UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

Tesis doctoral

Nuevos métodos y algoritmos de altas prestaciones para el cálculo de funciones de matrices

Presentada por: Pedro A. Ruiz Martínez

Dirigida por: Dr. J. Javier Ibáñez González
Dr. Jorge Sastre Martínez

24 de enero de 2020

Agradecimientos

Después de un largo camino, por fin consigo ver la luz al final del túnel y, echando la vista atrás, soy consciente del esfuerzo y el tiempo que ha conllevado llegar hasta aquí. Afortunadamente, a lo largo del camino he sido acompañado por mucha gente a la que me gustaría expresar mi agradecimiento.

Quisiera, en primer lugar, agradecer profundamente a mis directores de tesis, Javier J. Ibáñez González y Jorge Sastre Martínez toda la ayuda y apoyo que me han ofrecido durante todo este tiempo, así como la paciencia que han desplegado esperando la finalización de este manuscrito. Gracias, Javier. Gracias, Jorge.

Gracias también a los demás compañeros de investigación que me han acompañado en estos últimos años: Jesús, Mike y, en especial, Emilio, que me ha animado incansablemente a terminar esta tesis de todas las maneras imaginables. Gracias a todos.

Quiero hacer una mención especial a Vicente Hernández, de cuya mano me incorporé hace ya muchos años al mundo de la investigación cuando me invitó a formar parte del grupo de Computación Paralela (actualmente, Grupo de Redes y Computación de Altas Prestaciones). Gracias, Vicente. En el seno de dicho grupo comencé mi andadura en el mundo de la docencia y la investigación, y allí he conocido a muchos compañeros que me han ido acompañando y ayudando siempre que los he necesitado: David, Fernando, Josemi, Nacho, Jose, Kike, Damián, Germán, Carlos, Miguel, etc. Muchas gracias a todos.

Por último, y fuera del ámbito académico, quiero agradecer a mi esposa, Ivana, todo el apoyo que he recibido siempre de su parte; a mis dos pequeños amores, Marc y Laia, por todas las alegrías que me dan, todas las risas que me regalan y las horas de sueño que me quitan. Gracias también a todos los buenos amigos que me acompañan en la vida, especialmente a La Sorpresa, sin cuya existencia la vida sería mucho menos divertida.

A esos ángeles llamados Marc y Laia,
a esa reina llamada Ivana,
y a mis padres, por siempre...

Índice general

1. Introducción y objetivos **7**

1.1. Objetivos 7

1.2. Resumen del estado del arte 8

1.2.1. Función exponencial 10

1.2.1.1. Polinomios matriciales ortogonales 11

1.2.1.2. Aproximaciones racionales 13

1.2.1.3. Otros métodos 15

1.2.2. Funciones seno y coseno 17

1.2.2.1. Aproximantes de Padé 17

1.2.2.2. Series de polinomios matriciales 19

1.2.3. *Software* para el cálculo de funciones matriciales 21

1.3. Organización de la tesis 22

Bibliografía 23

2. Exponencial de una matriz **27**

2.1. Introducción 27

2.2. Efficient orthogonal matrix polynomial based method for computing matrix exponential 29

2.2.1. Introduction 29

2.2.2. Hermite matrix polynomial series expansions of matrix exponential . 30

2.2.3. Error analysis. 31

2.2.4. Numerical examples 39

2.2.5. Conclusions 44

2.3. Accurate matrix exponential computation to solve coupled differential models in Engineering 46

2.3.1. Introduction 46

2.3.2. Error analysis and algorithm . . . 47
2.3.2.1. Taylor matrix polynomial evaluation . . . 48
2.3.2.2. Scaling algorithm . . . 48
2.3.3. Numerical experiments and conclusions . . . 51
2.4. New scaling-squaring Taylor algorithms for computing the matrix exponential 54
2.4.1. Introduction . . . 54
2.4.2. State of the art . . . 55
2.4.2.1. Taylor and Padé series . . . 55
2.4.2.2. Details about Taylor series approximants . . . 55
2.4.2.3. Relations between the order and the computational effort . 56
2.4.2.4. Maximizing the order . . . 58
2.4.2.5. Estimating the total cost . . . 59
2.4.3. Proposed modifications . . . 60
2.4.3.1. Modification of the choices for order and scaling . . . 61
2.4.3.2. Neglecting higher-order terms of the Taylor polynomial . . . 63
2.4.4. Numerical experiments . . . 67
2.4.4.1. Case Study 1 . . . 67
2.4.4.2. Case Study 2 . . . 69
2.4.5. Conclusions . . . 71
2.5. Accurate and efficient matrix exponential computation . . . 72
2.5.1. Introduction . . . 72
2.5.2. Taylor Algorithm . . . 73
2.5.2.1. Roundoff error analysis . . . 75
2.5.2.2. Analysis of truncation error . . . 76
2.5.2.3. Scaling algorithm . . . 79
2.5.2.4. Calculation of initial scaling s_0 . . . 80
2.5.2.5. Scaling refinement . . . 81
2.5.2.6. New bounds for $\|g_{m+1}(2^s A)\|$ and $\|h_{m+1}(2^{-s} A)\|$ . . . 83
2.5.3. Numerical experiments and conclusions . . . 84
2.6. High perfomance computing of the matrix exponential . . . 89
2.6.1. Introduction . . . 89
2.6.2. Taylor algorithm . . . 90
2.6.3. Error analysis . . . 92
2.6.4. New Taylor algorithm . . . 93

2.6.4.1. New scaling algorithm . . . 93
2.6.4.2. Taylor algorithm . . . 96
2.6.5. Numerical experiments and conclusions . . . 97
2.6.6. Conclusions . . . 102
Bibliografía . . . 103

3. Coseno y seno de una matriz **107**
3.1. Introducción . . . 107
3.2. Computing matrix functions solving coupled differential models . . . 109
3.2.1. Introduction . . . 109
3.2.2. Hermite matrix polynomials series expansions of matrix sine and matrix cosine . . . 110
3.2.3. Accurate and error bounds for cosine and sine approximation. Algorithm 112
3.2.4. Numerical examples . . . 114
3.2.5. Conclusions . . . 117
3.3. Computing matrix functions arising in engineering models with orthogonal matrix polynomials . . . 120
3.3.1. Introduction . . . 120
3.3.2. Hermite matrix polynomial series expansions of matrix cosine. Error bound . . . 122
3.3.3. Algorithm . . . 123
3.3.4. Numerical examples. . . . 125
3.3.5. Conclusions. . . . 127
3.4. Efficient computation of the matrix cosine . . . 128
3.4.1. Introduction . . . 128
3.4.2. Matrix polynomial computation by Paterson-Stockmeyer's method . 129
3.4.3. General Algorithm . . . 130
3.4.4. Error analysis in exact arithmetic and practical considerations . . . 131
3.4.5. Scaling algorithm . . . 134
3.4.5.1. Initial value of the scaling parameter . . . 135
3.4.5.2. Refinement of the scaling parameter . . . 136
3.4.6. Rounding error analysis . . . 140
3.4.7. Numerical experiments . . . 141
3.4.8. Conclusions . . . 143
Bibliografía . . . 145

4. Aplicaciones **147**

4.1. Introducción 147

4.2. Solving Initial Value Problems for Ordinary Differential Equations by two approaches: BDF and Piecewise-linearized Methods 148

4.2.1. Introduction 148

4.2.2. A BDF algorithm 149

4.2.3. A piecewise-linearized approach for solving IVPs for ODEs 152

4.2.3.1. Algorithms based on the scaling and squaring technique . . 155

4.2.3.2. Algorithms not based on scale-squaring technique 164

4.2.4. Experimental results 168

4.2.4.1. Case study 1 (Chemical Akzo Nobel problem) 169

4.2.4.2. Case study 2 (HIRES problem) 172

4.2.4.3. Case study 3 174

4.2.4.4. Case study 4 176

4.2.4.5. Case study 5 (Medical Akzo Nobel problem) 178

4.2.5. Conclusions and future work 179

4.3. A Piecewise-linearized Algorithm based on Krylov Subspace for solving stiff ODEs 181

4.3.1. Introduction 181

4.3.2. A piecewise-linearized algorithm for solving ODEs based on the Krylov subspace approach 182

4.3.3. Experimental results 183

4.3.3.1. Case study 1 (the pollution problem [LdS98]) 186

4.3.3.2. Case study 2 (the EMEP problem [LdS98]) 186

4.3.3.3. Case study 3 (the Medical Akzo Nobel problem [LdS98]) . . 188

4.3.3.4. Case study 4 (the Brusselator problem) [HW96, pp. 6] . . . 189

4.3.4. Conclusions and future work 189

Bibliografía 193

5. Conclusiones **195**

Índice de tablas

2.1. Maximal values θ_m^∞ of $\|2^{-s}A\|$ such that the backward error bound (2.21) does not exceed $u = 2^{-53}$ for $|\lambda| \to \infty$, i.e. Taylor expansion of matrix exponential. 34

2.2. Maximal values θ_m of $\|2^{-s}A\|$ such that the backward error bound (2.21) does not exceed $u = 2^{-53}$ and values of λ for which this is accomplished. 34

2.3. Evaluation of $h_m(\lambda, A)$, where $F_0^n = e^{\frac{1}{\lambda^2}} E_0^n$. Set $e^{\frac{1}{\lambda^2}} = 1$ and $F_0^n = 1$ to obtain the Taylor approximation. 34

2.4. Number of matrix multiplications, π_m. Measure of overall cost C_m. 35

2.5. Theoretical optimal values of θ_m for Hermite method (m = order of approximation, π_m = number of matrix products). For $m \leq 20$, $\theta_m = \theta_m^\infty$, i.e. the values relating to Taylor approximation. For $m = 25, 30$, θ_m have been obtained for the Hermite approximation, with the corresponding optimal values of λ. Parameter comparison with Padé `expm` method. 36

2.6. Comparison of maximum theoretical cost C_m^H in terms of matrix products for `dgeexfhrp` with maximum order $m = 30$, and cost C_m^P for `expm`, for $\|A\| \leq 5.37$. 42

2.7. Relative error comparison (%) between `dgeexfher` and `dgeexfhrp`. 42

2.8. Comparison of relative error and total number of matrix products (%) between `dgeexfhrp` and `expm`. The error comparison results were exactly the same with `dgeexfher` at a greater cost (see last table row). 43

2.9. Relative error comparison between `dgeexfhrp` and `funm`. The results were exactly the same with `dgeexfher`. 43

2.10. Maximal values $\Theta_m = \|2^{-s}A\|$ such that $\tilde{h}_{m+1}(\Theta_m) \leq \max\{1, \Theta_m\}u$, coefficient ratios $c_k^{(m)}/c_{m+2}^{(m)}$ for the first values of $k \geq m+1$, and values $u/|c_{m+2}^{(m)}|$. 49

2.11. Cost in terms of total number of matrix product evaluations (P) and relative error comparison between `exptayns`, `expm` and `expm_new`. 52

2.12. Comparison of Padé and Taylor approximations. 56

2.13. k, m_k, q_k, r_k, Θ_{m_k}, and Π_{m_k} 58

2.14. Overall cost (2.66) as a function of Taylor order m. 60

2.15. Matrix products needed when using options 0, 1, and 2 in Algorithm **Order-scale-2** for $K \geq 7$ to approximate $\exp(A)$ when $\|A\|/2^{\hat{s}}$ lies in the intervals shown, where $\hat{s}$ is defined by (2.67). 62

2.16. k, m_k, Θ_{m_k}, Θ'_{m_k}, and ϑ_{m_k} 65

2.17. Total number of matrix products P computed by each function to evaluate the exponentials of all the test matrices from Case Study 1. 68

2.18. Total number of matrix products P computed by each function to evaluate the exponentials of all 50×50 matrices from Case Study 2. 69

2.19. Total number of matrix products P to compute the exponential of all 500×500 matrices from Case Study 2. Mean and standard deviation of total execution time t in seconds for 100 repetitions of the experiment, with $m_K = 30$ in Taylor functions, and the optimal maximum order $m = 13$ in **expm**; see [Hig05]. . . 71

2.20. Values of q_k depending on the selection of m_M. 75

2.21. Maximal values Θ_m such that $\tilde{h}_{m+1}(\Theta_m) \leq \Theta_m u$, maximal values $\tilde{\Theta}_m$ such that $\tilde{g}_{m+1}\left(\tilde{\Theta}_m\right) \leq \phi(m,n)u$ for $\phi(m,n) = 1$, and values $\vartheta_m = \max\left\{\Theta_m, \tilde{\Theta}_m\right\}$. 79

2.22. Comparison of functions **exptaynsv2** (labelled 2) and **exptayns** (labelled 1) with maximum order $m_M = 30$: Maximum and minimum values of the matrix norm for each matrix set, maximum and minimum relative error ratios $\mathrm{E}_2/\mathrm{E}_1$, maximum and minimum relative error ratio E_1/u where u is the unit roundoff, number N of matrices where **exptaynsv2** cost is one matrix product lower than **exptayns**. 85

2.23. Values of q_k depending on the selection of m_M. 92

2.24. Maximal values $\Theta_m = \|2^{-s}A\|$ such that $\tilde{h}_{m+1}(\Theta_m) \leq \max\{1, \Theta_m\}u$, coefficient ratios $c_{m+1}^{(m)}/c_{m+2}^{(m)}$, values $u/|c_{m+2}^{(m)}|$, maximal values $\hat{\Theta}_m$ using only the first 2 terms in series $\tilde{h}_{m+1}(\hat{\Theta}_m)$, and values $\tilde{h}_{m+1}(\hat{\Theta}_m)/(\max\{1, \hat{\Theta}_m\}u)$ considering 200 series terms. 93

2.25. Execution time comparison in seconds between the functions `exptayns`, `exptaynsv2` and `exptaynsv3` in Matlab. 101

3.1. Values of q_k depending on the selection of m_M. 131

3.2. Highest values Θ_{m_k} such that $\sum_{i=m_k+1}^{\infty} \frac{\theta^i}{(2i)!} \leqslant u$. 134

3.3. Selection of initial scaling $s_0^{(k)}$ and order $m \in \mathbb{M}$, $m \leqslant m_M$ using only the first part of the proposed scaling algorithm, described in Subsection 3.4.5.1, depending on the values of $\beta_{min}^{(m)}$, for $m_M = 9, 12, 16, 20$. Total cost, denoted by C_m^T, is also presented. The cost with $m_M = 12$ and 9 is the same and it is presented in one column. $\beta_{min}^{(1)}$ and $\beta_{min}^{(2)}$ are not calculated for first orders $m = 1$ and 2, using $\beta_1^{(1)}$ and $\beta_2^{(2)}$ instead. The tests are done from top to bottom: If the condition for current row is not verified then we test the condition for the next row. In last four rows i can take the values $i = 0, 1, \ldots$ 137

3.4. Values of bound (3.83). 139

4.1. BDF method parameters (order r=1, 2, 3, 4 and 5). 152

4.2. Relative errors considering t_f=60 and varying Δt (case study 1) 170

4.3. Execution time of the MATLAB implementations considering t_f=60 and varying Δt (case study 1) 171
4.4. Relative errors considering Δt=0.01 and varying t_f (case study 1) 171
4.5. Relative errors considering t_f=50 and varying Δt (case study 2) 172
4.6. Execution time of the MATLAB implementations considering t_f=50 and varying Δt (case study 2) 173
4.7. Relative errors considering Δt=0.01 and varying t_f (case study 2) 173
4.8. Relative errors considering t_f=100 and varying Δt (case study 3) 175
4.9. Execution time of the MATLAB implementations considering t_f=100 and varying Δt (case study 3) 175
4.10. Relative errors considering Δt=0.01 and varying t_f (case study 3) 175
4.11. Relative errors considering t_f=10 and varying Δt (case study 4) 176
4.12. Execution time the MATLAB implementations for t_f=10 and Δt (case study 4) 177
4.13. Relative errors considering Δt=0.1 and varying t_f (case study 4) 177
4.14. Comparative precision of implemented algorithms for the five case studies: The symbols +, − and ≅ indicate respectively greater, less and similar precision. A/NA denotes Autonomous/Non-Autonomous ODEs, and S/NS denotes whether the problem is Stiff or Non-Stiff. 180
4.15. Comparative execution times of the implemented algorithms for the five case studies: The symbols +, − indicate respectively greater and less execution time. A/NA denotes Autonomous/Non-Autonomous ODEs, and S/NS denotes whether the problem is Stiff or Non-Stiff. 180
4.16. Relative error (E_r) with t = 10 and Δt variable (case study 1). 186
4.17. Execution time (T_e) in seconds with t = 10 and Δt variable (case study 1). . 186
4.18. Relative error (E_r) Δt = 0.01 and t variable (case study 1). 186
4.19. Relative error (E_r) with Δt = 0.1 and t variable (case study 2). 187
4.20. Execution time (T_e) in seconds with Δt = 0.1 and t variable (case study 2). . 187
4.21. Relative error (E_r) considering n = 100, t = 1 and Δt variable (case study 3). 188
4.22. Execution time (T_e) in seconds considering n = 100, t = 1 and Δt variable (case study 3). 188
4.23. Relative error (E_r) considering Δt = 0.001, t = 1 and n variable (case study 3). 189
4.24. Execution time (T_e) in seconds considering Δt = 0.001, t = 1 and n variable (case study 3). 189
4.25. Relative error (E_r) considering n = 100, t = 1 and Δt variable (case study 4). 189
4.26. Execution time (T_e) in seconds considering n = 100, t = 1 and Δt variable (case study 4). 190
4.27. Relative error (E_r) considering Δt = 0.001, t = 1 and n variable (case study 4). 190

4.28. Execution time (T_e) in seconds considering $\Delta t = 0.001$, $t = 1$ and n variable (case study 4). 190

Índice de figuras

2.1. Bound (2.21) for $m = 22, 23, \ldots, 33$ and $\theta = \theta_m^\infty$, vs. parameter λ. 34

2.2. Comparative of `dgeexfhrp`, `expm` and `funm`. 44

2.3. (a) Performance profile ($m_M = 16, 20, 25, 30$). (b) Ratio of relative errors $\mathrm{E}_{\texttt{expm\_new}}/\mathrm{E}_{\texttt{exptayns}}$ with Taylor maximum orders $m_M = 16$ and 30. 53

2.4. Comparison results, Case Study 1. 68

2.5. Comparison results, Case Study 2. 70

2.6. Performance profile and cost in terms of matrix products for the 128×128 diagonalizable and 256×256 Jordan matrices from sets 1 and 2, and $m_M = 30$ in Taylor functions. 86

2.7. Relative error ratios E_2/E_1 for functions **exptayns** (E_1) and **exptaynsv2** (E_2), performance profile and cost in terms of matrix products, for $m_M = 30$ and the 1000×1000 matrices from the Matrix Computation Toolbox. 87

2.8. Relative error ratios E_2/E_1 for functions **exptayns** (E_1), **exptaynsv2** (E_2), performance profiles and cost in terms of matrix products, for $m_M = 30$ and matrices with sizes 25 and 100 from the Matrix Computation Toolbox multiplied by constants t_i, see text. 88

2.9. $\{\|A^k\|_2^{1/k}\}_{k=1}^{61}$ for 103 matrices A with $\|A\|_2 = 1$ and sizes from 2×2 to 100×100. 95

2.10. Comparison of the cost in terms of matrix products (P) and the relative error (E) between `exptaynsv3` (P_3 and E_3) and `exptayns` (P_1 and E_1) with matrix sets 1 and 2 (a), and matrix set 3 (b). 99

2.11. Comparison of the cost in terms of matrix products (P) and the relative error (E) between `exptaynsv3` (P_3 and E_3) and `exptaynsv2` (P_1 and E_1) with matrix sets 1 and 2 (a), and matrix set 3 (b). 99

2.12. Relative error comparison between `exptaynsv3` and `exptaynsv2` with matrix sets 1 and 2 (a) and matrix set 3 (b). Taylor maximum order $m_M = 30$. . . . 100

2.13. Relative error comparison between `exptaynsv3` and `expm_new` with matrix sets 1 and 2 (a) and matrix set 3 (b). Taylor maximum order $m_M = 30$. . . . 100

2.14. Cost in seconds (T) (a) and relative error comparison (E) (b) between `exptaynsv3` (T_3 and E_3) and `f01ecc` from NAG Library (T_N and E_N) with matrix sets 2 and 3. 101

2.15. Relative error comparison between `exptaynsv3` and function `f01ecc` from NAG Software. 102

3.1. For $N = 8$ fixed and varying λ. 115

3.2. Relative error of Hermite series (3.11) for example 3.2.1 for $\lambda = 4.1$. 115

3.3. Comparison between the relative errors for cosine and sine computation with $N = 20$ and $\lambda = 0.7936$. 118

3.4. Comparison between the relative errors for cosine and sine computation with $N = 25$ and $\lambda = 0.6175$. 119

3.5. Tables of $\lambda_{\text{mín}}$ and Θ_N. 125

3.6. Comparatives `cosher-cosm` and `cosher-cosm`. The first three rows show the percentage of times that relative error of `cosher` is lower (L), equal (E) or greater (G) than relative error of `cosm` or `funm`. The last row shows the ratio (R) of costs in terms of matrix products between `cosher` and `cosm` (3.6a), and `cosher` and `funm` (3.6b). 126

3.7. Performance profile of `cosher`, `cosm` and `funm` for the set of test matrices. . 127

3.8. Comparatives `costay-cosm` and `costay-cosher`. The first three rows show the percentage of times that relative error of `costay` is lower (L), equal (E) or greater (G) than relative error of `cosm` or `cosher`. The last row shows the ratio (R) of cost in terms of matrix products between `costay` divided by the cost of `cosm` in 3.8a, and `cosher` in 3.8b. 142

3.9. Normwise relative errors and perfomance profile of `cosm`, `cosher(16)` and `costay` for $m_M = 12, 16, 20$. 143

4.1. Schema of piecewise-linearized Algorithms 4.4($inolsp$), 4.9($inolwp$), 4.7($iaolsp$) and 4.10($iaolwp$) 167

4.2. Execution time of the MATLAB implementations considering $\Delta t = 0.01$ and varying t_f (case study 1) 171

4.3. Execution time of the MATLAB implementations considering $\Delta t = 0.01$ and varying t_f (case study 2) 173

4.4. Execution time of the MATLAB implementations considering $\Delta t = 0.01$ and varying t_f (case study 3) 175

4.5. Execution time of the MATLAB implementations considering $\Delta t = 0.1$ and varying t_f (case study 4) 177

4.6. Execution time of the Fortran implementations considering $\Delta t = 10^{-6}$ and varying t_f (case study 5) 179

4.7. Execution time in seconds of the MATLAB implementations considering $\Delta t = 0.01$ and varying t (case study 1). 187

4.8. Execution time in seconds of the MATLAB implementations considering $\Delta t = 0.1$ and varying t between 15400 and 19400 (case study 2). 188

4.9. Execution time in seconds of the MATLAB implementations considering Δt = 0.001 and varying t between 50 and 250 (case study 4). 190

Summary

The aim of this thesis is the development of high performance computing (HPC) innovative algorithms and implementations for computing matrix functions based on matrix polynomials series. Specifically, algorithms for the calculation of the most commonly-used functions, the exponential, sine and cosine have been developed.

