name | min eig(A) | norm(A) | # poles | error | # poles | error |
|---|---|---|---|---|---|---|---|
| Circuit simulatio | rajat19 | 1.7e-1 | 3.9e1 | 20 | 5.8e-6 | 45 | 3.7e-14 |
| Structural problem | nos3 | 1.8e-2 | 7.7e2 | 50 | 8.8e-4 | 100 | 5.2e-9 |
| Power network | 685_bus | 6.2e-2 | 2.6e4 | 50 | 1.5e-3 | 95 | 8.0e-5 |
| Electro magnetic | mhd 1280b | 1.5e-11 | 8.0e1 | 50 | 2.5e-2 | 50 | 8.8e-3 |

Custom sampling for AAA, more dense near 0, brings error from $10^{-3}$ to $10^{-12}$ (matrix "685_bus").

## Future Work

Different algorithms could be used for the evaluation of the diagonal blocks in Schur-Parlett (replacing Taylor).

An AAA solution could solve some of Schur-Parlett’s instability issues and lift the dependency on automatic differentiation, bringing the automatic algorithm to other languages such as Matlab and C (more research needed).

Try it out at:
https://github.com/robzan8/MatFun.jl

Questions?