Computational graphs for matrix functions

Elias Jarlebring [♭],[1] Massimiliano Fasi[♮] and Emil Ringh[♭]

(♭) Mathematics department, KTH Royal institute of technology
Stockholm Sweden.
(♮) Department of Computer Science, Durham University,
Durham, United Kingdom.

Abstract

If a function $f : \mathbb{C} \to \mathbb{C}$ is assumed to be analytic in a sufficiently large domain, there are well-established ways to generalize it to a matrix function $f : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$:

$$F = f(A). \tag{1.1}$$

Several formal definitions exist, each with slightly different assumptions, but all are consistent for a broad class of functions and matrices. A crucial area within the field of numerical linear algebra is the computation of F in various settings. This proposal concerns the computation of matrix functions and their generalizations. The most well-known example of a non-trivial matrix function is the matrix exponential: $f(A) = \exp(A)$. It represents a closed form of the solution of a system of linear ordinary differential equations, with arbitrary initial conditions. Its direct computation is extensively used in systems and control, data science, and in general natural science where modeling with partial differential equations plays an important role.

Many numerical methods for evaluating a function $f(A)$ at an $n \times n$ matrix A can be based on a variety of different approaches, but for a large class of algorithms, the matrix $f(A)$ is approximated using only three operations: 1) $C = \alpha A + \beta B$, 2) $C = AB$, 3) $C = A^{-1}B$. This includes the scaling-and-squaring algorithm, which is the computational method most commonly used for the matrix exponential (for example, in the MATLAB command `expm`). Rephrasing these methods as directed acyclic graphs (DAGs) is a particularly effective approach to study existing techniques, improve them, and eventually derive new ones, which is illustrated in [1]. The accuracy of these matrix techniques can be characterized by the accuracy of their scalar counterparts, thus designing algorithms for matrix functions can be regarded as a scalar-valued optimization problem. The derivatives needed during the optimization can be calculated automatically by exploiting the structure of the DAG, in a fashion analogous to backpropagation. This paper describes `GraphMatFun.jl`, a Julia package that offers the means to generate and manipulate computational graphs, optimize their coefficients, and generate Julia, MATLAB, and C code to evaluate them efficiently at a matrix argument. The software also provides tools to estimate the accuracy of a graph-based algorithm and thus obtain numerically reliable methods. For the exponential, for example, using a particular form (degree-optimal) of polynomials produces implementations that, in many cases, are more computationally efficient than the Padé-based techniques typically used in mathematical software. The optimized graphs and the corresponding generated code are available online.

References

[1] E. Jarlebring, M. Fasi, and E. Ringh, Computational graphs for matrix functions, *ACM Trans. Math. Software*, 2022.

[1] eliasj@kth.se