The study of orthogonal matrix polynomials is an emerging field whose growth is achieving important results both theoretically and practically. The last investigations made by the doctoral student, together with the members of the research group, High Performance Scientific Computing (HiPerSC), he is linked, reveal why the matrix polynomials play a fundamental role in the approximation of matrix functions, providing very interesting properties. In this thesis new high-performance algorithms based on matrix polynomial series have been developed. In particular, algorithms for computing the exponential, sine and cosine of a matrix using Taylor and Hermite matrix polynomial series have been implemented. In addition, the error bounds for the approximations calculated have been provided and optimal theoretical and experimental parameters for such approximations have also been provided. Final algorithms have been compared to other state of the art implementations to test the improvement obtained in terms of efficiency and performance.

The results obtained during the investigation and presented in this memory have been published in several high-level journals and presented as papers at various editions of the International Congress Mathematical Modelling in Engineering & Human Behaviour to give them the widest possible distribution. On the other hand, implemented computer codes have been made freely available to the international scientific community at our web page http://hipersc.blogs.upv.es.

Resumen

El objetivo de esta tesis es el desarrollo de algoritmos e implementaciones innovadoras de altas prestaciones (HPC) para la computación de funciones de matrices basadas en series de polinomios matriciales. En concreto, se desarrollarán algoritmos para el cálculo de las funciones matriciales más utilizadas: la exponencial, el seno y el coseno.

El estudio de los polinomios ortogonales matriciales es un campo emergente cuyo avance está alcanzando importantes resultados tanto desde el punto de vista teórico como práctico. Las últimas investigaciones realizadas por el doctorando, junto a los miembros del grupo de investigación al que está vinculado, High Performance Scientific Computing (HiPerSC), revelan por qué los polinomios matriciales desempeñan un papel fundamental en la aproximación de funciones de matrices, proporcionando propiedades muy interesantes. En esta tesis se han desarrollado nuevos algoritmos de alto rendimiento basados en series polinomiales matriciales. En particular, se han implementado algoritmos para el cálculo de la exponencial, el seno y el coseno de una matriz usando las series matriciales polinomiales de Taylor y de Hermite. Además, se han proporcionado cotas del error cometido en las aproximaciones calculadas, proporcionando además los parámetros teóricos y experimentales óptimos de dichas aproximaciones. Los algoritmos finales han sido comparados con otras implementaciones del estado del arte para comprobar la mejora que presentan en cuanto a eficiencia y prestaciones.

Los resultados obtenidos a lo largo de la investigación y presentados en esta memoria han sido publicados en varias revistas de alto nivel y se han presentado como ponencias en diversas ediciones del congreso internacional Mathematical Modelling in Engineering & Human Behaviour para dotarlas de la mayor difusión posible. Por otra parte, los códigos informáticos implementados han sido puestos a disposición de la comunidad científica internacional a través de nuestra página web http://hipersc.blogs.upv.es.

Resum

L'objectiu d'aquesta Tesi és el desenvolupament d'algoritmes i implementacions innovadores d'altes prestacions (HPC) per a la computació de funcions de matrius basades en sèries de polinomis matricials. En concret, es desenvoluparan algoritmes per al càlcul de les funcions matricials més emprades: l'exponencial, el sinus i el cosinus.

L'estudi dels polinomis ortogonals matricials és un camp emergent, el creixement del qual està aconseguint importants resultats tant des del punt de vista teòric com pràctic. Les últimes investigacions realitzades pel doctorand junt amb els membres del grup d'investigació on està vinculat, High Performance Scientific Computing (HiPerSC), revelen per què els polinomis matricials exerceixen un paper fonamental en l'aproximació de funcions de matrius, proporcionant propietats molt interessants. En aquesta Tesi s'han desenvolupat nous algoritmes d'alt rendiment basats en sèries polinomials matricials. En particular, s'han implementat algoritmes per al càlcul de l'exponencial, el sinus i el cosinus d'una matriu usant les sèries matricials polinomials de Taylor i d'Hermite. A més, s'han proporcionat cotes de l'error comès en les aproximacions calculades, proporcionant a més els paràmetres teòrics i experimentals òptims d'aquestes aproximacions. Els algoritmes finals han estat comparats amb altres implementacions de l'estat de l'art per a provar la millora que presenten en termes d'eficiència i prestacions.

Els resultats obtinguts al llarg de la investigació i presentats en aquesta memòria han estat publicats en diverses revistes d'alt nivell i s'han presentat com a ponències en diferents edicions del congrés internacional Mathematical Modelling in Engineering & Human Behaviour per a dotar-les de la major difusió possible. D'altra banda, s'han posat els codis informàtics implementats a disposició de la Comunitat Científica Internacional mitjançant la nostra pàgina web http://hipersc.blogs.upv.es.

Capítulo 1

Introducción y objetivos

En este capítulo se explican los objetivos buscados en las labores de investigación llevadas a cabo durante la realización de esta tesis. Además, se presenta un resumen del estado del arte relativo al cálculo de funciones matriciales y, por último, se hace una descripción de cómo está organizada esta memoria.

1.1. Objetivos

El objetivo principal de esta tesis ha sido el desarrollo de nuevos algoritmos de altas prestaciones para el cálculo de funciones matriciales. Este campo ha suscitado bastante interés y ha recibido un gran impulso en los últimos años debido a su aplicación en numerosas áreas de la Ingeniería, con nuevas aplicaciones y abundante bibliografía [SIDR15, SIDR14, Sas12, Hig08, AMH09, Hig05, HH05, AMHR15, ML03]. Funciones como la exponencial, el seno, el coseno o el logaritmo aparecen en la resolución de sistemas de ecuaciones diferenciales de primer y segundo orden asociados a multitud de aplicaciones. También aparecen en numerosos problemas de Ingeniería, como la resolución de modelos de Markov, en Teoría de Control, Circuitos, Resonancia Nuclear Magnética o en el campo de procesado de imagen y vídeo, por poner algunos ejemplos. Tanto el coste computacional como el error a la hora de calcular las funciones matriciales pueden ser elevados, especialmente cuando trabajamos con matrices de gran dimensión, por lo que se hace patente la necesidad de contar con métodos y algoritmos que nos aseguren una buena precisión con el coste más bajo posible. Tradicionalmente se ha considerado que los métodos basados en aproximaciones racionales, como Padé o Chebyshev, eran más adecuados para resolver estos problemas que los métodos basados en polinomios matriciales ortogonales, como las series de Taylor o Hermite, por ejemplo. Sin embargo, a lo largo de nuestra investigación hemos desarrollado métodos basados en polinomios matriciales con excelentes resultados que superan a los algoritmos del estado del arte, como se podrá ver en los artículos incluidos en este trabajo.

En lo que respecta a esta tesis, nos hemos centrado en tres de las funciones matriciales más utilizadas, como son las funciones exponencial, seno y coseno. En la siguiente sección se

realiza un detallado estado del arte de dichas funciones, incluyendo algunas de sus posibles aplicaciones, así como los métodos más utilizados para su cálculo.

1.2. Resumen del estado del arte

A lo largo de esta sección, $\mathbb{C}^{n\times n}$ denota el conjunto de matrices complejas de tamaño $n\times n$, I denota la matriz identidad para dicho conjunto, $\|\cdot\|$ indica cualquier norma matricial subordinada y $\sigma(A)$ denota el espectro de la matriz A, es decir, el conjunto de todos sus valores propios.

De manera informal, dada una matriz cuadrada A y una función compleja de variable compleja $f(z)$ definida sobre el espectro de A, entonces la matriz $f(A)$ se puede obtener sustituyendo la variable z por la matriz A en la expresión que define a $f(z)$. Formalmente, sin embargo, existen diversas formas equivalentes de definir $f(A)$. Posiblemente la más elegante de ellas sea la que presentan Golub y Van Loan en [GL96, capítulo 1]:

Definición 1.1 *Sea A una matriz cuadrada de dimensión n definida en el conjunto de los números complejos $\mathbb{C}$ y $f(z)$ una función analítica definida en un abierto $U \subset \mathbb{C}$ que contiene el espectro de A. Si $\Gamma \subset U$ es una curva rectificable que rodea al espectro de A, entonces la matriz $f(A)$ se define como*

$$f(A) = \frac{1}{2\pi i} \oint_\Gamma f(z)(I_n z - A)^{-1} dz.$$

De este modo, el elemento (k, j) de la matriz $f(A)$ vendría dado por la expresión

$$f_{kj} = \frac{1}{2\pi i} \oint_\Gamma f(z) e_k^T (I_n z - A)^{-1} e_j dz,$$

donde e_k y e_j xon la k-ésima y la j-ésima columna de la matriz identidad I_n, respectivamente.

Esta definición proviene del teorema integral de Cauchy, por lo que se trata de una definición independiente de la curva Γ.

A partir de esta definición se pueden demostrar propiedades, calcular y definir de otras formas funciones de matrices:

Propiedad 1.1 *Sea $A \in \mathbb{C}^{n\times n}$ y $f(z)$ una función analítica definida en un abierto $U \subset \mathbb{C}$ que contiene el espectro de A, entonces A y $f(A)$ verifican la ecuación conmutante*

$$Af(A) = f(A)A.$$

Propiedad 1.2 *Sea $A \in \mathbb{C}^{n\times n}$ y $f(z)$ una función analítica definida en un abierto $U \subset \mathbb{C}$ que contiene el espectro de A, entonces A y $f(A)$ verifican que*

$$f(A^T) = f(A)^T.$$

Propiedad 1.3 *Sea $A \in \mathbb{C}^{n\times n}$ y $f(z)$ una función analítica definida en un abierto $U \subset \mathbb{C}$ que contiene el espectro de A. Si $A = XBX^{-1}$, entonces*

$$f(A) = Xf(B)X^{-1}.$$

Propiedad 1.4 *Sea $A \in \mathbb{C}^{n\times n}$ y $f(z)$ una función analítica definida en un abierto $U \subset \mathbb{C}$ que contiene el espectro de A. Si X conmuta con A, entonces X conmuta con $f(A)$.*

Propiedad 1.5 *Sea $A \in \mathbb{C}^{n\times n}$ y $f(z)$ una función analítica definida en un abierto $U \subset \mathbb{C}$ que contiene el espectro de A. Si $A = (A_{ij})$ es una matriz triangular a bloques, entonces $F = f(A)$ es triangular a bloques con la misma estructura que A y $F_{ii} = f(A_{ii})$.*

Propiedad 1.6 *Sea $A \in \mathbb{C}^{n\times n}$ y $f(z)$ una función analítica definida en un abierto $U \subset \mathbb{C}$ que contiene el espectro de A. Si A es una matriz diagonal a bloques, $A = \operatorname{diag}(A_1, A_2, \cdots, A_r)$, con $A_i \in \mathbb{C}^{n_i\times n_i}$, $i = 1, 2, \cdots, r$, $\sum_{i=1}^{r} n_i = n$, entonces*

$$f(A) = \operatorname{diag}(f(A_1), f(A_2), \cdots, f(A_r)).$$

Otra manera de definir las funciones de matrices, muy utilizada, está basada en la forma canónica de Jordan [Hig08]:

Definición 1.2 *Sea $A \in \mathbb{C}^{n\times n}$ y $f(z)$ una función analítica definida en un abierto $U \subset \mathbb{C}$ que contiene el espectro de A. Si*

$$A = XJX^{-1}$$

es la descomposición canónica de Jordan de la matriz A, donde $J = \operatorname{diag}(J_{\lambda_1}, J_{\lambda_2}, \cdots, J_{\lambda_r})$,

$$J_{\lambda_i} = \begin{bmatrix} \lambda_i & 1 & 0 & \cdots & 0 \\ 0 & \lambda_i & 1 & \vdots & \vdots \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & \cdots & \ddots & \lambda_i & 1 \\ 0 & \cdots & \cdots & 0 & \lambda_i \end{bmatrix} \in \mathbb{C}^{n_i\times n_i},\ i = 1, 2, \cdots, r,$$

siendo J no singular y $n_1 + n_2 + \cdots + n_r = n$, entonces

$$f(A) = X\operatorname{diag}(f(J_{\lambda_1}), f(J_{\lambda_2}), \cdots, f(J_{\lambda_r}))X^{-1},$$

donde

$$f(J_{\lambda_i}) = \begin{bmatrix} f(\lambda_i) & f^{(1)}(\lambda_i) & \cdots & \dfrac{f^{(n_i-2)}(\lambda_i)}{(n_i-2)!} & \dfrac{f^{(n_i-1)}(\lambda_i)}{(n_i-1)!} \\ 0 & f(\lambda_i) & \cdots & \dfrac{f^{(n_i-3)}(\lambda_i)}{(n_i-3)!} & \dfrac{f^{(n_i-2)}(\lambda_i)}{(n_i-2)!} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \cdots & f(\lambda_i) & f^{(1)}(\lambda_i) \\ 0 & \cdots & \cdots & 0 & f(\lambda_i) \end{bmatrix} \in \mathbb{C}^{n_i\times n_i}.$$

Finalmente, una herramienta básica para aproximar funciones de matrices son las series matriciales de potencias, especialmente las series de Taylor. El siguiente teorema garantiza la convergencia de la serie matricial de Taylor cuando los valores propios entran dentro del rango de convergencia de la serie escalar de Taylor asociada [Hig08, Teorema 4.7].

Teorema 1.1 *Supongamos una función f con la siguiente expansión en series de Taylor*

$$f(z) = \sum_{k=0}^{\infty} c_k (z-\alpha)^k, \quad c_k = \frac{f^{(k)}(\alpha)}{k!},$$

con radio de convergencia r. Dada una matriz $A \in \mathbb{C}^{n\times n}$, entonces $f(A)$ viene dada por la expresión

$$f(A) = \sum_{k=0}^{\infty} c_k (A-\alpha I)^k$$

si y solo si los valores propios de A, $\lambda_1, \ldots, \lambda_s$, satisfacen una de las siguientes condiciciones

- $|\lambda_i - \alpha| < r$,
- $|\lambda_i - \alpha| = r$ *y la serie para* $f^{(n_i-1)}(\lambda)$, *donde* n_i *es el índice de* λ_i, *es convergente en el punto* $\lambda = \lambda_i$, $i = 1, \ldots, s$.

De las distintas definiciones que hemos visto, las más útiles desde un punto de vista computacional son las basadas en la forma canónica de Jordan de A y en series matriciales de potencias, ya que permiten la implementación de algoritmos eficientes y numéricamente estables.

1.2.1. Función exponencial

La función exponencial ha motivado numerosas y variadas aproximaciones, recopiladas y revisadas en [ML03, Hig08], debido a su relación con la resolución de Sistemas de Ecuaciones Diferenciales (SED) de primer orden. Estos sistemas aparecen en la resolución de numerosos modelos asociados a fenómenos físicos, químicos, biológicos, etc. [Kar59, Var62]. Así, por ejemplo, el modelado de fenómenos físicos mediante SEDs de primer orden de la forma

$$Y' = AY,\ Y(0) = y_0,\ A \in \mathbb{C}^{n\times n},\ y_0 \in \mathbb{C}^n, \tag{1.1}$$

donde A es una matriz constante e y_0 un vector, surge en numerosas áreas [Kar59, Lau85]. La solución de este problema puede escribirse en términos de la exponencial de una matriz e^{At} [Kar59]:

$$Y(t) = e^{At} y_0.$$

No obstante, la aplicación de la exponencial de una matriz no sólo se circunscribe a la resolución de (1.1), sino que también aparece en numerosos problemas de Ingeniería como la Resonancia Nuclear Magnética, la resolución de modelos de Markov (junto con la función logaritmo) o en Teoría de Control [Hig08]. Por ejemplo, una de las aplicaciones más recientes en las que ha sido utilizada con éxito es en áreas relativas al procesado de vídeo como son la codificación de vídeo y el reconocimiento de objetos [WSTO11, CO09, TVSC11, MR07].

La función exponencial también puede ser utilizada para resolver la Ecuación Diferencial Matricial de Riccati [JI09, JI11], que aparece en áreas de Física aplicada e Ingeniería, como Teoría de Control, Sitemas, Mecánica, etc.:

$$X'(t) = Q(t) + X(t)A(t) + B(t)X(t) - X(t)R(t)X(t),\ t_0 \le t \le t_f,\ X(t_0) = X_0 \in \mathbb{R}^{m\times n},$$

donde $A(t) \in \mathbb{R}^{n\times n}$, $B(t) \in \mathbb{R}^{m\times m}$, $Q(t) \in \mathbb{R}^{m\times n}$, $R(t) \in \mathbb{R}^{n\times m}$ y $X(t) \in \mathbb{R}^{m\times n}$.

En cuanto a las formas de resolución, existen múltiples métodos para calcular la exponencial de una matriz [ML03, Hig08], aunque los más prometedores para ser utilizados en implementaciones de altas prestaciones se encuentran los basados en aproximaciones racionales y polinómicas. Estos métodos permiten la reorganización del código de forma que se puedan usar núcleos computacionales básicos para el cálculo eficiente de operaciones de tipo matricial. A continuación se explica más en detalle en qué consisten estos métodos.

1.2.1.1. Polinomios matriciales ortogonales

Entre las distintas series de polinomios matriciales, caben destacar en el área que nos ocupa las series de Taylor y de Hermite [JC96, DJ98]. Las series de Taylor son una herramienta básica para la aproximación de funciones de matrices. Así, la serie matricial de Taylor que nos permite aproximar la exponencial de una matriz viene dada por la expresión:

$$e^A = \sum_{k=0}^{\infty} \frac{1}{k!}A^k = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots,$$

que, puesto que converge para $z \in \mathbb{C}$, resulta ser también convergente para cualquier matriz cuadrada A (Teorema 1.1).

A la hora de obtener una aproximación lo suficientemente precisa, deberemos contar con dos tipos de errores: errores de truncamiento y errores de redondeo. Los errores de truncamiento surgen debido a la necesidad de despreciar los términos de la serie a partir de cierto orden, puesto que no es posible calcular los infinitos términos. Es fundamental establecer adecuadamente el número de términos a calcular para obtener un buen equilibrio entre precisión y eficiencia. Los errores de redondeo, por su parte, son inherentes al cálculo aritmético en coma flotante, al contar éste con una precisión finita. Es importante, en particular, evitar en lo posible el problema de la cancelación de dígitos significativos que se produce cuando se suman y restan números del mismo orden de magnitud [ML78]. Por otra parte, los errores de redondeo y los costes computacionales aumentan conforme aumenta la norma de A. Para evitar este problema se utiliza la técnica de escalado y potenciación (*scaling and squaring method*, [ML78, Fun04, Hig05]), basada en el uso de la igualdad

$$e^A = (e^{A/m})^m,$$

donde $m = 2s$, siendo s entero no negativo. De este modo, puesto que la serie de Taylor es precisa cerca del origen, se puede calcular $e^{A/m}$ con suficiente precisión al reducirse la norma de la matriz, para posteriormente obtener de manera aproximada la matriz e^A mediante sucesivas potenciaciones. El Algoritmo 1.1 muestra un algoritmo general de Taylor con escalado y potenciación para el cálculo de la exponencial de una matriz.

El preprocesado de la matriz, pasos 2–5, es opcional y se usa para reducir la norma de la matriz. Para ello, se usan dos técnicas. Por una parte, teniendo en cuenta que tanto la serie

Algorithm 1.1 Algoritmo general de Taylor para calcular la exponencial de una matriz, $A \in \mathbb{C}^{n\times n}$, usando la técnica del escalado y potenciación

1: Preprocesado de la matriz A:
2: $\mu = \text{trace}(A)/n$
3: $A \leftarrow A - \mu I$
4: Determinar una matriz diagonal D tal que $D^{-1}AD$ esté equilibrada.
5: $A \leftarrow D^{-1}AD$
6: Elección del orden de aproximación más adecuado, p, y del parámetro de escalado, s.
7: Calcular la aproximación de Taylor de orden p: $B = T_p(A/2^s)$.
8: **for** $i = 1 : s$ **do**
9: $B = B^2$
10: **end for**
11: Postprocesado de la matriz B:
12: $B \leftarrow e^{\mu} DBD^{-1}$

$\sum_{k=0}^{\infty} A^k/k!$ como la serie $e^{\mu}\sum_{k=0}^{\infty}(A-\mu I)^k/k!$ para cualquier $\mu \in \mathbb{C}$ convergen a e^A, se trata de buscar un valor de μ tal que $\|A-\mu I\| \leq \|A\|$ de forma que se acelere dicha convergencia. El Teorema 10.18 junto con el Teorema 4.21 (a) de [Hig08] sugieren que un buen valor para μ vendría dado por $\mu = \text{trace}(A)/n$, donde $\text{trace}(A)$ es la suma de los elementos de la diagonal de A, lo cual coincide con la suma de sus valores propios. No obstante, hay que tener en cuenta que esta técnica no siempre produce buenos resultados, e incluso puede provocar una pérdida de precisión en algunos casos, por lo que no es recomendable utilizarla de forma automática [Hig08]. La segunda técnica usada en el preprocesado consiste en intentar equilibrar las normas de la i–ésima fila y la i–ésima columna, con $1 \leq i \leq n$, determinando una matriz diagonal D con la que aplicar la transformación de semejanza $\tilde{A} = D^{-1}AD$. Este equilibrado es un proceso heurístico que tiende a reducir la norma, aunque en determinados casos puede provocar una pérdida significativa de precisión, por lo que es recomendable usarlo con precaución. Si optamos por realizar el preprocesado de la matriz A, tendremos que realizar el postprocesado de la matriz B para revertir todos los cambios (paso 12).

En el paso 6 se seleccionan el orden de aproximación, p, y el escalado a aplicar, s. La elección correcta de estos dos parámetros es fundamental para obtener resultados precisos con un coste computacional mínimo. Posteriormente se calcula la exponencial de la matriz escalada mediante la serie matricial de Taylor de orden p,

$$T_p(A/2^s) = \sum_{k=0}^{p} \frac{1}{k!}(A/2^s)^k.$$

Para realizar este paso de la forma más eficiente posible, se hace necesaria la utilización de algún método que permita reagrupar los términos del polinomio matricial, como el desarrollado por Paterson–Stockmeyer [PS73] o, mejor aún, el nuevo y más eficiente método desarrollado recientemente por Sastre [Sas18]. Por último, es necesario realizar s pasos de potenciación sucesivos para obtener la exponencial de la matriz original A.

En el caso de las series de Hermite, supongamos una matriz B de estabilidad positiva en $\mathbb{C}^{r\times r}$, es decir, $Re(\lambda) > 0$, para todo $\lambda \in \sigma(B)$, entonces el polinomio matricial de orden n de Hermite se define en (3.4) de [JC96, p. 25] como

$$H_n(x,B) = n! \sum_{k=0}^{[\frac{n}{2}]} \frac{(-1)^k \left(x\sqrt{2B}\right)^{n-2k}}{k!(n-2k)!}\ , \tag{1.2}$$

y de las ecuaciones (3.1) y (3.2) del mismo artículo se obtiene su función generadora

$$G(x,t) = e^{xt\sqrt{2B}-t^2 I} = \sum_{n\geq 0} H_n(x,B)t^n/n!, \quad |t| < \infty, \tag{1.3}$$

donde $x \in \mathbb{C}$ y $t \in \mathbb{C}$. Si cogemos $A = \sqrt{2B}$, $y = tx$ y $\lambda = 1/t$, entonces podemos reescribir la ecuación anterior como

$$e^{Ay} = e^{\frac{1}{\lambda^2}} \sum_{n\geq 0} \frac{1}{\lambda^n n!} H_n\left(\lambda y, \frac{1}{2}A^2\right), \ \lambda \in \mathbb{C}, \quad y \in \mathbb{C}, \quad A \in \mathbb{C}^{r\times r}, \tag{1.4}$$

sin restricciones para $\sigma(A)$. Y la ecuación (1.2), por su parte, se puede reescribir como

$$H_n(\lambda y, A^2/2) = n! \sum_{k=0}^{[\frac{n}{2}]} \frac{(-1)^k (\lambda y A)^{n-2k}}{k!(n-2k)!} \ , \tag{1.5}$$

también sin restricciones en $\sigma(A)$. Llamando $h_m(\lambda y, A)$ a la suma de los primeros m términos de la serie (1.4), tenemos

$$h_m(\lambda y, A) = e^{\frac{1}{\lambda^2}} \sum_{n=0}^{m} \frac{1}{\lambda^n n!} H_n\left(\lambda y, \frac{1}{2}A^2\right) \approx e^{Ay}, \quad \lambda, y \in \mathbb{C}, \quad A \in \mathbb{C}^{r\times r}. \tag{1.6}$$

Esta es la serie que se usa en el método para calcular la exponencial de la matriz A, donde m es el orden de aproximación [SIDR11]. A partir de (1.5) y (1.6), por inducción tenemos que

$$h_m(\lambda y, A) = \begin{cases} e^{\frac{1}{\lambda^2}}\left[E_0^{\frac{m-1}{2}}(I+Ay) + E_0^{\frac{m-1}{2}-1}\left(\frac{(Ay)^2}{2!}+\frac{(Ay)^3}{3!}\right)+\cdots\right. \\ \left.\cdots + E_0^1\left(\frac{(Ay)^{m-3}}{(m-3)!}+\frac{(Ay)^{m-2}}{(m-2)!}\right)+\frac{(Ay)^{m-1}}{(m-1)!}+\frac{(Ay)^m}{m!}\right], \quad \text{si } m \text{ impar}, \\ e^{\frac{1}{\lambda^2}}\left[E_0^{\frac{m}{2}} I + E_0^{\frac{m}{2}-1}\left(Ay+\frac{(Ay)^2}{2!}\right)+E_0^{\frac{m}{2}-2}\left(\frac{(Ay)^3}{3!}+\frac{(Ay)^4}{4!}\right)+\cdots\right. \\ \left.\cdots + E_0^1\left(\frac{(Ay)^{m-3}}{(m-3)!}+\frac{(Ay)^{m-2}}{(m-2)!}\right)+\frac{(Ay)^{m-1}}{(m-1)!}+\frac{(Ay)^m}{m!}\right], \quad \text{si } m \text{ par}, \end{cases} \tag{1.7}$$

donde

$$E_i^j = \sum_{k=i}^{j} \frac{\left(-\frac{1}{\lambda^2}\right)^k}{k!} \ , \ i,j \in \mathbb{N}, j \geq i \ . \tag{1.8}$$

Obsérvese que para $|\lambda| \to \infty$, $e^{\frac{1}{\lambda^2}} = 1$, $E_0^j = 1$, y $h_m(\lambda y, A)$ tiende a la serie de Taylor de orden m para el cálculo de e^{Ay}. Para un cálculo eficiente y preciso, será necesario ajustar adecuadamente el valor de λ para cada orden de aproximación m que vayamos a utilizar. El algoritmo necesario para obtener la exponencial de una matriz mediante las series de Hermite sería prácticamente idéntico al que hemos visto previamente para el caso de Taylor, haciendo uso también la técnica del escalado y potenciación para reducir los costes computacionales y los errores de redondeo, pero utilizando la aproximación de Hermite en lugar de la de Taylor en el paso 7, obviamente.

1.2.1.2. Aproximaciones racionales

Las aproximaciones racionales son otra potente herramienta para el cálculo de aproximaciones de funciones. Entre estas, las más utilizadas son las de Chebyshev y, especialmente,

las de Padé [ML78, ML03, Hig08, Hig05]: dada la función escalar $f(z)$, la función racional

$$r_{km}(z) = \frac{p_{km}(z)}{q_{km}(z)}$$

es un aproximante de Padé de orden $[k, m]$ de la función f si p_{km} y q_{km} son polinomios de grados, como mucho, k y m, respectivamente, $q_{km}(0) = 1$ y

$$f(z) - r_{km}(z) = O(z^{k+m+1}).$$

Esta última condición nos muestra que r_{km} reproduce los primeros $k+m+1$ términos (desde z^0 hasta $z^{(m+k)}$) de la serie de Taylor de la función f sobre el origen. Obsérvese que si $m = 0$, r_{km} coincide con la serie de Taylor de orden k.

Si el aproximante de Padé de orden $[k/m]$ existe, entonces es único. Para el caso de la función exponencial, los aproximantes de Padé de orden $[k/m]$ son conocidos para cualquier k y m:

$$p_{km}(z) = \sum_{j=0}^{k} \frac{(k+m-j)!k!z^j}{(k+m)!(k-j)!j!}, \quad q_{km}(z) = \sum_{j=0}^{m} \frac{(k+m-j)!m!(-z)^j}{(k+m)!(m-j)!j!}.$$

Así, dada una matriz $A \in \mathbb{C}^{n\times n}$, la aproximación de Padé para calcular su exponencial vendría dada por la expresión

$$e^A \approx r_{km}(A) = [q_{km}(A)]^{-1} p_{km}(A).$$

El error cometido en esta aproximación viene dado por [GL96, cap. 11]

$$e^A - r_{km}(A) = \frac{(-1)^m}{(k+m)!} A^{k+m+1} q_{km}(A)^{-1} \int_0^1 e^{tA}(1-t)^k t^m dt.$$

El problema de este método es que solo proporciona buenas aproximaciones en un entorno del origen, tal y como revela la expresión anterior. Si embargo, este problema se puede evitar haciendo uso de la técnica de escalado y potenciación comentada previamente.

Tradicionalmente se ha considerado [ML03, Hig08, GL12] que los aproximantes diagonales ($k = m$) son mejores que los no diagonales ($k \neq m$) porque con $k \neq m$, r_{km} tiene menos precisión que r_{jj}, donde $j = \max(k, m)$, aunque r_{jj} se puede evaluar con argumento matricial con el mismo coste [Hig08, p. 4]. A grandes rasgos, la explicación consiste en que las potencias de la matriz A calculadas para el numerador pueden reutilizarse para obtener el denominador con el mismo orden. Es bastante habitual expresar los aproximantes diagonales con un único subíndice: $r_m(z) = p_m(A)q_m(A)^{-1}$. Sin embargo, Sastre en [Sas12] ha demostrado teóricamente que las aproximaciones no diagonales con numerador de mayor grado que el denominador pueden transformarse en aproximaciones mixtas racionales y polinomiales que alcanzan mayores órdenes de aproximación que las diagonales para el mismo coste computacional.

El Algoritmo 1.2 muestra un algoritmo general para el cálculo de la exponencial de una matriz mediante Padé combinado con la técnica de escalado y potenciación.

Las fases de preprocesado y postprocesado son idénticas a las explicadas en el algoritmo de Taylor (Algoritmo 1.1). Al igual que en el caso del método de Taylor, para el cálculo de $p_m(A)$ y $q_m(A)$ es conveniente el uso de algún método que permita reagrupar los términos

Algorithm 1.2 Algoritmo general de Padé para calcular la exponencial de una matriz, $A \in \mathbb{C}^{n\times n}$, usando la técnica del escalado y potenciación

1: Preprocesado de la matriz A.
2: Elección del orden de aproximación más adecuado, m, y del parámetro de escalado, s.
3: Calcular $p_m(A/2^s)$ y $q_m(A/2^s)$.
4: Resolver el sistema $q_m(A/2^s)B = p_m(A/2^s)$ para B.
5: **for** $i = 1 : s$ **do**
6: $\quad B = B^2$
7: **end for**
8: Postprocesado de la matriz B.

del polinomio matricial. Cabe señalar que en este método se hace necesaria la resolución de un sistema de ecuaciones (paso 4), lo que conlleva restricciones sobre la matriz A para que la resolución del sistema sea factible y además se pueda realizar de forma precisa.

El método de Padé combinado con la técnica de escalado y potenciación ha sido el más utilizado durante mucho tiempo, ya que las aproximaciones basadas en series de polinomios matriciales de Taylor fueron descartadas en los años 70 debido a la conclusión comúnmente aceptada [ML03, Hig08, GL12] de que las aproximaciones racionales conseguían una precisión similar con un coste inferior. En los últimos años se han producido importantes contribuciones en el desarrollo de métodos que utilizan la técnica de escalado y potenciación basados en aproximaciones de tipo Padé racional [Hig05, AMH09]; sin embargo, también se han producido importantes contribuciones en aproximaciones basadas en polinomios de Taylor y Hermite, con implementaciones en altas prestaciones que superan en precisión y eficiencia a los algoritmos basados en aproximantes de Padé [Hig05, AMH09]. Dichas contribuciones han dado lugar a varios artículos de investigación, algunos de los cuales se incluyen en el capítulo 2 de esta tesis.

1.2.1.3. Otros métodos

Existen más métodos para el cálculo de la exponencial de una matriz, aunque ninguno de ellos es tan utilizado como los métodos que hemos visto en los apartados anteriores, ya que ofrecen en general peores prestaciones. Entre el resto de los métodos cabría destacar los basados en la descomposición de matrices. Estos métodos se basan en hallar transformaciones de semejanza

$$A = XBX^{-1}$$

de manera que el cálculo de $f(B)$ sea más sencillo que el de $f(A)$. Una vez se ha obtenido $f(B)$, se aplica la Propiedad 1.3 para obtener

$$f(A) = Xf(B)X^{-1}.$$

Los métodos basados en la descomposición de matrices resultan ser muy eficientes en problemas que involucran a matrices de dimensión elevada. Tienen la ventaja de que si A es una matriz simétrica, entonces estos métodos proporcionan algoritmos simples y efectivos. Sin embargo, tienen el problema de que si la matriz está mal condicionada, entonces se pueden producir elevados errores de redondeo.

Entre este tipo de métodos podemos encontrar, por ejemplo, los basados en la diagonalización de una matriz: si $A \in \mathbb{C}^{n\times n}$ tiene un conjunto de vectores propios $v_1, v_2, \cdots, v_n$ linealmente independientes y consideramos la matriz $v = [v_1, v_2, \cdots, v_n]$, entonces se verifica la ecuación matricial

$$AV = VD,$$

donde

$$D = \text{diag}(\lambda_1, \lambda_2, \cdots, \lambda_n),$$

siendo $\lambda_1, \lambda_2, \ldots, \lambda_n$ los valores propios de A. Aplicando las propiedades 1.3 y 1.6 se tiene que

$$e^A = V\text{diag}(e^{\lambda_1}, e^{\lambda_2}, \cdots, e^{\lambda_n})V^{-1}.$$

Otro método dentro de esta categoría es el basado en la forma canónica de Jordan. Así, según la Definición 1.2, si una matriz $A \in \mathbb{C}^{n\times n}$ admite la descomposición canónica de Jordan

$$A = X\text{diag}(J_{\lambda_1}, J_{\lambda_2}, \cdots, J_{\lambda_r})X^{-1},$$

entonces su exponencial se podría calcular como

$$e^A = X\text{diag}(e^{J_{\lambda_1}}, e^{J_{\lambda_2}}, \cdots, e^{J_{\lambda_r}})X^{-1}.$$

Este método tiene el problema de que al utilizar aritmética en coma flotante, cualquier pequeño error de redondeo puede modificar significativamente la estructura de X y de $\text{diag}(J_{\lambda_1}, J_{\lambda_2}, \cdots, J_{\lambda_r})$, produciendo resultados muy poco precisos. No obstante, en [KR80] se puede encontrar un algoritmo preciso y estable para calcular la forma canónica de Jordan de una matriz compleja cuadrada usando aritmética finita.

Un último método que cabe destacar dentro de esta categoría es la descomposición de Schur: dada una matriz $A \in \mathbb{C}^{n\times n}$, su descomposición de Schur viene dada por [DH03]

$$A = QSQ^*,$$

donde $Q \in \mathbb{C}^{n\times n}$ es una matriz ortogonal y $S \in \mathbb{C}^{n\times n}$ es triangular superior, de manera que los valores propios de A son los elementos diagonales de S. Aplicando la Propiedad 1.3, tenemos que

$$f(A) = Qf(S)Q^*.$$

Tras esta transformación, el problema se reduce a calcular $f(S)$, lo cual se puede hacer por diversos métodos que no entraremos a valorar.

Por otra parte, si $A \in \mathbb{R}^{n\times n}$ es real, entonces hablamos de la descomposición real de Schur. En este caso sería deseable poder reducir A a forma triangular usando matrices ortogonales reales, ya que la aritmética real permite implementar algoritmos más eficientes computacionalmente que la aritmética compleja. Si embargo, una matriz real puede tener valores propios reales, pero también complejos en forma de pares conjugados. Esto implica que en el caso real, la matriz S es una matriz triangular superior por bloques, con bloques diagonales de tamaño 1×1 o 2×2, de forma que los bloques de tamaño 1×1 corresponderían a los valores propios de A reales y los de tamaño 2×2 a sus valores propios complejos conjugados.

1.2.2. Funciones seno y coseno

Aunque la función exponencial de una matriz es sin duda la que más atención acapara y la que, por lo tanto, ha sido más estudiada, las funciones seno y coseno matriciales son también muy utilizadas puesto que su cálculo se hace necesario en gran cantidad de problemas. Así, de la misma forma que en la resolución de sistemas modelados por SEDs de primer orden surge la función matricial exponencial, las funciones matriciales trigonométricas seno y coseno juegan un papel fundamental en la resolución de sistemas modelados por sistemas de ecuaciones diferenciales de segundo orden. Por ejemplo, el problema

$$\frac{d^2y}{dt^2} + Ay = 0, \quad y(0) = y_0, \quad y'(0) = y'_0,$$

donde $A \in \mathbb{C}^{n\times n}$, tiene como solución

$$y(t) = \cos(\sqrt{A}t)y_0 + (\sqrt{A}t)^{-1}\text{sen}(\sqrt{A}t)y'_0,$$

siendo $\sqrt{A}$ la raíz cuadrada de A. El seno y coseno matricial también surgen en otros problemas más generales de este tipo, en los que aparece un término $f(t)$ en la parte derecha de la ecuación diferencial, como pueden ser los procedentes de los sistemas mecánicos sin amortiguamiento o de la semidiscretización de la ecuación de ondas [Ser79, SB80, HS03, HH05].

En 1980, Serbin y Blalock propusieron uno de los primeros métodos eficientes para el cálculo del coseno matricial [SB80], que tuvo su precursor en [Ser79]. Dicho método empleaba aproximaciones racionales y la reducción de la norma de la matriz junto con la fórmula del ángulo doble,

$$\cos(2A) = 2\cos^2(A) - I,$$

para el cálculo del coseno de la matriz original. La técnica del ángulo doble permite reducir la norma de la matriz de trabajo, obteniendo así unos resultados más precisos, de forma similar a la técnica del escalado y potenciación utilizada en el cálculo de la exponencial matricial. Por otra parte, utilizando la relación

$$\sin(A) = \cos(A - \frac{\pi}{2}I),$$

se puede utilizar el mismo algoritmo para calcular tanto el seno como el coseno, por lo que habitualmente se implementa únicamente un algoritmo, típicamente el del coseno. A continuación se explican los métodos más habituales utilizados para el cálculo de estas funciones trigonométricas que, como ocurre en el caso de la exponencial, son los aproximantes de Padé y las series de polinomios matriciales ortogonales.

1.2.2.1. Aproximantes de Padé

De entre las diversas contribuciones realizadas posteriormente al trabajo de Serbin y Blalock, caben destacar las propuestas realizadas por Higham, Smith y Hargreaves en [HS03, HH05, AMHR15], basadas en la fórmula del ángulo doble y en aproximantes de Padé. Estas propuestas mejoran considerablemente la eficiencia y precisión del algoritmo original, proporcionando además un análisis teórico más preciso de los errores de truncamiento y

redondeo. A diferencia del caso de la exponencial, no sabemos si existe un aproximante de Padé, r_{km}, para cualquier valor de k y m. Sin embargo, se pueden calcular aproximantes concretos utilizando la formulación de Magnus y Wynn [MW75], que proporciona los coeficientes de los aproximantes diagonales, r_m, en términos de determinantes de matrices cuyas entradas son coeficientes binomiales. Así, los dos primeros aproximantes de Padé para la función coseno son [Hig08]

$$r_2(x) = \frac{p_2(x)}{q_2(x)} = \frac{1 - \frac{5}{12}x^2}{1 + \frac{1}{12}x^2}, \quad r_4(x) = \frac{p_4(x)}{q_4(x)} = \frac{1 - \frac{115}{252}x^2 + \frac{313}{15120}x^4}{1 + \frac{11}{252}x^2 + \frac{13}{15120}x^4}.$$

Obsérvese que como la función coseno es par, solo debemos considerar los aproximantes de grado par, $(2m)$. Además, el numerador y denominador de los coeficientes racionales crece muy rápidamente, por lo que para calcular r_{2m} resulta conveniente evaluar $p_{2m} = \sum_{i=0}^{m} a_{2i}x^{2i}$ y $q_{2m} = \sum_{i=0}^{m} b_{2i}x^{2i}$ y después resolver el sistema de ecuaciones múltiple $q_2 r_2 = p_2$. Para evaluar eficientemente p_{2m} y q_{2m} es conveniente tratarlos como polinomios de grado m en A^2 y utilizar algún método de reagrupación de términos del polinomio matricial.

El Algoritmo 1.3 muestra los pasos para calcular el coseno matricial mediante Padé usando la técnica del ángulo doble.

Algorithm 1.3 Algoritmo general de Padé para calcular el coseno de una matriz, $A \in \mathbb{C}^{n\times n}$, usando la técnica del doble ángulo

Preprocesado de la matriz A:
    Determinar j que minimice $\|A - \pi jI\|$
    $A \leftarrow A - \pi jI$
    Determinar una matriz diagonal D tal que $D^{-1}AD$ esté equilibrada.
    $A \leftarrow D^{-1}AD$
$B = A^2$
Elección del orden de aproximación más adecuado, m, y del parámetro de escalado, s.
Calcular $p_m(B/4^s)$ y $q_m(B/4^s)$.
Resolver el sistema $q_m(B/4^s)C = p_m(B/4^s)$ para C.
for $i = 1 : s$ **do**
    $C = 2C^2 - I$
end for
Postprocesado de la matriz C:
    $C \leftarrow (-1)^j DCD^{-1}$

Durante el preprocesado, que es un paso opcional, lo que se pretende es reducir la norma de la matriz A mediante diversas transformaciones, para lo cual se usan dos técnicas: en primer lugar se aprovecha la periodicidad de la relación

$$\cos(A - \pi jI) = (-1)^j \cos(A), \quad j \in \mathbb{Z},$$

para buscar el valor de j que minimice la norma de la matriz $A - \pi jI$. Este valor se puede buscar utilizando el Teorema 4.21 de [Hig08]. La segunda técnica, que también se usa en el caso del cálculo de la exponencial, consiste en intentar equilibrar las normas de la i–ésima fila y la i–ésima columna, con $1 \le i \le n$, determinando una matriz diagonal D con la que aplicar la transformación de semejanza $\tilde{A} = D^{-1}(A - \pi jI)D$. Este equilibrado es un proceso heurístico que tiende a reducir la norma, aunque es algo que no se puede garantizar, por lo que habría que utilizarlo únicamente con las matrices donde la norma realmente se reduzca.

Posteriormente, se hace el cambio de variable $B = A^2$ y se averigua el orden de aproximación m más adecuado y el escalado s que aplicaremos a la matriz. Al igual que ocurre en el caso de la función exponencial, la elección adecuada de estos parámetros es básica para obtener resultados precisos. A continuación se obtienen los polinomios $p_m(B/4^s)$ y $q_m(B/4^s)$ que determinan el aproximante diagonal de Padé de orden m y se calcula $C = p_m(B/4^s)/q_m(B/4^s)$ resolviendo el sistema de ecuaciones $q_m(B/4^s)C = p_m(B/4^s)$ (ver 1.2.1.2). Una vez resuelto dicho sistema, es necesario revertir el escalado que hemos aplicado utilizando la fórmula del ángulo doble s veces (pasos 10–12). Finalmente, si se llevó a cabo la fase de preprocesado, será necesario realizar el postprocesado de la matriz C (paso 14).

1.2.2.2. Series de polinomios matriciales

En [DJ98] se propone por primera vez un método basado en series de polinomios matriciales ortogonales de Hermite para el cálculo del seno y el coseno matriciales. Este método ha sido posterioremente revisado y mejorado dentro del proceso de investigación de esta tesis, dando lugar a nuevos algoritmos usando series tanto de Hermite como de Taylor, que mejoran ampliamente en precisión y velocidad a los obtenidos con aproximantes de Padé, como se puede ver en las publicaciones incluidas en el capítulo 3.

Analizando el caso de las series polinomiales de Hermite, tenemos que dada una matriz $A \in \mathbb{C}^{r\times r}$ y un escalar $y \in \mathbb{C}$, el coseno matricial $\cos(Ay)$ se puede calcular como

$$\cos(Ay) = \frac{e^{iAy} + e^{-iAy}}{2}.$$

Utilizando la definición del polinomio matricial de Hermite definido en (1.2) en combinación con [JC96, p. 25], resulta que

$$H_n(-x, A) = (-1)^n H_n(x, A).$$

Por lo tanto, tenemos que

$$\cos(Ay) = e^{\frac{1}{\mu^2}} \sum_{n\geq 0} \frac{1}{\mu^{2n}(2n)!} H_{2n}\left(iy\mu, \frac{1}{2}A^2\right).$$

Tomando $\lambda = i\mu$ en la expresión anterior obtenemos la expresión final:

$$\cos(Ay) = e^{-\frac{1}{\lambda^2}} \sum_{n\geq 0} \frac{(-1)^n}{\lambda^{2n}(2n)!} H_{2n}\left(y\lambda, \frac{1}{2}A^2\right). \tag{1.9}$$

Actuando de forma análoga para el caso del seno matricial, teniendo en cuenta que

$$\sin(Ay) = \frac{e^{iAy} - e^{-iAy}}{2i},$$

se deduce que

$$\sin(Ay) = e^{-\frac{1}{\lambda^2}} \sum_{n\geq 0} \frac{(-1)^n}{\lambda^{2n+1}(2n+1)!} H_{2n+1}\left(y\lambda, \frac{1}{2}A^2\right). \tag{1.10}$$

Si denotamos como $C_N(\lambda, A^2)$ y $S_N(\lambda, A^2)$ a la suma parcial de los Nth primeros términos de las series (1.9) y (1.10), respectivamente, para $y = 1$ obtenemos que

$$C_N(\lambda, A^2) = e^{-\frac{1}{\lambda^2}} \sum_{n=0}^{N} \frac{(-1)^n}{\lambda^{2n}(2n)!} H_{2n}\left(\lambda, \frac{1}{2}A^2\right) \approx \cos(A), \ \lambda \in \mathbb{C}, \ A \in \mathbb{C}^{r\times r}, \quad \text{y}$$

$$S_N(\lambda, A^2) = e^{-\frac{1}{\lambda^2}} \sum_{n=0}^{N} \frac{(-1)^n}{\lambda^{2n+1}(2n+1)!} H_{2n+1}\left(\lambda, \frac{1}{2}A^2\right) \approx \sin(A), \ \lambda \in \mathbb{C}, \ A \in \mathbb{C}^{r\times r}.$$

En lo que respecta a Taylor, el coseno y el seno matricial de una matriz $A \in \mathbb{C}^{n\times n}$, se puede definir como

$$\cos(A) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k)!} A^{2k} = I - \frac{A^2}{2!} + \frac{A^4}{4!} - \frac{A^6}{6!} + \cdots,$$
$$\sin(A) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)!} A^{2k+1} = A - \frac{A^3}{3!} + \frac{A^5}{5!} - \frac{A^7}{7!} + \cdots.$$

Tomando $B = A^2$, podemos expresar estas aproximaciones de Taylor de orden $2m$ como polinomios de orden m. Por ejemplo, en el caso del coseno tendríamos

$$P_m(B) = \sum_{i=0}^{m} \frac{(-1)^i}{(2i)!} B^i \approx \cos(A).$$

Los algoritmos 1.4 y 1.5 muestran los algoritmos generales para calcular el coseno matricial usando series polinomiales de Hermite y de Taylor, respectivamente. Aunque los algoritmos son muy similares, se han incluido los dos por claridad y limpieza. Es ambos casos es posible llevar a cabo un preprocesamiento previo de la matriz con la intención de reducir su norma y mejorar así la precisión del cálculo. Las técnicas a utilizar serían las mismas que se han explicado en el algoritmo de Padé (Algoritmo 1.3).

Algorithm 1.4 Algoritmo general de Hermite para calcular el coseno de una matriz, $A \in \mathbb{C}^{n\times n}$, usando la técnica del doble ángulo

1: Preprocesado de la matriz A.
2: $B = A^2$
3: Determinar el valor óptimo de λ, el orden de aproximación más adecuado, N, y el parámetro de escalado, s.
4: Calcular la aproximación de Hermite de orden N: $C = C_N(\lambda, B/4^s)$.
5: **for** $i = 1 : s$ **do**
6: $\quad C = 2C^2 - I$
7: **end for**
8: Postprocesado de la matriz C.

Algorithm 1.5 Algoritmo general de Taylor para calcular el coseno de una matriz, $A \in \mathbb{C}^{n\times n}$, usando la técnica del doble ángulo

1: Preprocesado de la matriz A.
2: $B = A^2$
3: Elección del orden de aproximación más adecuado, m, y del parámetro de escalado, s.
4: Calcular la aproximación de Taylor de orden m: $C = P_m(B/4^s)$.
5: **for** $i = 1 : s$ **do**
6: $\quad C = 2C^2 - I$
7: **end for**
8: Postprocesado de la matriz C.

1.2.3. *Software* para el cálculo de funciones matriciales

Existen muy pocos paquetes numéricos para el cálculo de funciones matriciales, tanto dentro del *software* libre como del comercial. En el ámbito del *software* libre cabe destacar el paquete Expokit [Sid98], el cual consta de código en FORTRAN 77 y MATLAB orientado al cálculo de la exponencial de una matriz. En concreto, ofrece rutinas en aritmética real o compleja, para calcular la exponencial de pequeñas matrices densas, la exponecial de matrices dispersas grandes por un vector y la solución de sistemas lineales de ecuaciones diferenciales ordinarias (1.1). Para ello, utiliza aproximaciones racionales de Padé y Chebyshev y productos del tipo $e^{At}v$, siendo A una matriz dispersa y v un vector, mediante un algoritmo basado en subespacios de Krylov. Por otra parte, también existe *The Matrix Function Toolbox* [Hig02] que contiene implementaciones en MATLAB de muchos de los algoritmos descritos en [Hig08], aunque los códigos no han sido diseñados buscando las altas prestaciones, sino la simplicidad y facilidad de comprensión.

En lo que respecta a las herramientas comerciales para el cálculo de funciones de matrices, solamente se pueden destacar dos: MATLAB y *The NAG Library* [Int02]. En ambas herramientas las implementaciones de las rutinas que calculan funciones matriciales están basadas en los mismos métodos:

- Para el cálculo de la exponencial de una matriz se utilizan aproximantes de Padé.
- Para el cálculo del resto de las funciones matriciales se usa el algoritmo de Schur–Parlett.

El inconveniente de las implementaciones basadas en el algoritmo de Schur–Parlett es que son muy costosas desde el punto de vista computacional. Además, al estar basadas en la forma de Schur de una matriz, su paralelización resulta ineficiente. Por otra parte, el inconveniente de MATLAB es que la ejecución de programas es más lento que si se trata de implementaciones en C o Fortran. Las librerías de NAG, por su parte, son muy completas y ofrecen librerías para trabajar en multitud de lenguajes y entornos de programación.

Dentro del marco de esta tesis se ha desarrollado e implementado abundante *software* para el cálculo de funciones matriciales basado en aproximaciones polinómicas. Como hemos visto en la sección anterior, las aproximaciones polinómicas junto con las aproximaciones

racionales son los métodos más utilizados actualmente, ya que son los que mejores resultados están dando. Estos métodos son, además, los más adecuados para ser utilizados en implementaciones de altas prestaciones, puesto que por una parte permiten reorganizar los cálculos de forma que pueden realizarse de manera muy eficiente, y por otra, la mayor parte de las operaciones a realizar se pueden llevar a cabo usando núcleos computacionales básicos, como pueden ser BLAS [DCHH88] y LAPACK [ABB+92], pues consisten fundamentalmente en el cálculo de productos de matrices.

1.3. Organización de la tesis

El contenido de esta memoria está organizado en cinco capítulos. En este primer capítulo introductorio se han especificado los objetivos buscados durante la realización de la tesis y se ha realizado un resumen del estado del arte relativo al cálculo de las funciones matriciales que nos ocupan: se han presentado algunas de sus definiciones, se han descrito sus propiedades más importantes y se han detallado los métodos más habituales para su cálculo.

En el segundo capítulo se presentan los artículos relacionados con el cálculo de la función exponencial de una matriz. Se incluyen 5 artículos a través de los cuales se observa la evolución de los algoritmos de cálculo a lo largo de la investigación.

El tercer capítulo está dedicado al cálculo de las funciones seno y coseno matricial, con la inclusión de tres artículos relacionados con dichas funciones.

En el capítulo cuarto se incluyen dos artículos sobre la resolución de ecuaciones diferenciales ordinarias, en los que se hace necesario el cálculo de la exponencial de una matriz. Finalmente, en el último capítulo se presentan las conclusiones del trabajo realizado, así como el trabajo actual y futuro dentro de la línea de investigación seguida en la tesis.

Bibliografía

[ABB+92] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM, 1992.

[AMH09] A. H. Al-Mohy and N. J. Higham. A new scaling and squaring algorithm for the matrix exponential. *SIAM J. Matrix Anal. Appl.*, 31(3):970–989, 2009.

[AMHR15] Awad Al-Mohy, Nicholas Higham, and Samuel Relton. New algorithms for computing the matrix sine and cosine separately or simultaneously. 37:A456–A487, 01 2015.

[CO09] B. J. Culpepper and B. A. Olshausen. Learning transport operators for image manifolds. In *NIPS*, pages 423–431. Curran Associates, Inc., 2009.

[DCHH88] J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN Basic Linear Algebra Subroutines. *ACM Transactions on Mathematical Software*, 14:1–17, 1988.

[DH03] P. I. Davies and N. J. Higham. A Schur–Parlett algorithm for computing matrix functions. *SIAM J. Matrix Anal. Appl.*, 25(2):464–485, 2003.

[DJ98] E. Defez and L. Jódar. Some applications of Hermite matrix polynomials series expansions. *J. Comput. Appl. Math.*, 99:105–117, 1998.

[Fun04] T. C. Fung. Computation of the matrix exponential and its derivatives by scaling and squaring. *International Journal for Numerical Methods in Engineering*, 59:1273–1286, 2004.

[GL96] G. H. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins Studies in Mathematical Sciences. The Johns Hopkins University Press, third edition, 1996.

[GL12] G. H. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins Studies in Mathematical Sciences. The Johns Hopkins University Press, fourth edition, 2012.

[HH05] G. I. Hargreaves and N. J. Higham. Efficient algorithms for the matrix cosine and sine. *Numer. Algorithms*, 40:383–400, 2005.

[Hig02] N. J. Higham. The Matrix Computation Toolbox. http://www.ma.man.ac.uk/~higham/mctoolbox, 2002.

[Hig05] N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl.*, 26(4):1179–1193, 2005.

[Hig08] N. J. Higham. *Functions of Matrices: Theory and Computation*. SIAM, Philadelphia, PA, USA, 2008.

[HS03] N. J. Higham and M. I. Smith. Computing the matrix cosine. *Numer. Algorithms*, 34:13–26, 2003.

[Int02] NAG Library Function Document. http://www.nag.co.uk/content/nag-library-new-content, 2002.

[JC96] L. Jódar and R. Company. Hermite matrix polynomials and second order matrix differential equations. *Journal Approximation Theory Application*, 12(2):20–30, 1996.

[JI09] V. Hernández J. Ibáñez. Solving differential matrix riccati equations by a piecewise-linearized method based on the conmutant equation. 180:2103–2114, 11 2009.

[JI11] V. Hernández J. Ibáñez. Solving differential matrix riccati equations by a piecewise-linearized method based on diagonal pade approximants. 182:669–678, 03 2011.

[Kar59] S. Karlin. *Mathematical Methods and Theory in Games, Programming, and Economics*. Reading, Mass: Addison-Wesley Pub. Co., 1959.

[KR80] Bo Kågström and Axel Ruhe. An algorithm for numerical computation of the jordan normal form of a complex matrix. *ACM Trans. Math. Softw.*, 6(3):398–419, September 1980.

[Lau85] A. Laub. Numerical linear algebra aspects of control design computations. *IEEE Transactions on Automatic Control*, AC-30:97–108, 1985.

[ML78] C. B. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review*, 20(4):801–836, 1978.

[ML03] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*. *SIAM Review*, 45:3–49, 2003.

[MR07] X. Miao and R. P. N. Rao. Learning the lie groups of visual invariance. *Neural Computation*, 19(10):2665–2693, 2007.

[MW75] Arne Magnus and Jan Wynn. On the Padé table of cos z. *Proceedings of the American Mathematical Society*, 47, 02 1975.

[PS73] M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, 2(1):60–66, 1973.

[Sas12] Jorge Sastre. Efficient mixed rational and polynomial approximation of matrix functions. *Applied Mathematics and Computation*, 218(24):11938 – 11946, 2012.

[Sas18] J. Sastre. Efficient evaluation of matrix polynomials. *Linear Algebra and its Applications*, 539:229 – 250, 2018.

[SB80] S. M. Serbin and S. A. Blalock. An algorithm for computing the matrix cosine. *SIAM J. Sci. Statist. Comput.*, 1(2):198–204, 1980.

[Ser79] S. M. Serbin. Rational approximations of trigonometric matrices with application to second-order systems of differential equations. *Appl. Math. Comput.*, 5(1):75–92, 1979.

[Sid98] R. B. Sidje. Expokit: a software package for computing matrix exponentials. *ACM Trans. Math. Software*, 24:130–156, 1998.

[SIDR11] J. Sastre, J. Ibáñez, E. Defez, and P. Ruiz. Efficient orthogonal matrix polynomial based method for computing matrix exponential. *Appl. Math. Comput.*, 217(14):6451–6463, 2011.

[SIDR14] J. Sastre, J. Ibáñez, E. Defez, and P. Ruiz. Accurate and efficient matrix exponential computation. *Int. J. Comput. Math.*, 91(1):97–112, 2014.

[SIDR15] J. Sastre, J. Ibáñez, E. Defez, and P. Ruiz. New scaling-squaring Taylor algorithms for computing the matrix exponential. *SIAM J. Sci. Comput.*, 37(1):A439–A455, 2015.

[TVSC11] P. Turaga, A. Veeraraghavan, A. Srivastava, and R. Chellappa. Statistical computations on grassmann and stiefel manifolds for image and video-based recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(11):2273–2286, 2011.

[Var62] R. S. Varga. *Matrix Iterative Analysis*. Prentice Hall, Englewood Cliffs, New Jersey, 1962.

[WSTO11] Ch. M. Wang, J. Sohl-Dickstein, I. Tosic, and B. A. Olshausen. Lie group transformation models for predictive video coding. In *2011 Data Compression Conference (DCC 2011), 29-31 March 2011, Snowbird, UT, USA*, pages 83–92, 2011.

Capítulo 2

Exponencial de una matriz

2.1. Introducción

En este capítulo se presentan los artículos relacionados con el cálculo de la función exponencial. Se han incluido 5 artículos, a través de los cuales se puede apreciar la evolución del algoritmo de cálculo conforme el proceso de investigación ha ido avanzando en el tiempo. La última versión (sección 2.6), presenta unos resultados realmente competitivos tanto a nivel de precisión como de tiempos de ejecución. En todos los casos se utiliza la técnica de escalado y potenciación [ML03, Hig05, Hig08, Fun04] que, como se explica en la sección 1.2.1, permite reducir significativamente los errores de redondeo al explotar la propiedad

$$e^A = (e^{(A/m)})^m.$$

Los métodos de cálculo empleados están basados en series de polinomios matriciales ortogonales de Taylor y de Hermite (sección 2.2). A continuación se presenta la lista de los artículos incluidos en este capítulo, junto a una breve reseña de la novedad que ofrece cada uno de ellos:

- *"Efficient orthogonal matrix polynomial based method for computing matrix exponential"* (sección 2.2, [SIDR11b]): en este primer trabajo se desarrolla un método basado en series de polinomios matriciales de Hermite para el cálculo de la exponencial. Inspirándonos en las ideas presentadas por Higham en [Hig05], obtenemos una cota para el error de tipo *backward* que permite obtener el escalado óptimo teórico para matrices generales en el método de Hermite. Para poner a prueba este método, se implementaron dos algoritmos en MATLAB que resultaron ser más precisos que los del estado del arte en prácticamente el 80 % de los casos testados, además de tener un coste computacional inferior en el 10 % de los mismos. Estos resultados tan positivos nos animaron a seguir investigando en esta dirección.

- *"Accurate matrix exponential computation to solve coupled differential models in Engineering"* (sección 2.3, [SIDR11a]): a partir de las ideas desarrolladas en el artículo anterior, en este trabajo se presentó un nuevo algoritmo basado en aproximaciones

de Taylor y en el cálculo de polinomios de matrices mediante el método de Paterson-Stockmeyer, junto con un estudio de errores de tipo *forward* y *backward* para la determinación de los valores óptimos del orden de la aproximación y del escalado. Para ello se usaron nuevas cotas más ajustadas tanto para matrices normales como no normales. Aplicado al método de Taylor, la implementación desarrollada en MATLAB del nuevo algoritmo demostró ser más preciso en la mayoría de los casos que los algoritmos del estado del arte, con un coste computacional similar. Para poder comparar nuestro algoritmo con los más avanzados del estado del arte hasta el momento, también realizamos la implementación en MATLAB del nuevo algoritmo propuesto por Al-Mohy y Higham en [AMH09].

- *"New scaling-squaring Taylor algorithms for computing the matrix exponential"* (sección 2.4, [SIDR15]): en este artículo se desarrollaron e implementaron en MATLAB cuatro nuevos algoritmos de Taylor para el cálculo de la función exponencial. Dichos algoritmos son variantes del algoritmo estándar, basadas en dos modificaciones pensadas para reducir el coste computacional sin afectar a la precisión. La primera modificación afecta al orden del parámetro de escalado utilizado, mientras que la segunda se basa en despreciar en el cálculo los términos de mayor orden de la serie de Taylor, cuando esto no afecte a la precisión final obtenida. Cabe señalar que la primera versión de este artículo se envió en el año 2008, siendo anterior al artículo anterior; sin embargo, por cuestiones de la revista tardó mucho tiempo en publicarse.

- *"Accurate and efficient matrix exponential computation"* (sección 2.5, [SIDR14]): la novedad del algoritmo presentado en este artículo es la introducción de variabilidad en las cotas de los errores relativos de tipo *forward* y *backward* de la aproximación de Taylor, teniendo en cuenta para ello los errores en coma flotante que se comenten en dicho método. Así, las nuevas cotas dependen tanto del tamaño de la matriz, como del orden usado en la aproximación de Taylor. La implementación en MATLAB de este algoritmo consiguió reducir el coste computacional en la mayoría de los casos, con un impacto muy pequeño a nivel de precisión al compararlo con el algoritmo desarrollado en [SIDR11a] y obteniendo mejor precisión que los basados en Padé.

- *"High performance computing of the matrix exponential"* (sección 2.6, [RSID16]): en este artículo se presenta la última y más eficiente versión desarrollada durante el desarrollo de la tesis para el cálculo de la función exponencial. Este nuevo algoritmo se basa en las mejoras realizadas en los artículos anteriores, especialmente en el artículo "Accurate matrix exponential computation to solve coupled differential models in Engineering", pero simplificando significativamente el cálculo de las cotas del error usados para seleccionar tanto el orden de aproximación como el parámetro de escalado. De este modo conseguimos un algoritmo con mejores prestaciones que los del estado del arte, incluidos los presentados en nuestros anteriores artículos. En este caso, además de implementar el nuevo algoritmo en MATLAB, también se realizó la implementación en FORTRAN para poder realizar una comparativa con el principal software comercial disponible para calcular la matriz exponencial, la librería NAG [NAG02], con unos resultados sobresalientes, tanto en precisión como en tiempo computacional.

En las próximas secciones de este capítulo se presentan los artículos mencionados al completo.

2.2. Efficient orthogonal matrix polynomial based method for computing matrix exponential

Referencia del artículo:
J. Sastre, J. Ibáñez, E. Defez, and P. Ruiz
Efficient orthogonal matrix polynomial based method for computing matrix exponential
Applied Mathematics and Computation, Volume 217, Issue 14, March 2011, Pages 6451–6463, ISSN 0096-3003, http://dx.doi.org/10.1016/j.amc.2011.01.004

Abstract

The matrix exponential plays a fundamental role in the solution of differential systems which appear in different science fields. This paper presents an efficient method for computing matrix exponentials based on Hermite matrix polynomial expansions. Hermite series truncation together with scaling and squaring and the application of floating point arithmetic bounds to the intermediate results provide excellent accuracy results compared with the best acknowledged computational methods. A backward-error analysis of the approximation in exact arithmetic is given. This analysis is used to provide a theoretical estimate for the optimal scaling of matrices. Two algorithms based on this method have been implemented as MATLAB functions. They have been compared with MATLAB functions `funm` and `expm` obtaining greater accuracy in the majority of tests. A careful cost comparison analysis with `expm` is provided showing that the proposed algorithms have lower maximum cost for some matrix norm intervals. Numerical tests show that the application of floating point arithmetic bounds to the intermediate results may reduce considerably computational costs, reaching in numerical tests relative higher average costs than `expm` of only 4.43 % for the final Hermite selected order, and obtaining better accuracy results in the 77.36 % of the test matrices. The MATLAB implementation of the best Hermite matrix polynomial based algorithm has been made available online.

2.2.1. Introduction

Many engineering processes are described by systems of linear first-order ordinary differential equations. Solving this kind of systems involves the evaluation of the exponential of square matrices, and the same occurs with the partial differential case when using the semi-discretization method [FDC46]–[Smi85].

A survey of methods for computing the exponential matrix was made by Moler and Van Loan in [ML78], which was updated in [ML03]. This paper and recent researches [Hig05, AMH09] show that probably one of the more promising methods is the scaling and squaring technique, which is also the most widely used. This technique exploits the relation $\exp(A) = (\exp(2^{-s}A))^{2^s}$, for a square matrix A and a nonnegative integer scaling parameter s. Most approximations to the exponential of the scaled matrix $\exp(2^{-s}A)$ are based on Padé expansions. [Hig05] provides a scaling and squaring Padé method with excellent results of efficiency and accuracy, which is the method implemented in the last MATLAB

versions, and [AMH09] has recently proposed a new scaling and squaring algorithm that alleviates the overscaling problem for nonnormal matrices.

In this paper we use Hermite matrix polynomial expansions of the matrix exponential together with scaling and squaring and the application of floating point arithmetic bounds to the intermediate results in order to perform a competitive method for computing the matrix exponential. This method has the advantage that it does not need the solution of multiple linear systems or the computation of matrix inversions. Moreover, adapting the analysis made in [Hig05] to the Hermite matrix series, a backward-error bound in exact arithmetic has been provided to find the optimal matrix scaling, i.e. the optimal value of scaling parameter s. A careful theoretical cost comparison with the method in [Hig05] has been provided, showing that Hermite method has lower cost for some matrix norm intervals.

This paper is organized as follows. Section 2.2.2 summarizes notation and previous results about Hermite matrix polynomials and includes the Hermite series expansion of the matrix exponential to be considered. Section 2.2.3 deals with the approximation error analysis and optimal scaling. Section 2.2.4 is addressed to study the cost of the method, and to present numerical tests in order to check the method accuracy performance. Finally conclusions are given in Section 2.2.5.

2.2.2. Hermite matrix polynomial series expansions of matrix exponential

Throughout this paper, for a complex number z, $\Re(z)$ and $\Im(z)$ denote its real and imaginary parts, respectively, and $\|\cdot\|$ denotes any subordinate matrix norm. $\mathbb{R}^{r\times r}$ and $\mathbb{C}^{r\times r}$ denote the set of real and complex matrices of size $r\times r$, respectively, and I denotes the identity matrix for these sets. For a matrix $A\in\mathbb{C}^{r\times r}$, its spectrum $\sigma(A)$ denotes the set of all the eigenvalues of A. If $f(z)$ and $g(z)$ are holomorphic functions of the complex variable z, which are defined in an open set Ω of the complex plane, and B is a matrix in $\mathbb{C}^{r\times r}$ with $\sigma(B)\subset\Omega$, then from the properties of the matrix functional calculus [DS57, p. 558], it follows that $f(B)g(B)=g(B)f(B)$. If $\mathbb{D}_0$ is the complex plane cut along the negative real axis and $\log(z)$ denotes the principal logarithm of z, [SZ71, p. 72], then $z^{\frac{1}{2}}$ represents $\exp\left(\frac{1}{2}\log(z)\right)$. If B is a matrix with $\sigma(B)\subset\mathbb{D}_0$, then $B^{\frac{1}{2}}=\sqrt{B}$ denotes the image by $z^{\frac{1}{2}}$ of the matrix functional calculus acting on the matrix B. We say that matrix A in $\mathbb{C}^{r\times r}$ is a positive stable matrix if $Re(z)>0$ for all $z\in\sigma(A)$. $[x]$ denotes the entire part of x, $\lceil x\rceil$ denotes the least integer not less than x and $\lfloor x\rfloor$ denotes the greatest integer not exceeding x.

For the sake of clarity in the presentation of the next results we recall some properties and results about Hermite matrix polynomials that have been established in [DJ98] and [JC96]. If B is a positive stable matrix in $\mathbb{C}^{r\times r}$, the n–th Hermite matrix polynomial is defined in (3.4) of [JC96, p. 25] by

$$H_n(x,B)=n!\sum_{k=0}^{[\frac{n}{2}]}\frac{(-1)^k\left(x\sqrt{2B}\right)^{n-2k}}{k!(n-2k)!}\ , \tag{2.1}$$

and from (3.1) and (3.2) of [JC96, p. 24] one gets its generating function

$$G(x,t) = e^{xt\sqrt{2B}-t^2 I} = \sum_{n\geq 0} H_n(x,B)t^n/n!, \quad |t| < \infty, \tag{2.2}$$

where $x \in \mathbb{C}$ and $t \in \mathbb{C}$. Taking $A = \sqrt{2B}$, $y = tx$ and $\lambda = 1/t$ in (2.2) it follows that

$$e^{Ay} = e^{\frac{1}{\lambda^2}} \sum_{n\geq 0} \frac{1}{\lambda^n n!} H_n\left(\lambda y, \frac{1}{2}A^2\right), \ \lambda \in \mathbb{C}, \quad y \in \mathbb{C}, \quad A \in \mathbb{C}^{r\times r}, \tag{2.3}$$

without restrictions on $\sigma(A)$. From (2.1) one gets

$$H_n(\lambda y, A^2/2) = n! \sum_{k=0}^{\left[\frac{n}{2}\right]} \frac{(-1)^k(\lambda y A)^{n-2k}}{k!(n-2k)!}\ , \tag{2.4}$$

also without restrictions on $\sigma(A)$. Denoting by $h_m(\lambda y, A)$ the $m-$th partial sum of series (2.3), one gets

$$h_m(\lambda y, A) = e^{\frac{1}{\lambda^2}} \sum_{n=0}^{m} \frac{1}{\lambda^n n!} H_n\left(\lambda y, \frac{1}{2}A^2\right) \approx e^{Ay}, \quad \lambda, y \in \mathbb{C}, \quad A \in \mathbb{C}^{r\times r}. \tag{2.5}$$

This expansion is the one to be used in the method for computing the matrix exponential, and we will refer to m as the order of the approximation. Using (2.4) and (2.5) by induction it is easy to show that

$$h_m(\lambda y, A) = \begin{cases} e^{\frac{1}{\lambda^2}}\left[E_0^{\frac{m-1}{2}}(I+Ay) + E_0^{\frac{m-1}{2}-1}\left(\frac{(Ay)^2}{2!}+\frac{(Ay)^3}{3!}\right)+\cdots\right. \\ \left.\cdots + E_0^1\left(\frac{(Ay)^{m-3}}{(m-3)!}+\frac{(Ay)^{m-2}}{(m-2)!}\right)+\frac{(Ay)^{m-1}}{(m-1)!}+\frac{(Ay)^m}{m!}\right] \ , \ \text{odd} \ \ m\ , \\ e^{\frac{1}{\lambda^2}}\left[E_0^{\frac{m}{2}} I + E_0^{\frac{m}{2}-1}\left(Ay+\frac{(Ay)^2}{2!}\right)+E_0^{\frac{m}{2}-2}\left(\frac{(Ay)^3}{3!}+\frac{(Ay)^4}{4!}\right)+\cdots\right. \\ \left.\cdots + E_0^1\left(\frac{(Ay)^{m-3}}{(m-3)!}+\frac{(Ay)^{m-2}}{(m-2)!}\right)+\frac{(Ay)^{m-1}}{(m-1)!}+\frac{(Ay)^m}{m!}\right] \ , \ \text{even} \ \ m\ , \end{cases} \tag{2.6}$$

where

$$E_i^j = \sum_{k=i}^{j} \frac{\left(-\frac{1}{\lambda^2}\right)^k}{k!}\ , \ i,j \in \mathbb{N}, j \geq i\ . \tag{2.7}$$

Note that for $|\lambda| \to \infty$, $e^{\frac{1}{\lambda^2}} = 1$, $E_0^j = 1$, and $h_m(\lambda y, A)$ tends to the Taylor series of order m of matrix exponential e^{Ay}.

2.2.3. Error analysis.

Using (2.6) with $y = 1$ and taking into account Taylor series of $\exp\left(-\frac{1}{\lambda^2}\right)$, for odd m it follows that

$$\begin{aligned} e^A - h_m(\lambda, A) &= \lim_{M\to\infty}\left(h_M(\lambda, A) - h_m(\lambda, A)\right) \\ &= e^{\frac{1}{\lambda^2}}\left[\sum_{k\geq\frac{m+1}{2}} \frac{\left(-\frac{1}{\lambda^2}\right)^k}{k!}(I+A) + \sum_{k\geq\frac{m-1}{2}} \frac{\left(-\frac{1}{\lambda^2}\right)^k}{k!}\left(\frac{A^2}{2!}+\frac{A^3}{3!}\right)+\cdots\right. \\ &\quad \left.\cdots + \sum_{k\geq 1}\frac{\left(-\frac{1}{\lambda^2}\right)^k}{k!}\left(\frac{A^{m-1}}{(m-1)!}+\frac{A^m}{m!}\right) + \sum_{k\geq 0}\frac{\left(-\frac{1}{\lambda^2}\right)^k}{k!}\sum_{j\geq m+1}\frac{A^j}{j!}\right] \end{aligned}$$

$$
\begin{aligned}
&= e^{\frac{1}{\lambda^2}}\left[\sum_{k\geq\frac{m+1}{2}}\frac{\left(-\frac{1}{\lambda^2}\right)^k}{k!}e^A + E_{\frac{m-1}{2}}^{\frac{m-1}{2}}\left(\frac{A^2}{2!}+\frac{A^3}{3!}\right)+\right.\\
&\left.+E_{\frac{m-3}{2}}^{\frac{m-1}{2}}\left(\frac{A^4}{4!}+\frac{A^5}{5!}\right)+\cdots+E_1^{\frac{m-1}{2}}\left(\frac{A^{m-1}}{(m-1)!}+\frac{A^m}{m!}\right)+E_0^{\frac{m-1}{2}}\sum_{j\geq m+1}\frac{A^j}{j!}\right]\\
&= \left(1-e^{\frac{1}{\lambda^2}}E_0^{\frac{m-1}{2}}\right)e^A + e^{\frac{1}{\lambda^2}}\left[E_{\frac{m-1}{2}}^{\frac{m-1}{2}}\left(\frac{A^2}{2!}+\frac{A^3}{3!}\right)+E_{\frac{m-3}{2}}^{\frac{m-1}{2}}\left(\frac{A^4}{4!}+\frac{A^5}{5!}\right)+\cdots\right.\\
&\left.\cdots+E_1^{\frac{m-1}{2}}\left(\frac{A^{m-1}}{(m-1)!}+\frac{A^m}{m!}\right)+E_0^{\frac{m-1}{2}}\sum_{j\geq m+1}\frac{A^j}{j!}\right]. \qquad (2.8)
\end{aligned}
$$

Analogously, using (2.6), for even m one gets

$$
\begin{aligned}
e^A - h_m(\lambda,A) = \left(1-e^{\frac{1}{\lambda^2}}E_0^{\frac{m}{2}}\right)e^A + e^{\frac{1}{\lambda^2}}&\left[E_{\frac{m}{2}}^{\frac{m}{2}}\left(A+\frac{A^2}{2!}\right)+E_{\frac{m}{2}-1}^{\frac{m}{2}}\left(\frac{A^3}{3!}+\frac{A^4}{4!}\right)\right.\\
&\left.+\cdots+E_1^{\frac{m}{2}}\left(\frac{A^{m-1}}{(m-1)!}+\frac{A^m}{m!}\right)+E_0^{\frac{m}{2}}\sum_{j\geq m+1}\frac{A^j}{j!}\right]. \qquad (2.9)
\end{aligned}
$$

In a similar way to the demonstration of Theorem 2.1 of [Hig05, p. 1182] we have the following result:

Theorem 1 *Assume that the matrix exponential Hermite approximation $h_m(\lambda,A)$ of (2.5) satisfies*

$$
e^{-2^{-s}A}h_m(\lambda,2^{-s}A) = I+G, \qquad (2.10)
$$

where $\|G\|<1$. Then there exists a matrix E that commutes with A such that

$$
[h_m(\lambda,2^{-s}A)]^{2^s} = e^{A+E}, \qquad (2.11)
$$

and

$$
\frac{\|E\|}{\|A\|} \leq \frac{-\log(1-\|G\|)}{\|2^{-s}A\|}. \qquad (2.12)
$$

We seek to bound the norm of G in (2.10) in terms of $\|2^{-s}A\|$. Define the function

$$
\rho(\lambda,A) = e^{-A}h_m(\lambda,A) - I, \qquad (2.13)
$$

and note that using (2.8), (2.9) and (2.13) one gets

$$
\rho(\lambda,A) = -e^{-A}(e^A - h_m(\lambda,A)) \qquad (2.14)
$$

$$
= \begin{cases}
\left(e^{\frac{1}{\lambda^2}}E_0^{\frac{m-1}{2}}-1\right)I - e^{-A}e^{\frac{1}{\lambda^2}}\left[E_{\frac{m-1}{2}}^{\frac{m-1}{2}}\left(\frac{A^2}{2!}+\frac{A^3}{3!}\right)+E_{\frac{m-3}{2}}^{\frac{m-1}{2}}\left(\frac{A^4}{4!}+\frac{A^5}{5!}\right)+\right.\\
\left.+\cdots+E_1^{\frac{m-1}{2}}\left(\frac{A^{m-1}}{(m-1)!}+\frac{A^m}{m!}\right)+E_0^{\frac{m-1}{2}}\sum_{j\geq m+1}\frac{A^j}{j!}\right], \text{ odd } m\\
\left(e^{\frac{1}{\lambda^2}}E_0^{\frac{m}{2}}-1\right)I - e^{-A}e^{\frac{1}{\lambda^2}}\left[E_{\frac{m}{2}}^{\frac{m}{2}}\left(A+\frac{A^2}{2!}\right)+E_{\frac{m}{2}-1}^{\frac{m}{2}}\left(\frac{A^3}{3!}+\frac{A^4}{4!}\right)+\right.\\
\left.+\cdots+E_1^{\frac{m}{2}}\left(\frac{A^{m-1}}{(m-1)!}+\frac{A^m}{m!}\right)+E_0^{\frac{m}{2}}\sum_{j\geq m+1}\frac{A^j}{j!}\right], \text{ even } m
\end{cases} \qquad (2.15)
$$

Hence, taking into account (2.15), and using Taylor series of e^{-A} one gets

$$\|\rho(\lambda, 2^{-s}A)\| \leq \begin{cases} \left|e^{\frac{1}{\lambda^2}} E_0^{\frac{m-1}{2}} - 1\right| + e^{\Re\left(\frac{1}{\lambda^2}\right)} \sum_{j\geq 2} |c_j| \theta^j \ , \text{ odd } m \\ \left|e^{\frac{1}{\lambda^2}} E_0^{\frac{m}{2}} - 1\right| + e^{\Re\left(\frac{1}{\lambda^2}\right)} \sum_{j\geq 1} |c'_j| \theta^j \ , \text{ even } m \end{cases} \tag{2.16}$$

where $\theta = \|2^{-s}A\|$. Note that using (2.15) for $|\lambda| \to \infty$ it follows that

$$\|\rho(\lambda, A)\| = \left\| -e^{-A} \sum_{j\geq m+1} \frac{A^j}{j!} \right\| = \left\| e^{-A} \sum_{j=0}^{m} \frac{A^j}{j!} - I \right\| \ , \ |\lambda| \to \infty \ , \tag{2.17}$$

which is the corresponding bound for Taylor series approximation of matrix exponential. Using (2.10), (2.15) and (2.16) it follows that

$$\|G\| = \|\rho(\lambda, 2^{-s}A)\| \leq f_m(\lambda, \theta) \ , \tag{2.18}$$

where $f_m(\lambda, \theta)$ is given by the expressions on the right hand side of (2.16) depending on m being odd or even. Note that taking $|\lambda| \to \infty$, using (2.17), (2.18) and Taylor expansion of e^{-A}, it follows that

$$\|G\| = \|\rho(\lambda, 2^{-s}A)\| \leq f_m(|\lambda| \to \infty, \theta) \tag{2.19}$$

where

$$f_m(|\lambda| \to \infty, \theta) = \sum_{j\geq m+1} \left|c''_j\right| \theta^j \ , \tag{2.20}$$

which is the corresponding bound for Taylor approximation of matrix exponential. Combining (2.12) with (2.18) one gets

$$\frac{\|E\|}{\|A\|} \leq \frac{-\log(1 - f_m(\lambda, \theta))}{\theta} \ . \tag{2.21}$$

Using MATLAB's Symbolic Math Toolbox we have evaluated $f_m(\lambda, \theta)$ for (2.18) and (2.20), in 250 decimal digit arithmetic, summing the first 150 series terms in both expressions, where the coefficients c_j, c'_j and c''_j have been obtained symbolically, c_j, c'_j being functions of parameter λ.

For $m = 1, 2, \ldots, 30$ we have used a zero-finder to determine the largest value of θ, denoted by θ_m^{∞}, such that the backward error bound (2.21) for $|\lambda| \to \infty$ does not exceed the unit roundoff u in IEEE double precision arithmetic, $u = 2^{-53}$. These values correspond to Taylor approximation of matrix exponential. Table 2.1 presents the results with 4 significant digits.

Substituting these values in (2.21) and varying parameter λ, experimentally the bound (2.21) is monotonically decreasing to u for $\lambda > 0$ and $m = 1, 2, \ldots, 21$, and it has minima lower than u for $6 < \lambda < 27$ and $m = 22, 23, \ldots, 30$, see figure 2.1. So it is possible to determine $\theta_m > \theta_m^{\infty}$ for $m \geq 22$ searching for those minima. We have used an iterative process and a minimum finder to obtain the new maximum values of θ_m, $m = 22, 23, \ldots, 30$, and the minima of $\lambda > 0$ for which the backward error bound (2.21) does not exceed u. Table 2.2 presents the results for the new θ_m values with 4 significant digits. Therefore, we will consider Taylor approximation of matrix exponential for $m \leq 21$ and Hermite approximation for $m \geq 22$.

m	1	2	3	4	5	6	7	8
θ_m^∞	2.220e-16	2.581e-8	1.386e-5	3.397e-4	2.401e-3	9.066e-3	2.384e-2	4.991e-2
m	9	10	11	12	13	14	15	16
θ_m^∞	8.958e-2	1.448e-1	2.142e-1	2.996e-1	3.998e-1	5.139e-1	6.411e-1	7.803e-1
m	17	18	19	20	21	22	23	24
θ_m^∞	9.305e-1	1.091	1.260	1.438	1.624	1.816	2.015	2.219
m	25	26	27	28	29	30		
θ_m^∞	2.429	2.643	2.861	3.084	3.310	3.540		

Table 2.1: Maximal values θ_m^∞ of $\|2^{-s}A\|$ such that the backward error bound (2.21) does not exceed $u = 2^{-53}$ for $|\lambda| \to \infty$, i.e. Taylor expansion of matrix exponential.

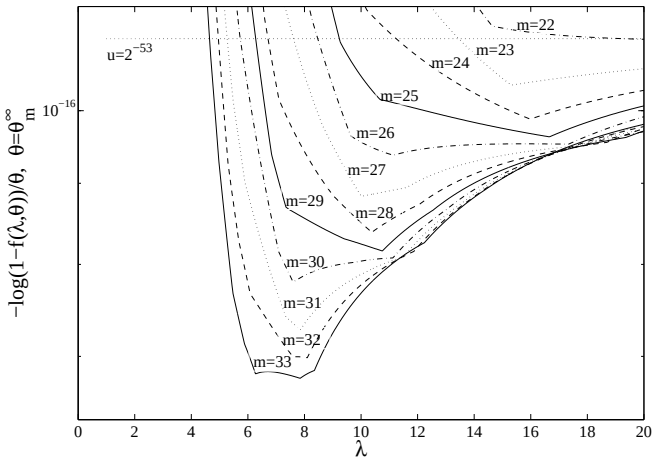


Figure 2.1: Bound (2.21) for $m = 22, 23, \ldots, 33$ and $\theta = \theta_m^\infty$, vs. parameter λ.

m	22	23	24	25	26	27	28	29	30
θ_m	1.816	2.020	2.229	2.441	2.659	2.884	3.113	3.342	3.579
λ	26.904	15.311	15.986	16.661	11.074	10.009	10.381	10.753	7.596

Table 2.2: Maximal values θ_m of $\|2^{-s}A\|$ such that the backward error bound (2.21) does not exceed $u = 2^{-53}$ and values of λ for which this is accomplished.

m	$h_m(\lambda, A)$
4	$(F_0^0/4!A^2 + F_0^0/3!A + F_0^1/2I)A^2 + F_0^1 A + F_0^2 I$
6	$((F_0^0/6!A^2 + F_0^0/5!A + F_0^1/4!I)A^2 + F_0^1/3!A + F_0^2/2I)A^2 + F_0^2 A + F_0^3 I$
9	$((F_0^0/9!A^3 + F_0^0/8!A^2 + F_0^1/7!A + F_0^1/6!I)A^3 + F_0^2/5!A^2 + F_0^2/4!A + F_0^3/3!I)A^3 + F_0^3/2A^2 + F_0^4(A + I)$

Table 2.3: Evaluation of $h_m(\lambda, A)$, where $F_0^n = e^{\frac{1}{\lambda^2}} E_0^n$. Set $e^{\frac{1}{\lambda^2}} = 1$ and $F_0^n = 1$ to obtain the Taylor approximation.

From pages 73-74 and Table 4.1 of [Hig08, p. 74], using Horner's and Paterson-Stockmeyer's methods [PS73], we can evaluate a matrix polynomial of degree m

m	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
π_m	0	1	2	2	3	3	4	4	4	5	5	5	6	6	6
C_m	52.00	26.21	18.14	13.52	11.70	9.79	9.39	8.32	7.48	7.79	7.22	6.74	7.32	6.96	6.64
m	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
π_m	6	7	7	7	7	8	8	8	8	8	9	9	9	9	9
C_m	6.36	7.10	6.87	6.67	6.48	7.30	7.14	6.99	6.84	6.71	7.59	7.47	7.36	7.26	7.16

Table 2.4: Number of matrix multiplications, π_m. Measure of overall cost C_m.

$$P_m(A) = \sum_{k=0}^{m} p_k A^k \,, \tag{2.22}$$

maximizing the degree of the polynomial for a given number of matrix product evaluations, for $m^* = \{1, 2, 4, 6, 9, 12, 16, 20, 25, 30, 36, \ldots\}$, i.e. for $m = k^2$, $k = 1, 2, 3, \ldots$, and $m = k^2 - k$, $k = 2, 3, \ldots$, using next evaluation formula

$$\begin{aligned} P_m(A) \;=\; & \Big(\big(\cdots\big(A^q p_m + A^{q-1} p_{m-1} + \cdots + A p_{m-q+1} + I p_{m-q}\big) \\ & \times\; A^q + A^{q-1} p_{m-q-1} + A^{q-2} p_{m-q-2} + \cdots + A p_{m-2q+1} + p_{m-2q}\Big) \\ & \quad \ldots\ldots \\ & \times\; A^q + A^{q-1} p_{q-1} + A^{q-2} p_{q-2} + \cdots + A p_1 + I p_0, \end{aligned} \tag{2.23}$$

calculating and saving previously the matrix powers $A^2, A^3, \ldots, A^q$, where one can take $q = \lceil \sqrt{m} \rceil$ or $q = \lfloor \sqrt{m} \rfloor$. The even matrix powers can be calculated as $A^2 = AA$, $A^4 = A^2 A^2$, $A^6 = A^4 A^2, \ldots$, and the odd matrix powers as $A^{2k+1} = AA^{2k}$, $k = 1, 2, \ldots$ Using (2.6), (2.7) and (2.23), Table 2.3 presents the evaluation formula of $h_m(\lambda, A)$ for some $m \in m^*$, where we use the coefficients $F_0^n = e^{\frac{1}{\lambda^2}} E_0^n$ and we selected $q = \lfloor \sqrt{m} \rfloor$ to minimize the number of matrix powers in memory. Note that one can obtain Taylor approximation from this table setting $e^{\frac{1}{\lambda^2}} = 1$ and $F_0^n = 1$. On the other hand, given parameter λ, the coefficients of the matrix powers in the approximations $h_m(\lambda, A)$ can be evaluated only once in 250 digit precision arithmetic before being rounded to IEEE double precision arithmetic. Evaluating $h_m(\lambda, A)$ for the rest of values of m in a similar way, we determine the cost of evaluating $h_m(\lambda, A)$, in terms of matrix powers. Selecting the optimal scaling parameter as $s = \lceil \log_2 \|A\| / \theta_m \rceil$ if $\|A\| \geq \theta_m$, and $s = 0$ otherwise, the cost of the algorithm in matrix multiplications is

$$\pi_m + s = \pi_m + \max(\lceil \log_2 \|A\| - \log_2 \theta_m \rceil, 0) \,, \tag{2.24}$$

where π_m denotes the number of matrix products evaluated to obtain Taylor or Hermite matrix polynomial approximations. It is important to note that the lower the θ_m values are, the larger the number of final squaring steps are necessary. Considering $\|A\| \geq \theta_m$ and ignoring the constant shift $\|A\|$ we have to minimize

$$C_m = \pi_m - \log_2 \theta_m \,, \tag{2.25}$$

in order to obtain the best choice for m in the Hermite approximation, see [Hig05, p. 1184]. Considering Taylor approximation of matrix exponential for $m < 22$ and therefore $\theta_m = \theta_m^{\infty}$, $m < 22$, table 2.4 presents the values C_m and π_m for $m = 1, 2, \ldots, 30$.

From table 2.4, C_m has an absolute minimum for $m = 16$, and local minima for $m = 9, 12, 20, 25, 30$, which correspond with optimal values in terms of number of matrix products.

Hermite				Padé		
m	θ_m	λ	π_m	m	θ'_m	π'_m
4	3.39716883997686e-4	–	2	3	1.495585217958292e-2	2
6	9.06565640759510e-3	–	3	5	2.539398330063230e-1	3
9	8.95776020322334e-2	–	4	7	9.504178996162932e-1	4
12	0.2996158913811581	–	5	9	2.097847961257068	5
16	0.7802874256626574	–	6	13	5.371920351148152	6
20	1.4382525968043369	–	7			
25	2.441356829252848	16.66121324200387	8			
30	3.578700513755017	7.596210771817034	9			

Table 2.5: Theoretical optimal values of θ_m for Hermite method (m = order of approximation, π_m = number of matrix products). For $m \leq 20$, $\theta_m = \theta_m^\infty$, i.e. the values relating to Taylor approximation. For $m = 25, 30$, θ_m have been obtained for the Hermite approximation, with the corresponding optimal values of λ. Parameter comparison with Padé `expm` method.

Thus, considering this analysis the optimal order, meaning with minimal cost, would be $m = 16$. Table 2.5 presents the corresponding values of θ_m in IEEE double precision arithmetic and a comparison with the same values for Padé method proposed in [Hig05]. From Table 2.5 one gets that $\theta_{20} < 2\theta_{16}$. Thus, if $m = 20$ and the resulting scaling parameter is $s \geq 1$ then there exist matrices such that $\theta_{20}/2 < \|A/2^s\| \leq \theta_{16}$, and for such matrices one can use order $m = 16$ instead of 20 in the approximation, saving one matrix product. The same occurs with orders $m = 25$ and 20, and $m = 30$ and 25 and we will take this into account in the Hermite based algorithms. We will show in section 2.2.4 that with this modification the optimal maximum order is $m = 20$ instead of $m = 16$.

With respect to rounding errors, we rule out $m = 1$ and 2 as maximum orders, as Taylor approximation can suffer from loss of significance in floating point arithmetic for those orders taking into account the values of θ_1^∞ and θ_2^∞ from Table 2.1, see [Hig05, p. 1184].

The effect of rounding errors on the evaluation of the matrix polynomial $h_m(\lambda, A)$ can be bounded analogously to the numerator of Padé approximants in [Hig05, p. 1185]. Using Theorem 2.2 of [Hig05, p. 1184], taking into account that $\|A\|_1 \leq \theta_m$, $e^{-\|A\|} \leq \|e^A\|$, and noting that $h_m(\lambda, A)$ has all positive coefficients for $\lambda \geq 1$, it follows that

$$\begin{aligned}\left\|h_m(\lambda,A)-\hat{h}_m(\lambda,A)\right\|_1 &\leq \widetilde{\gamma}_{mn} h_m\left(\lambda,\|A\|_1\right) \approx \widetilde{\gamma}_{mn} e^{\|A\|_1} \leq \widetilde{\gamma}_{mn}\left\|e^A\right\|_1 e^{2\|A\|_1} \\ &\simeq \widetilde{\gamma}_{mn}\left\|h_m(\lambda,A)\right\|_1 e^{2\|A\|_1} \leq \widetilde{\gamma}_{mn}\left\|h_m(\lambda,A)\right\|_1 e^{2\theta_m},\end{aligned} \tag{2.26}$$

where $A \in \mathbb{C}^{n\times n}$, $\hat{h}_m(\lambda, A)$ is the computed Hermite approximation using explicit formation of matrix powers as in (2.23), and $\widetilde{\gamma}_k = cku/(1-cku)$ with c a small integer constant [Hig02a]. Hence, the relative error is bounded approximately by $\widetilde{\gamma}_{mn} e^{2\theta_m}$, which is a satisfactory bound taking into account the values of θ_m given in Tables 2.1 and 2.2. Analogously, a similar bound can be obtained for Taylor matrix polynomial.

We have implemented two algorithms for computing the matrix exponential by the Hermite method presented in this paper. The first algorithm (`dgeexfher`) computes, for double precision general matrices, the exponential function by a Hermite approximation using the classical Horner's and Paterson-Stockmeyer methods [PS73], [GL96, p. 568], [Hig08, p. 73].

The second algorithm saves computational cost taking into account the relative accuracy bounds in IEEE double precision arithmetic. The underlying idea is that if the contribution of the highest degree terms of the Hermites series to the exponential of the scaled matrix is negligible taking into account floating point arithmetic bounds, then we can save the evaluation of matrix products without substantial changes in the final result. For example, for $m = 9$, let

$$F = p_9 A^3 + p_8 A^2 + p_7 A, \tag{2.27}$$

where p_i, $i = 0, 1, \cdots, 9$, are the coefficients of the matrix powers A^i of the corresponding Hermite expansion in Table 2.3. Since

$$\frac{\left\| p_9 A^9 + p_8 A^8 + p_7 A^7 \right\|}{\left\| e^A \right\|} \leq \frac{\|F\| \left\| A^3 \right\|^2}{e^{-\|A\|}}, \tag{2.28}$$

if

$$\frac{\min\{\|F\|, \|F + p_6 I\|\} \left\| A^3 \right\|^2}{e^{-\|A\|}} < u, \tag{2.29}$$

or

$$\|F\| < |p_6| u, \tag{2.30}$$

then matrix F or matrix $F + p_6 I$ can be neglected saving one matrix product. This is likely to occur for instance if the norm of matrix A was only slightly greater than θ_6. Similar tests can be devised and applied recursively, eliminating sets of 3 terms each time. It is important to note that if the condition is accomplished more than once or twice in the evaluation of $h_m(\lambda, A)$ for a scaled matrix A, it might be a sign of overscaling.

With respect to the computational cost of the bound test, if we denote B the original unscaled matrix, then $A = B/2^s$. Therefore, when doing the tests, the norm of the original unscaled matrix and the matrix power A^3 are already obtained. Hence, making the recursive tests only involves in practice to evaluate once at the beginning the norm of A^3, the expression $ue^{-\|A\|_1} = ue^{-\|B\|_1/2^s}$, and two matrix norms for each test. The cost of these operations for $r \times r$ matrices is of order $O(r^2)$, negligible when compared to a matrix product, whose cost is of order $O(r^3)$.

Similar tests can be applied to the evaluation of Taylor approximation of orders $m = 4, 6, 9, 16, 20$, and $h_m(\lambda, A)$ with $m = 25, 30$, giving Algorithm 2.2 (`dgeexfhrp`) which computes, for double precision general matrices, the exponential function by scaling-squaring Hermites approximation reducing the number of matrix products when possible. This algorithm has essentially the same stages as Algorithm 2.1, checking the tests when needed and potentially saving matrix products. Numerical tests will show that in practically the 100 % of the test matrices the savings are obtained with the same accuracy in double precision arithmetic. Both algorithms 2.1 and 2.2 do not consider orders $m = 1$ and 2 because $\|A\|$ should be very small to use those orders, given the low values of θ_1 and θ_2, see Table 2.1.

Algorithm 2.1 can be divided into the following stages (algorithm 2.2 can be divided in analogous stages):

1. Preprocessing of matrix A using the techniques proposed by Ward in [War77] (steps 1-4). Note that in numerical tests we did not use preprocessing because turning it on provided similar comparison results to those turning it off.

2. After preprocessing, the optimal value of the scaling parameter s is calculated (steps 5-20).

3. In steps 17 and 35-37 the matrix scaling and the squaring of the approximation is done, respectively.

4. Finally in step 38 the postprocessing is applied. In the same way as preprocessing, this step has not been applied in numerical tests.

We have made available online a MATLAB sequential version of the complete algorithm `dgeexfhrp` at http://personales.upv.es/jorsasma/dgeexfhrp.zip, which implements the rest of cases $m = 9, 12, \ldots, 30$ from lines 31−34 of the algorithm, and offers the possibility to select the maximum order m.

Algorithm 2.1 computes the exponential of a matrix by Hermite series with scaling and squaring and maximum order $m = 30$.

Function $F = \texttt{dgeexfher}(A)$
Input: Matrix $A \in \mathbb{C}^{r\times r}$ **Output**: Matrix $F \cong e^A \in \mathbb{C}^{r\times r}$
1: $\mu = \text{trace}(A)/r$
2: $A \leftarrow A - \mu I$
3: Determine a diagonal matrix D and a permutation matrix P such that $D^{-1}P^TAPD$ is balanced
4: $A \leftarrow D^{-1}P^TAPD$ ▷ Preprocessing of A
5: Initialize θ with values of Tables 2.5, and coefficients p_i for each approximation order $m = 4, 6, 9, \cdots, 30$.
6: $normA = \|A\|_1$
7: **for** m = [4 6 9 12 16 20 25 30] **do**
8: **if** $normA \leq \theta_m$ **then**
9: break
10: **end if**
11: **end for**
12: **if** $normA > \theta_{30}$ **then**
13: $s = \log_2(normA/\theta_{30})$
14: **if** $normA/2^s \leq \theta_{25}$ **then** ▷ This condition can occur because $\theta_{30}/2 < \theta_{25}$
15: $m = 25$
16: **end if**
17: $A \leftarrow A/2^s$ ▷ Scaling Phase: matrix A is scaled
18: **else**
19: $s = 0$
20: **end if**
21: $A_2 = A^2$
22: **if** $m == 4$ **then** ▷ Taylor approximation for m = 4, 6, 9, 12, 16, 20
23: $F = A_2\,(p_4A_2 + p_3A + p_2I) + p_1A + p_0I$
24: **else if** $m == 6$ **then**
25: $F = A_2\,(A_2\,(p_6A_2 + p_5A + p_4I) + p_3A + p_2I) + p_1A + p_0I$
26: **else if** $m == 9$ **then**
27: $A_3 = A \cdot A_2$
28: $F = A_3\,(A_3\,(p_9A_3 + p_8A_2 + p_7A + p_6I) + p_5A_2 + p_4A + p_3I) + p_2A_2 + p_1A + p_0I$
29: **else if** $m == 12$ **then**
30: .....
31: .....
32: **else if** $m == 30$ **then**
33: .....
34: **end if**
35: **for** $k = 1 : s$ **do** ▷ Squaring Phase: Repeated squaring of matrix F
36: $F = F^2$
37: **end for**
38: $F \leftarrow e^{\mu}PDFD^{-1}P^T$ ▷ Postprocessing of F

Algorithm 2.2 computes the exponential of a matrix by scaling and squaring Hermite approach with maximum order $m = 30$ reducing the number of matrix products when possible.

Function F = `dgeexfhrp`(A)
Input: Matrix $A \in \mathbb{C}^{r\times r}$ **Output**: Matrix $F \cong e^A \in \mathbb{C}^{r\times r}$
1: Same as `dgeexfher`(A) lines 1-22
2: $u = 2^{-53}$
3: $LowBound = ue^{-normA/2^s}$
4: **if** $m == 4$ **then**
5: $F = p_4A_2 + p_3A$
6: $aux = \|F\|_1$
7: $F = F + p_2I$
8: **if** $(aux < |p_2|u)$ or $(\text{mín}\{aux, \|F\|_1\} \cdot \|A_2\|_1 < LowBound)$ **then**
9: $F = p_2A_2 + p_1A + p_0I$ ▷ One matrix product is saved
10: **else**
11: $F = F \cdot A_2 + p_1A + p_0I$
12: **end if**
13: **else if** $m == 6$ **then**
14: $normA_2 = \|A_2\|_1$
15: $F = p_6A_2 + p_5A$
16: $aux = \|F\|_1$
17: $F = F + p_4I$
18: **if** $(aux < |p_4|u)$ or $(\text{mín}\{aux, \|F\|_1\} \cdot \|A_2\|_1^2 < LowBound)$ **then**
19: $F = p_4A_2 + p_3A$ ▷ One matrix product is saved
20: **else**
21: $F = F \cdot A_2 + p_3A$
22: **end if**
23: $aux = \|F\|_1$
24: $F = F + p_2I$
25: **if** $(aux < |p_2|u)$ or $(\text{mín}\{aux, \|F\|_1\} \cdot \|A_2\|_1 < LowBound)$ **then**
26: $F = p_2A_2 + p_1A + p_0I$ ▷ One matrix product is saved
27: **else**
28: $F = F \cdot A_2 + p_1A + p_0I$
29: **end if**
30: **else if** $m == 9$ **then**
31: .....
32: .....
33: **else if** $m == 30$ **then**
34: .....
35: **end if**
36: Same as `dgeexfher`(A) lines 35-38

2.2.4. Numerical examples

The main objective of this section is to compare MATLAB implementations of the algorithms developed in Section 2.2.3 with other efficient algorithms implemented in MATLAB that compute matrix exponential. MATLAB 7.7 (R2008b) implementations were tested on an Intel Core 2 Duo processor at 2.52 GHz with 4 GB main memory. In the comparative the following MATLAB functions were used:

- `Expm` is a MATLAB function that uses Padé approximants of exponential function with scaling and squaring proposed by Higham in [Hig05].
- `Funm` is a built-in MATLAB 7.7 function that enables computation of general matrix functions at square matrices. The matrix function must have a Taylor series with an infinite radius of convergence, except for the matrix logarithm, which is treated as a special case. The exponential, cosine, sine, hyperbolic sine, hyperbolic cosine and

the logarithm of a matrix are all allowed. This function implements the Schur-Parlett algorithm of Davies and Higham [DH03].

Algorithm accuracy was tested by computing the relative error

$$\mathrm{E} = \frac{\|e^A - \tilde{Y}\|_1}{\|e^A\|_1},$$

where $\tilde{Y}$ is the computed solution and e^A the exact solution.

As it is mentioned above, in the tests we did not use any preprocessing/postprocessing in the Hermite implemented algorithms. Analogously to the experiments in [Hig05], we found that turning on preprocessing in this algorithm provided similar results to those presented in this section without preprocessing.

Regarding memory issues, it is important to note that Algorithm 2.1 needs the same matrices in memory as `expm` when both methods use their maximum orders, $m = 30$ and $m = 13$ respectively: $A, A^2, \ldots, A^5$ plus one to perform the calculation for Hermite method, and A, A^2, A^4, A^6 plus two for the numerator and denominator for Padé method, taking into account that the final rational approximation can be performed re-using the memory allocated for the power of A involved in the numerator and denominator computation.

Regarding computational cost, from Table 2.4 and Table 2.3 of [Hig05], Table 2.5 presents the orders of the approximation, the θ_m and θ'_m values and the number of matrix products π_m and π'_m required for Hermite `dgeexfher` function, and Padé `expm` function, respectively. Note that the number of matrix products for `dgeexfher` is a maximum bound of the matrix products for `dgeexfhrp`. Then, using (2.24), for matrices with $\|A\| > \theta'_{13} = 5.37$ (showing three significant digits), the cost of `dgeexfher` in terms of matrix products, denoted by C_m^H, and representing the maximum cost of `dgeexfhrp`, is

$$C_{30}^H = 9 + s_H = 9 + \lceil \log_2 \|A\| - \log_2(3.58) \rceil, \text{ if } \frac{\|A\|}{2^{s_H}} > 2.44, \tag{2.31}$$

$$C_{25}^H = 8 + s_H = 8 + \lceil \log_2 \|A\| - \log_2(3.58) \rceil, \text{ if } \frac{\|A\|}{2^{s_H}} \leq 2.44, \tag{2.32}$$

where s_H denotes the scaling in Hermite methods. On the other hand, the cost of `expm` Padé method, denoted by C_m^P, is

$$C_{13}^P = 6 + C_{LS} + s_P = 6 + C_{LS} + \lceil \log_2 \|A\| - \log_2(5.37) \rceil, \tag{2.33}$$

where

$$s_P = \lceil \log_2 \|A\| - \log_2(5.37) \rceil, \tag{2.34}$$

denotes the scaling in `expm` and C_{LS} denotes the cost of solving the multiple right-hand sides linear system in Padé method, in terms of matrix products. From [BD99] the cost of the matrix product in $\mathbb{R}^{r \times r}$ and the solution of the multiple right-hand sides of the same size with Padé approximants is $2r^3 - r^2$ and $\frac{8r^3}{3} - \frac{r^2}{2} + \frac{5r}{6}$ flops, respectively. Therefore, asymptotically $C_{LS} \approx 4/3$.

Taking into account that θ_{30} for Hermite methods is greater than $\theta'_{13}/2$ for Padé method [Hig05, p. 1186], taking `expm`'s matrix scaling by 2^{s_P}, for matrices with

$$5.37/2 = 2.68 < \frac{\|A\|}{2^{s_P}} \leq 3.58, \tag{2.35}$$

Hermite methods use $m = 30$ and $s_H = s_P$, therefore `dgeexfhrp` needs a maximum of $3 - C_{LS}$ more matrix products than `expm`, which results in a maximum relative higher cost of $(1+2/3)/(7+1/3+s_P)\times 100\,\% \leq 20\,\%$ in matrix products, which decreases with the matrix norm because of the increasing scaling. For matrices such that

$$3.58 < \frac{\|A\|}{2^{s_P}} \leq 2 \times 2.44 = 4.88, \tag{2.36}$$

Hermite methods use $m = 25$ and $s_H = s_P + 1$, therefore `dgeexfhrp` needs a maximum of $3 - C_{LS}$ more matrix products than `expm` again, resulting in the same relative higher cost $(1+2/3)/(7+1/3+s_P)\times 100\,\% \leq 20\,\%$ in matrix products, decreasing with the matrix norm. In fact, using (2.35) and (2.36), for instance for matrices with norm

$$343.80 < \|A\| \leq 624.98, \tag{2.37}$$

it follows that $s_P = 7$ and the maximum relative higher cost of `dgeexfhrp` decreases to $11.63\,\%$. Finally, for matrices such that

$$4.88 < \frac{\|A\|}{2^{s_P}} \leq 5.37, \tag{2.38}$$

Hermite methods use $m = 30$ and $s_H = s_P + 1$, therefore `dgeexfhrp` needs a maximum of $4 - C_{LS}$ more matrix products than `expm`, resulting in a maximum relative higher cost of $(2 + 2/3)/(7 + 1/3 + s_P) \times 100\,\% \leq 32\,\%$ in matrix products. Note that this norm interval represents only the $18.21\,\%$ of the total interval considered in the three cases (2.35), (2.36) and (2.38).

The cost comparison for matrices with $\|A\| \leq 5.37$ is presented in Table 2.6, where the θ_m values are presented with three significant digits, and C_m^H represents a bound on the maximum cost of `dgeexfhrp`. Note that there are some cases where the maximum cost for `dgeexfhrp` is lower than the cost for `expm`, i.e. $2.53e-1 < \|A\| \leq 2.99e-1$, $1.49e-2 < \|A\| \leq 8.95e-2$ and $\|A\| \leq 9.06e-3$, reaching relative efficiency gains from $6.25\,\%$ up to $40\,\%$. For matrices satisfying $4.88 \leq \|A\| \leq 5.37$ the maximum cost of `dgeexfhrp` is $36.36\,\%$ higher and in the rest of cases `dgeexfhrp` maximum cost exceeds `expm` cost in $2/3$ or $1 + 2/3$ matrix products. Once again the interval where the cost difference is higher is a small part of all the interval considered, representing only the $9.11\,\%$ of the total. One more final squaring step than in `expm` is required for matrices satisfying $3.58 < \|A\|/2^{s_P} \leq 5.37$ with $\|A\| > 5.37$, and $3.58 < \|A\| \leq 5.37$. This might be a possible source of error, especially if A is ill-conditioned. However, Hermite methods with maximum order $m = 30$ obtained better accuracy than `expm` in a high percentage of cases in numerical tests (see Table 2.8).

In a similar way, it is easy to show that, for any matrix $A \in \mathbb{C}^{r\times r}$, the maximum cost of using `dgeexfhrp` with maximum order $m = 16$ is the same as using $m = 20$. Thus, maximum order $m = 20$ should be used instead of 16 because taking into account that $\theta_{20} > \theta_{16}$, the scaling parameter s with $m = 20$ is lower in the majority of matrix norm intervals, and the squaring process might be a possible source of error. Analogously, it is also easy to check that if we use maximum orders $m = 20$ or 25 then `dgeexfhrp` presents a maximum higher cost than `expm` of $1+2/3$ matrix multiplications, instead of $2+2/3$ that `dgeexfhrp` presented with $m = 30$. On the other hand, `dgeexfhrp` with maximum orders $m = 20$ and 25 presents lower maximum cost than `expm` for the same matrix norms as `dgeexfhrp` with $m = 30$.

It is important to note that all the costs of `dgeexfhrp` presented in this analysis are maximum costs and, as we will see in tests, they may decrease considerably in practice.

	dgeexfhrp			expm			
Norm ranges	m	s_H	C_m^H	m	s_P	C_m^P	$\frac{C_m^H - C_m^P}{C_m^P}$ %
$\|\|A\|\| \leq 3.39e-4$	4	0	2	3	0	$2+C_{LS}$	-40
$3.39e-4 < \|\|A\|\| \leq 9.06e-3$	6	0	3	3	0	$2+C_{LS}$	-10
$9.06e-3 < \|\|A\|\| \leq 1.49e-2$	9	0	4	3	0	$2+C_{LS}$	20
$1.49e-2 < \|\|A\|\| \leq 8.95e-2$	9	0	4	5	0	$3+C_{LS}$	-7.69
$8.95e-2 < \|\|A\|\| \leq 2.53e-1$	12	0	5	5	0	$3+C_{LS}$	15.38
$2.53e-1 < \|\|A\|\| \leq 2.99e-1$	12	0	5	7	0	$4+C_{LS}$	-6.25
$2.99e-1 < \|\|A\|\| \leq 7.80e-1$	16	0	6	7	0	$4+C_{LS}$	12.50
$7.80e-1 < \|\|A\|\| \leq 9.50e-1$	20	0	7	7	0	$4+C_{LS}$	31.25
$9.50e-1 < \|\|A\|\| \leq 1.43$	20	0	7	9	0	$5+C_{LS}$	10.53
$1.43 < \|\|A\|\| \leq 2.09$	25	0	8	9	0	$5+C_{LS}$	26.32
$2.09 < \|\|A\|\| \leq 2.44$	25	0	8	13	0	$6+C_{LS}$	9.09
$2.44 < \|\|A\|\| \leq 3.58$	30	0	9	13	0	$6+C_{LS}$	22.73
$3.58 < \|\|A\|\| \leq 2 \times 2.44$	25	1	9	13	0	$6+C_{LS}$	22.73
$2 \times 2.44 < \|\|A\|\| \leq 5.37$	30	1	10	13	0	$6+C_{LS}$	36.36

Table 2.6: Comparison of maximum theoretical cost C_m^H in terms of matrix products for `dgeexfhrp` with maximum order $m = 30$, and cost C_m^P for `expm`, for $\|\|A\|\| \leq 5.37$.

Table 2.7: Relative error comparison (%) between `dgeexfher` and `dgeexfhrp`.

maximum order	m=16	m=20	m=25	m=30
$E_{\texttt{dgeexfher}} < E_{\texttt{dgeexfhrp}}$	3.77	0	0	0
$E_{\texttt{dgeexfher}} = E_{\texttt{dgeexfhrp}}$	95.28	99.06	100.00	100.00
$E_{\texttt{dgeexfher}} > E_{\texttt{dgeexfhrp}}$	0.94	0.94	0.00	0.00
$P_{\texttt{dgeexfher}}/P_{\texttt{dgeexfhrp}}$	105.52	108.43	108.58	109.91

For tests, 105 matrices were used: 49 matrices from the Matrix Computation Toolbox [Hig93], 24 matrices from the Eigtool MATLAB package [Wri02], 18 matrices from papers of the state-of-the-art of matrix functions [ML03, War77, DH03, NH95, KL98, Wes90, Lu03, DP00], and 14 special matrices such as matrices of Vandermonde, Hankel, Toeplitz, Wilkinson, symmetric matrices, defective matrices and non defective matrices.

In the examples the matrix exponentials were calculated analytically, when it was possible, or by using [33/33] diagonal Padé method with scaling and squaring with 1000-digit precision in an iterative way: different increasing scalings starting from that provided in [Hig05] for `expm` were used, until the norm of the relative difference between the approximations converted to IEEE double precision arithmetic was zero in four iterations. The [33/33] diagonal Padé approximation was evaluated with matrix power aggregation similar to that proposed in [Hig05, p. 1183].

Table 2.7 shows a comparative between the implementations `dgeexfher` and `dgeexfhrp`. The three first rows contain the percentage of times that relative error of a function is lower, equal or greater than relative error of other function. The last row of Table 2.7 contains the ratio of the total number of matrix products evaluated for the two functions over all test matrices, denoted by $P_{\texttt{dgeexfher}}$ and $P_{\texttt{dgeexfhrp}}$. Both functions presented practically the same accuracy, however the total number of matrix products evaluated for `dgeexfhrp` is lower than those for `dgeexfher`.

Table 2.8: Comparison of relative error and total number of matrix products (%) between `dgeexfhrp` and `expm`. The error comparison results were exactly the same with `dgeexfher` at a greater cost (see last table row).

maximum order	m=16	m=20	m=25	m=30
$E_{\texttt{dgeexfhrp}} < E_{\texttt{expm}}$	44.34	68.87	77.36	77.36
$E_{\texttt{dgeexfhrp}} = E_{\texttt{expm}}$	1.89	2.83	1.89	1.89
$E_{\texttt{dgeexfhrp}} > E_{\texttt{expm}}$	53.77	28.30	20.75	20.75
$P_{\texttt{dgeexfhrp}}/P_{\texttt{expm}}$	106.62	103.76	104.01	104.43
$P_{\texttt{dgeexfher}}/P_{\texttt{expm}}$	112.51	112.51	112.93	114.78

Table 2.9: Relative error comparison between `dgeexfhrp` and `funm`. The results were exactly the same with `dgeexfher`.

maximum order	m=16	m=20	m=25	m=30
$E_{\texttt{dgeexfhrp}} < E_{\texttt{funm}}$	71.70	77.36	79.25	80.19
$E_{\texttt{dgeexfhrp}} = E_{\texttt{funm}}$	0.94	0	0	0
$E_{\texttt{dgeexfhrp}} > E_{\texttt{funm}}$	28.30	22.64	20.75	19.81

Table 2.8 presents the relative error and matrix product comparison of `dgeexfhrp` and `dgeexfher` with `expm`. `dgeexfher` obtained the same comparative results of accuracy as `dgeexfhrp` at a higher cost, see the last two rows. `dgeexfhrp` relative error is lower than `expm` relative error for $m \geq 20$ (68.87 %-77.36 %), with a slightly higher cost, varying the ratio of matrix products between 103.76 % and 104.43 % for $m = 20, 25, 30$.

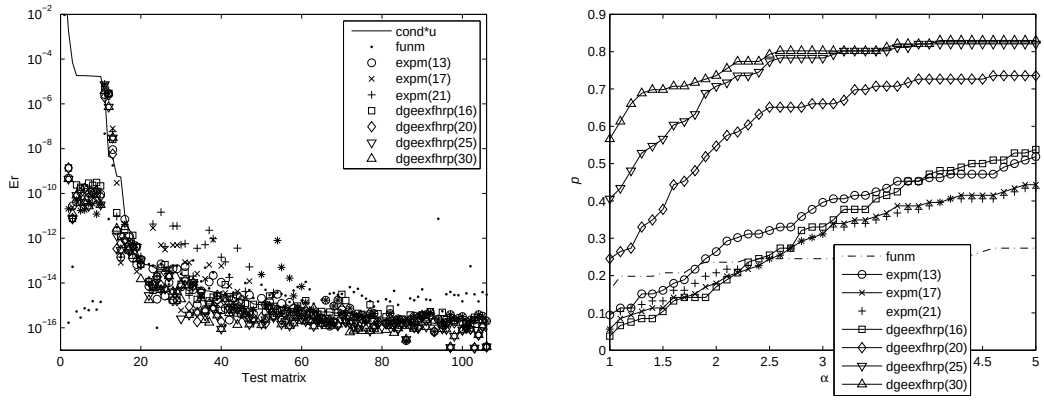
In Table 2.9 the relative errors of `dgeexfhrp` and `dgeexfher` are compared to `funm`. As shown in these tables, once again `dgeexfher` and `dgeexfhrp` obtained the same comparative results of accuracy, and in general their relative errors were lower than the relative errors of `funm` (71.70 %-80.19 %).

Figure 2.2a presents the normwise relative errors of `dgeexfhrp`, `expm` and `funm` (a similar figure is obtained when `dgeexfher` is used). This figure shows the relative error of all implementations sorted in decreasing order, and a solid line that represents the unit roundoff multiplied by the relative condition number of the exponential function at X [Hig08, p. 56],

$$\mathrm{cond}_{\exp}(X) = \lim_{\varepsilon \to 0} \sup_{\|E\| \leq \varepsilon} \frac{\left\| e^{X+E} - e^{X} \right\|}{\varepsilon} \frac{\|X\|}{\|e^{X}\|}.$$

Relative condition number was computed using the MATLAB function `expm_cond`. This function is incorporated into the Matrix Function Toolbox developed by Higham [Hig08, Appendix D] and available at `http://www.ma.man.ac.uk/~higham/mftoolbox`.

For a method to perform in a backward and forward stable manner, its error should lie not far above this line on the graph [Hig05, p. 1188]. Figure 2.2a shows that all functions perform in a numerically stable way on this test, even for matrices 64-70 where there were overscaling problems [DP00].



(a) Normwise relative errors

(b) Performance profile

Figure 2.2: Comparative of `dgeexfhrp`, `expm` and `funm`.

Performance profile [DM02] is presented in Figure 2.2b. This figure shows the performances of the compared functions, where α coordinate varies between 1 and 5 in steps equal to 0.1, and p coordinate is the probability that the considered function has a relative error lower or equal than α-times the smallest error over all the methods. The probabilities are defined over all matrices considered in the tests. As shown in this figure, `dgeexfhrp` with maximum order $m = 30$ is the most accurate function, and it was achieved with very similar cost to `dgeexfhrp` with maximum orders $m = 16, 20$ or 25. Hence, we consider $m = 30$ as the best choice of maximum order for `dgeexfhrp`.

2.2.5. Conclusions

In this work an efficient method to approximate the matrix exponential based on Hermite matrix polynomial expansions has been presented. Following the ideas of [Hig05] we have developed an optimal backward error bound for the scaling and squaring Hermite method in exact arithmetic, which depends on A only through $\|A\|$ and enables to obtain the theoretical optimal scaling for general matrices. The optimal parameter λ of the algorithm has been obtained for each order $m = 21, 22, \ldots, 30$, providing greater values of the Hermite scaling parameter θ_m than Taylor methods for those orders, i.e. $\theta_m > \theta_m^\infty$, $m = 21, 22, \ldots, 30$, where θ_m^∞ is the scaling parameter for Taylor methods, see Tables 2.1 and 2.2. Based on that result, two mixed Hermite-Taylor algorithms have been developed in order to evaluate different order matrix polynomial approximations: a Hermite-Taylor series Paterson-Stockmeyer algorithm, `dgeexfher`, and a modified algorithm, `dgeexfhrp`, which taking into account floating point arithmetic error bounds may reduce the number of matrix product evaluations. This modification (`dgeexfhrp`) presented practically the same accuracy as `dgeexfher` in numerical tests with a lower computational cost of up to 9.91 %. Both algorithms use Hermite series for order $m > 20$ and Taylor series for order $m \leq 20$. From experimental results, we have identified the most efficient choice of maximum degree m of Hermite approximation in terms of efficiency and accuracy: $m = 30$.

We have shown that `dgeexfhrp` with maximum order $m = 30$ has lower theoretical maximum cost than `expm` for some matrix norm intervals, and in numerical tests the cost of `dgeexfhrp` was similar to that for `expm`.

`dgeexfhrp` stores the same number of matrices in memory as `expm` when both functions use their maximum orders, $m = 30$ and $m = 13$ respectively, and does not need the solution of multiple linear systems. [Hig05] shows that this solution does not introduce large errors in `expm`. However, `dgeexfhrp` obtained higher accuracy than both `funm` and `expm` in the majority of test matrices, i.e. the 80.19 % and 77.36 % respectively. These results are based on empirical observations. However, numerical results are promising and further research on Hermite matrix polynomial series for the matrix exponential is being carried out to reduce the computational costs.

Acknowledgements

This work has been supported by the Programa de Apoyo a la Investigación y el Desarrollo of the Universidad Politécnica de Valencia PAID-05-09-4338, 2009.

2.3. Accurate matrix exponential computation to solve coupled differential models in Engineering

Referencia del artículo:
J. Sastre, J. Ibáñez, E. Defez, and P. Ruiz
Accurate matrix exponential computation to solve coupled differential models in engineering
Mathematical and Computer Modelling, Volume 54, Issues 7–8, October 2011, Pages 1835–1840, ISSN 0895-7177, http://dx.doi.org/10.1016/j.mcm.2010.12.049

Abstract

The matrix exponential plays a fundamental role in linear systems arising in engineering, mechanics and control theory. This work presents a new scaling-squaring algorithm for matrix exponential computation. It uses forward and backward error analysis with improved bounds for normal and nonnormal matrices. Applied to the Taylor method, it has presented a lower or similar cost compared to the state-of-the-art Padé algorithms with better accuracy results in the majority of test matrices, avoiding Padé's denominator condition problems.

2.3.1. Introduction

Many engineering processes are described by systems of linear first-order ordinary differential equations with constant coefficients, whose solutions are given in terms of the matrix exponential, and a large number of methods for its computation have been proposed [ML03, Hig08]. This work presents a competitive new scaling and squaring algorithm for matrix exponential computation. Throughout this paper $\mathbb{C}^{n\times n}$ denotes the set of complex matrices of size $n\times n$, I denotes the identity matrix for this set, $\rho(A)$ is the spectral radius of matrix A, and $\mathbb{N}$ denotes the set of positive integers. The matrix norm $\|\cdot\|$ denotes any subordinate matrix norm; in particular $\|\cdot\|_1$ is the 1-norm. This paper is organized as follows. Section 2.3.2 presents the scaling and squaring error analysis and the developed algorithm, and Section 2.3.3 deals with numerical tests and conclusions. Next theorem will be used in next section to bound the norm of matrix power series.

Theorem 2 *Let $h_l(x) = \sum_{k\geq l} b_k x^k$ be a power series with radius of convergence w, and let $\tilde{h}_l(x) = \sum_{k\geq l} |b_k| x^k$. For any matrix $A \in \mathbb{C}^{n\times n}$ with $\rho(A) < w$, if a_k is an upper bound for $\|A^k\|$ ($\|A^k\| \leq a_k$), $p \in \mathbb{N}$, $1 \leq p \leq l$, and $\alpha_p = \max\{(a_k)^{\frac{1}{k}} : k = p, l, l+1, \ldots, l+p-1\}$, then $\|h_l(A)\| \leq \tilde{h}_l(\alpha_p)$. If $p = 2$ and l is odd the same bound holds taking $\alpha_2 = \max\{(a_k)^{\frac{1}{k}} : k = 2, l\}$.*

Proof. For the first part note that

$$\|h_l(A)\| \leq \sum_{j\geq 0} \sum_{i=l}^{l+p-1} |b_{i+jp}| \|A^p\|^j \|A^i\| \leq \sum_{j\geq 0} \sum_{i=l}^{l+p-1} |b_{i+jp}| \alpha_p^{i+jp} = \sum_{k\geq l} |b_k| \alpha_p^k = \tilde{h}_l(\alpha_p). \tag{2.39}$$

If $p = 2$ and l is odd note that if $k \geq l$ is odd then $k = 2j+1$, $j \in \mathbb{N}$, and one gets $\|A^k\| = \|A^{2j+1}\| \leq \|A^l\|\|A^2\|^{\frac{2j+1-l}{2}} \leq a_l a_2^{\frac{2j+1-l}{2}} = \left(a_l^{1/l}\right)^l \left(a_2^{1/2}\right)^{2j+1-l} \leq \max\left\{a_l^{1/l}, a_2^{1/2}\right\}^{2j+1} = \alpha_2^k$, and for even $k > l$, $k = 2j$, $j \in \mathbb{N}$, $\|A^k\| \leq \|A^2\|^j \leq \left(a_2^{1/2}\right)^{2j} \leq \alpha_2^k$. Hence

$$\|h_l(A)\| \leq \sum_{k\geq l} |b_k|\|A^k\| \leq \sum_{k\geq l} |b_k|\alpha_2^k = \tilde{h}_l(\alpha_2). \quad \square \tag{2.40}$$

2.3.2. Error analysis and algorithm

If we denote the truncated matrix exponential Taylor series as $T_m(A) = \sum_{i=0}^m A^i/i!$, and its remainder as $R_m(A) = \sum_{i\geq m+1} A^i/i!$, for a scaled matrix $2^{-s}A$, $s \in \mathbb{N}\cup\{0\}$, see [SIDR11b], we can write

$$\left(T_m(2^{-s}A)\right)^{2^s} = e^A \left(I + g_{m+1}(2^{-s}A)\right)^{2^s} = e^{A+2^s h_{m+1}(2^{-s}A)}, \tag{2.41}$$

$$g_{m+1}(2^{-s}A) = -e^{-2^{-s}A} R_m(2^{-s}A),\ h_{m+1}\left(2^{-s}A\right) = \log\left(I+g_{m+1}(2^{-s}A)\right), \tag{2.42}$$

where log denotes the principal logarithm, $h_{m+1}(X)$ is defined in the set $\Omega_m = \left\{X \in \mathbb{C}^{n\times n} : \rho\left(e^{-X}T_m(X) - I\right) < 1\right\}$, see [AMH09, sec. 3], and both $g_{m+1}(2^{-s}A)$ and $h_{m+1}\left(2^{-s}A\right)$ are holomorphic functions of A in Ω_m and then commute with A. If we choose s so that $2^{-s}A \in \Omega_m$, then from (2.41) one gets that $\Delta A = 2^s h_{m+1}\left(2^{-s}A\right)$ and $\Delta E = e^A\left[\left(I + g_{m+1}\left(2^{-s}A\right)\right)^{2^s} - I\right]$ represent the backward and forward errors in exact arithmetic from the approximation of e^A by the Taylor series with scaling and squaring, respectively. If s is chosen so that

$$\left\|h_{m+1}\left(2^{-s}A\right)\right\| \leq \max\left\{1, \left\|2^{-s}A\right\|\right\} u, \tag{2.43}$$

where $u = 2^{-53}$ is the unit roundoff in IEEE double precision arithmetic, then: if $2^{-s}\|A\| \geq 1$, then $\Delta A \leq \|A\| u$ and using (2.41) one gets $\left(T_m\left(2^{-s}A\right)\right)^{2^s} = e^{A+\Delta A} \approx e^A$, and if $2^{-s}\|A\| < 1$, using (2.41)-(2.43) and the Taylor series one gets

$$\begin{aligned}\left\|R_m\left(2^{-s}A\right)\right\| &= \left\|e^{2^{-s}A} g_{m+1}\left(2^{-s}A\right)\right\| = \left\|e^{2^{-s}A}\left(e^{h_{m+1}\left(2^{-s}A\right)} - I\right)\right\| \\ &= \left\|e^{2^{-s}A}\sum_{k\geq 1}\left(h_{m+1}\left(2^{-s}A\right)\right)^k/k!\right\| \leq \left\|e^{2^{-s}A}\right\| \sum_{k\geq 1} u^k/k! \\ &\approx \left\|T_m\left(2^{-s}A\right)\right\| u\left(1+u/2!+u^2/3!+\cdots\right) \approx \left\|T_m\left(2^{-s}A\right)\right\| u.\end{aligned} \tag{2.44}$$

Hence, as we will evaluate explicitly $T_m\left(2^{-s}A\right)$, by (2.44) one gets $T_m\left(2^{-s}A\right) + R_m\left(2^{-s}A\right) \approx T_m\left(2^{-s}A\right)$, and there is no need to increase m or the scaling parameter s to try to get better accuracy. Using the Taylor series in (2.42) one gets

$$g_{m+1}(x) = \sum_{k\geq m+1} b_k^{(m)} x^k, \quad h_{m+1}(x) = \sum_{k\geq 1} \frac{(-1)^{k+1}(g_{m+1}(x))^k}{k} = \sum_{k\geq m+1} c_k^{(m)} x^k, \tag{2.45}$$

where $b_k^{(m)}$ and $c_k^{(m)}$ depend on order m, and $b_k^{(m)} = c_k^{(m)}, k = m+1, m+2, \ldots, 2m+1$. Using MATLAB symbolic Math Toolbox, high precision arithmetic, 200 series terms and a zero finder we obtained the maximal values Θ_m of $\Theta = \left\|2^{-s}A\right\|$, shown in Table 2.21, such that, using the notation of Theorem 2

$$\left\|h_{m+1}\left(2^{-s}A\right)\right\| \leq \tilde{h}_{m+1}(\Theta) = \sum_{k\geq m+1} |c_k^{(m)}|\Theta^k \leq \max\{1,\Theta\}\, u. \tag{2.46}$$

Hence, if $\|2^{-s}A\| \leq \Theta_m$ then (2.43) holds. For the cases where $\Theta_m > 1$, note that $f(\Theta) = \tilde{h}_{m+1}(\Theta) - \Theta u$ is a continuous function in $[0, \Theta_m]$ and $f(\Theta_m) = 0$, $f(0) = 0$. For $m = 20, 25, 30$ we have checked that there are no other zeros in $[0, \Theta_m]$, and $f(\Theta) < 0$, $\Theta \in]0, \Theta_m[$. Thus, for those orders the next bound holds

$$\|h_{m+1}(2^{-s}A)\| \leq \tilde{h}_{m+1}(\|2^{-s}A\|) = \tilde{h}_{m+1}(\Theta) \leq \Theta u, \quad 0 \leq \Theta \leq \Theta_m. \tag{2.47}$$

In Section 2.3.2.2 we will obtain an initial maximum value of the scaling parameter s, denoted by s_0, using values Θ_m, Theorem 2, (2.47) and the powers of A computed for the evaluation of $T_m(2^{-s}A)$ which we analyze in next subsection.

2.3.2.1. Taylor matrix polynomial evaluation

For the evaluation of $T_m(2^{-s}A)$ we have improved the Horner's and Paterson-Stockmeyer's method of [SIDR11b] calculating matrix powers $A_i = A^i, i = 2, 3, \ldots, q$ in the same way, but including the scaling in the Taylor series coefficients and saving some divisions of matrix A by scalar as follows:

$$\begin{aligned} T_m(2^{-s}A) = \Bigg\{\Bigg\{\cdots\Bigg\{\frac{A_q}{2^s m} + A_{q-1}\Bigg\}/[2^s(m-1)] + A_{q-2}\Bigg\}/[2^s(m-2)] + \cdots + A_2\Bigg\}/[2^s(m-q+2)] + A \\ + 2^s(m-q+1)I\Bigg\}\frac{A_q}{2^{2s}(m-q+1)(m-q)} + A_{q-1}\Bigg\}/[2^s(m-q-1)] + A_{q-2}\Bigg\} \\ /\,[2^s(m-q-2)] + \cdots + A_2\Bigg\}/[2^s(m-2q+2)] + A + 2^s(m-2q+1)I\Bigg\} \\ \times \frac{A_q}{2^{2s}(m-2q+1)(m-2q)} + \cdots + A_2\Bigg\}/[2^s(q+2)] + A + 2^s(q+1)I\Bigg\} \\ \times \frac{A_q}{2^{2s}(q+1)q} + A_{q-1}\Bigg\}/[2^s(q-1)] + \cdots + A_2\Bigg\}/[2^s 2] + A\Bigg\}/2^s + I. \end{aligned} \tag{2.48}$$

Note that the matrix powers A_i will be obtained before the optimal scaling s is calculated and with this formula it is not necessary to calculate explicitly and save scaled matrices $A_i/2^{si} \to A_i, i = 1, 2, \ldots, q$. We will use the optimal values of m in terms of the number of evaluations of matrix products $m_k = [1, 2, 4, 6, 9, 12, 16, 20, 25, 30]$, $k = 0, 1, \ldots, 9$, respectively, see [SIDR11b]. If the maximum allowed order, denoted by m_M, is 25 or 30, we will take $q = [1, 2, 2, 3, 3, 4, 4, 5, 5, 5]$ for each value of m_k, respectively, because in the scaling algorithm it will be necessary that the two last orders m_{M-1} and m_M use the same matrix powers of A, i.e. $A^i, i = 2, 3, \ldots, q$. If the maximum allowed order is $m_M = 20$ we will use $q = 4$ for that order for the same reason. Counting the number of evaluations of matrix products in (2.48), denoted by Π_{m_k}, including those for obtaining matrix powers A^i, $i = 2, 3, \ldots, q$, for the proposed values of m_k and q we have that using (2.48), $T_{m_k}(2^{-s}A)$ is evaluated in $\Pi_{m_k} = k$ matrix products. Similar rounding error bounds to those in [SIDR11b] could be applied to the intermediate results in $T_{m_k}(2^{-s}A)$ to try to save matrix products.

2.3.2.2. Scaling algorithm

For all norms appearing in the scaling algorithm we will use the 1-norm. Let m_M be the maximum allowed Taylor order. Using the same bounds and process that we will use in the proposed scaling algorithm described below, we will first check if any of the Taylor optimal orders $m_k = 1, 2, 4, \ldots, m_{M-1}$ satisfy (2.43) without scaling, i.e. with $s = 0$. If not,

Table 2.10: Maximal values $\Theta_m = \|2^{-s}A\|$ such that $\tilde{h}_{m+1}(\Theta_m) \le \max\{1, \Theta_m\}u$, coefficient ratios $c_k^{(m)}/c_{m+2}^{(m)}$ for the first values of $k \ge m+1$, and values $u/|c_{m+2}^{(m)}|$.

		$c_k^{(m)}/c_{m+2}^{(m)}$				$\frac{u}{c_{m+2}^{(m)}}$
m	Θ_m	$m+1$	$m+3$	$m+4$	$m+5$	
1	1.490116111983279e-8	-3/2	-3/4	3/5	-1/2	3.3e-16
2	8.733457513635361e-6	-4/3	-2/5	0	1/7	8.9e-16
4	1.678018844321752e-3	-6/5	-3/7	1/8	-1/36	1.6e-14
6	1.773082199654024e-2	-8/7	-4/9	2/15	-1/33	6.4e-13
9	1.137689245787824e-1	-11/10	-11/24	11/78	-11/336	4.4e-10
12	3.280542018037257e-1	-14/13	-7/15	7/48	-7/204	7.5e-7
16	7.912740176600240e-1	-18/17	-9/19	3/20	-1/28	4.2e-2
20	1.438252596804337	-22/21	-11/23	11/72	-11/300	5.9e03
25	2.428582524442827	-27/26	-27/56	9/58	-3/80	4.7e10
30	3.539666348743690	-32/31	-16/33	8/51	-4/105	9.4e17

we will calculate the optimal scaling s for order m_M as follows: First, we will compute the 1-norm estimate of $\|A^{m_M+1}\|$ using the block 1-norm estimation algorithm of [HT00]. For a $n \times n$ matrix this algorithm carries out a 1-norm power iteration whose iterates are $n \times t$ matrices, where t is a parameter that has been taken to be 2, see [AMH09, p. 983]. Hence, the estimation algorithm has $O(n^2)$ computational cost, negligible compared to matrix products, whose cost is $O(n^3)$. The bounds a_k for $\|A^k\|$ needed to apply Theorem 2 in (2.46) will be obtained using the products of norms of matrix powers estimated for previous and current tested orders, $\|A^{m_k+1}\|$, $k = 0, 1, 2, \ldots, M$, and the powers of A computed for the evaluation of $T_{m_M}(2^{-s}A)$, A^i, $i = 1, 2, \ldots, q$, as

$$\begin{aligned} \left\|A^k\right\| \le a_k \;=\; & \min\Big\{\|A\|^{i_1}\left\|A^2\right\|^{i_2}\cdots\|A^q\|^{i_q}\left\|A^{m_1+1}\right\|^{i_{m_1+1}}\left\|A^{m_2+1}\right\|^{i_{m_2+1}}\cdots \\ & \times \left\|A^{m_M+1}\right\|^{i_{m_M+1}} : i_1 + 2i_2 + \cdots + qi_q + (m_1+1)i_{m_1+1} \\ & + (m_2+1)i_{m_2+1} + \cdots + (m_M+1)i_{m_M+1} = k\Big\}, \end{aligned} \tag{2.49}$$

where the minimum is desirable, but not necessary. We will obtain successively α_p value of Theorem 2 with $l = m_M + 1$ for $p = 2, 3, \ldots, q, m_1+1, m_2+1, \ldots, m_M+1$, stopping the process when $(a_p)^{1/p} \le \max\{(a_k)^{1/k} : k = m+1, m+2, \ldots, m+p\}$. We will select the minimum value of all values α_p, denoted by α_{min}. Then we will take the appropriate initial minimum scaling parameter $s_0 \ge 0$ so that $2^{-s_0}\alpha_{min} \le \Theta_{m_M}$, i.e. if $\alpha_{min} \le \Theta_{m_M}$ then $s_0 = 0$, and otherwise $s_0 = \lceil \log_2(\alpha_{min}/\Theta_{m_M}) \rceil$. Then, if $\Theta_{m_M} \le 1$ using Theorem 2 and (2.46), and taking for simplicity in the rest of the algorithm description $m = m_M$, it follows that

$$\|h_{m+1}(2^{-s_0}A)\| \le \tilde{h}_{m+1}(2^{-s_0}\alpha_{min}) \le \tilde{h}_{m+1}(\Theta_m) \le u, \tag{2.50}$$

and (2.43) holds. Taking into account that $\|A^k\|^{1/k} \le \|A\|$, from (2.49) it follows that $a_k^{1/k} \le (\|A\|^k)^{1/k} = \|A\|$. Thus, α_{min} from Theorem 2 satisfies $\alpha_{min} \le \|A\|$. Hence, if $m = 20, 25$ or 30, where $\Theta_m > 1$, using (2.47) one gets

$$\|h_{m+1}(2^{-s_0}A)\| \le \tilde{h}_{m+1}(2^{-s_0}\alpha_{min}) \le 2^{-s_0}\alpha_{min}\, u \le 2^{-s_0}\|A\|u, \tag{2.51}$$

and (2.43) also holds.

Once obtained s_0, if $s_0 \geq 1$ check if (2.43) holds reducing the scaling $s = s_0 - 1$, and using the bounds for $\left\|A^k\right\| \leq a_k$ to test if bound

$$\frac{\|h_{m+1}(2^{-s}A)\|}{|c_{m+2}^{(m)}|} \leq \sum_{k\geq m+1} \left|\frac{c_k^{(m)}}{c_{m+2}^{(m)}}\right| \frac{a_k}{2^{sk}} \leq \max\left\{1, \left\|2^{-s}A\right\|\right\} \frac{u}{|c_{m+2}^{(m)}|}, \tag{2.52}$$

holds, truncating the series. Note that we will stop the series summation if after summing one term the sum is greater than $\max\{1, \|2^{-s}A\|\}u/|c_{m+2}^{(m)}|$. This has been the case many times in numerical tests after calculating just the first series term. If the sum of one or more terms is lower than the bound but the complete truncated series sum is not, we can estimate $\|A^{m+2}\|$ to improve the bound a_{m+2} and check if (2.52) holds then. This has improved the computational cost in numerical tests. If (2.52) does not hold with $s = s_0 - 1$, then we will check if next bound holds

$$\begin{aligned}\frac{\|h_{m+1}(2^{-s}A)\|}{|c_{m+2}^{(m)}|} &\leq \frac{\left\|A^{m+1}\right\|}{2^{s(m+2)}} \left\| \frac{c_{m+1}^{(m)} 2^s I}{c_{m+2}^{(m)}} + A + \frac{c_{m+3}^{(m)} A_2}{c_{m+2}^{(m)} 2^s} + \frac{c_{m+4}^{(m)} A_3}{c_{m+2}^{(m)} 2^{2s}} + \cdots + \frac{c_{m+q+1}^{(m)} A_q}{c_{m+2}^{(m)} 2^{s(q-1)}} \right\| \\ &+ \sum_{k\geq m+2+q} \left|\frac{c_k^{(m)}}{c_{m+2}^{(m)}}\right| \frac{a_k}{2^{sk}} \leq \max\left\{1, \left\|2^{-s}A\right\|\right\} \frac{u}{|c_{m+2}^{(m)}|}\end{aligned} \tag{2.53}$$

where we will truncate the series by $k = m + N$, with $N \geq 2 + q$, and we have divided by the coefficient of A to save the product of matrix A by a scalar. We propose using at least one term of the infinite series because the norm of the previous matrix polynomial in (2.53) might vanish in some cases where the infinite series might be large, e.g. scalar A when A is a zero of the resulting scalar polynomial. For convenience we will also truncate the series in (2.52) by the same value of k, i.e. $k = m+N$. For the proposed orders m_k and the first 1000 series terms we have observed in (2.45) that $b_{m+j}^{(m)} = (-1)^j |b_{m+j}^{(m)}|$ and $1/(j+1) < |b_{m+1+j}^{(m)}/b_{m+j}^{(m)}| < 1/j$, $j \geq 1$, and then bounds for the last terms of $g_{m+1}(\|2^{-s}A\|)$ and $h_{m+1}(\|2^{-s}A\|)$ can be obtained. Table 2.10 presents some values of $c_k^{(m)}/c_{m+2}^{(m)}$, and the values $u/|c_{m+2}^{(m)}|$. Next we obtain lower bounds for expression (2.53) to avoid its unnecessary evaluation: Taking

$$T_{max} = \max\left\{ \left|\frac{c_{m+1}^{(m)}}{c_{m+2}^{(m)}}\right| 2^s, \|A\|, \left|\frac{c_{m+3}^{(m)}}{c_{m+2}^{(m)}}\right| \|A_2\|/2^s, \cdots, \left|\frac{c_{m+q+1}^{(m)}}{c_{m+2}^{(m)}}\right| \|A_q\|/2^{s(q-1)} \right\}, \tag{2.54}$$

and T_i as the other q elements of the same set, note that

$$\left\| \frac{c_{m+1}^{(m)} 2^s I}{c_{m+2}^{(m)}} + A + \frac{c_{m+3}^{(m)} A_2}{c_{m+2}^{(m)} 2^s} + \frac{c_{m+4}^{(m)} A_3}{c_{m+2}^{(m)} 2^{2s}} + \cdots + \frac{c_{m+q+1}^{(m)} A_q}{c_{m+2}^{(m)} 2^{s(q-1)}} \right\| \geq minsum, \tag{2.55}$$

where $minsum = \max\left\{0, T_{max} - \sum_{i=1}^{q} T_i\right\}$, and if

$$\frac{\left\|A^{m+1}\right\|}{2^{s(m+2)}} \times minsum + \sum_{k=m+2+q}^{m+N} \left|\frac{c_k^{(m)}}{c_{m+2}^{(m)}}\right| \frac{a_k}{2^{sk}} > \max\left\{1, \left\|2^{-s}A\right\|\right\} \frac{u}{|c_{m+2}^{(m)}|}, \tag{2.56}$$

then there is no need to evaluate bound (2.53). Using (2.56) saved many times the evaluation of (2.53) in numerical tests. Using now the 2-norm and taking into account that for normal matrices $\|A^i\|_2 = \|A\|_2^i$, $i = 2, 3, \ldots$, for any scalar coefficients $d_k \in \mathbb{R}, k = 0, 1, \ldots, q$, one gets

$$\begin{aligned}&\|A^{m+1}\|_2 \|d_0 I + d_1 A + \cdots + d_q A^q\|_2 \leq \|A^{m+1}\|_2 (|d_0| + |d_1| \|A\|_2 + \cdots \\ &+ |d_q| \|A^q\|_2) = |d_0| \|A^{m+1}\|_2 + |d_1| \|A^{m+2}\|_2 + \cdots + |d_q| \|A^{m+1+q}\|_2,\end{aligned} \tag{2.57}$$

and then using the 2-norm, the bound in (2.53) is lower or equal than the bound in (2.52) for normal matrices. As our algorithm uses 1-norm, any of (2.52) or (2.53) may be the lower

bound depending on the matrix. In the case of nonnormal matrices any of (2.52) or (2.127) may also be the lower bound. Moreover, the first $m+1$ non-zero coefficients of $h_{m+1}(x)$ have an alternating sign, see Table 2.10, and then the bound in (2.127) may be higher for matrices with all negative elements and lower for matrices with all positive elements. For instance, considering normal matrices B_1 and B_2, and nonnormal matrices B_3, B_4

$$B_1 = \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix},\ B_2 = \begin{pmatrix} 1 & 2 \\ 2 & -1 \end{pmatrix},\ B_3 = \begin{pmatrix} 1 & 25 \\ 0 & -1 \end{pmatrix},\ B_4 = \begin{pmatrix} 1 & 1 \\ 2 & 1 \end{pmatrix}, \tag{2.58}$$

and $B_5 = -B_4$, for $m = 4$ one gets $\left\|B_i^5\right\|_1 \|c_5^{(4)}/c_6^{(4)} I + B_i + c_7^{(4)}/c_6^{(4)} B_i^2\|_1 = 108.3,\ 475.7,\ 718.3, 175.4, 712.8$ and $\left\|B_i^5\right\|_1 |c_5^{(4)}/c_6^{(4)}| + \left\|B_i^6\right\|_1 + \left\|B_i^7\right\|_1 |c_7^{(4)}/c_6^{(4)}| = 387.4,\ 375.7, 43.3, 605.1,\ 605.1$, respectively, confirming that any of both bounds may be the best depending on the case. We have also obtained the corresponding α_{min} values from the scaling algorithm, for $m = 4$, using norms of matrix powers $\|B_i^2\|_1$ and $\|B_i^5\|_1$, i.e. $\alpha_{min} = 2.53, 2.37, 1.92, 2.60, 2.60$, being lower in all cases than those obtained using theorem 4.2 of [AMH09], i.e. máx$\{\|B_i^2\|_1^{1/2}$, $\|B_i^3\|_1^{1/3}\} = 2.65, 2.47, 2.96, 2.65, 2.65$, respectively.

If any of both bounds (2.52) or (2.127) is satisfied with $s_0 - 1$ then repeat the process with $s = s_0 - 2, s_0 - 3, \ldots$ Note that the computational cost of evaluating (2.52) or (2.127) is $O(n^2)$ and if any of them is satisfied with $0 \le s < s_0$ their evaluation saves matrix products, whose cost is $O(n^3)$. If the last value of scaling parameter s where (2.52) or (2.127) are satisfied is $s \ge 1$ then if $\Theta_{m_M} < 2\Theta_{m_{M-1}}$ it is possible that the same value of scaling s and order m_{M-1} also satisfy (2.52) or (2.127), see [SIDR11b], and this occurred in numerical tests. Thus, if the final resulting scaling is $s \ge 1$ we propose testing bounds (2.52) and/or (2.127) with the same value of the scaling parameter s, and order m_{M-1}. Finally, the algorithm will return s and the minimum order satisfying (2.52) or (2.127), which may be m_M or m_{M-1}. It is possible to evaluate $T_m(2^{-s}A)$ with both orders with the optimal number of matrix products at this point because we set in its evaluation that both last orders used the same matrix powers of A.

The complete matrix exponential computation algorithm will consist of: using Theorem 2, (2.52) and (2.53) check if one of orders $m = 1, 2, 4, \ldots, m_{M-1}$ satisfies (2.43) with $s = 0$. If not, obtain the values of scaling parameter s and order m using the previous algorithm, and use (2.48) and squaring to evaluate $\left(T_m\left(2^{-s}A\right)\right)^{2^s}$. We have made available online the commented MATLAB implementation of the algorithm, denoted by `exptayns`, in `http://personales.upv.es/jorsasma/exptayns.zip`.

If $\hat{T}_m(A)$ denotes the computed Taylor approximation, using error analysis techniques for the evaluation of matrix products from [Hig02a, sec. 3.5] we have that $\left\|T_m(A) - \hat{T}_m(A)\right\| \le \tilde{\gamma}_{mn} T_m\left(\|A\|\right) \le \tilde{\gamma}_{mn} e^{\|A\|}$, where $\tilde{\gamma}_k = \frac{cku}{1-cku}$, with c a small integer constant. This bound might be unsatisfactory taking into account that with the proposed scaling algorithm $\|A\|$ can be large. However, the proposed algorithm behaved in a stable way in all numerical tests.

2.3.3. Numerical experiments and conclusions

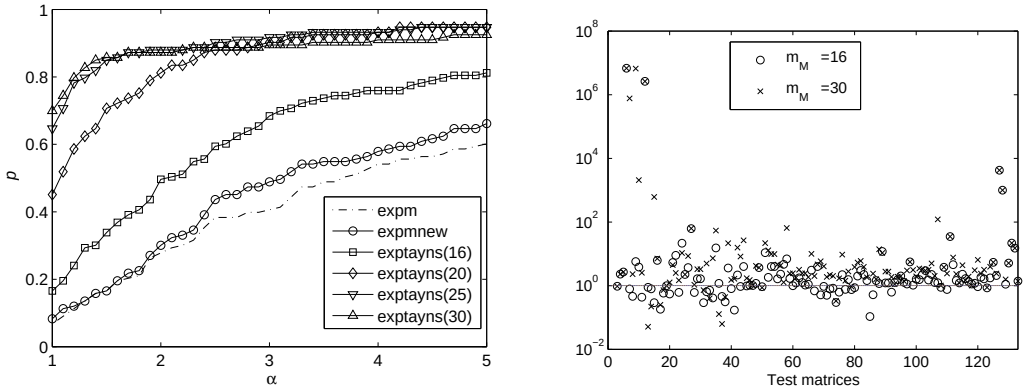
133 matrices from 2×2 to 10×10 from MATLAB (`gallery` test matrices and other special matrices), Eigtool package [Wri02], and references [SIDR11b, Hig05], have been used to compare the proposed algorithm `exptayns` to MATLAB functions `expm` [Hig05], and `expm_new`

Table 2.11: Cost in terms of total number of matrix product evaluations (P) and relative error comparison between `exptayns`, `expm` and `expm_new`.

Maximum allowed Taylor order m_M	16	20	25	30
$E_{exptayns} < E_{expm}$ %	74.44	90.98	89.47	88.72
$(P_{exptayns} - P_{expm})/P_{expm}$ %	-15.47	-15.69	-14.95	-14.35
$E_{exptayns} < E_{expm\_new}$ %	66.17	87.22	87.22	86.47
$(P_{exptayns} - P_{expm\_new})/P_{expm\_new}$ %	1.31	1.04	1.94	2.65

from [AMH09]. The accuracy was tested by computing relative errors $E = \|e^A - \tilde{X}\|_1/\|e^A\|_1$, where $\tilde{X}$ is the computed approximation. The "exact" value of matrix exponential e^A was computed using MATLAB's Symbolic Math Toolbox and a [33/33] diagonal Padé method with scaling and squaring at 1000 decimal digit precision. We have compared function `exptayns` truncating the series in (2.52) and (2.53) by $N = 150$ terms, and truncating them with $N = q+2$ terms. The same results were obtained in almost the 100 % of the test matrices and we used definitely $N = q + 2$ series terms in the comparison with `expm` and `expm_new`. Table 2.11 shows that the cost for `exptayns` is lower than the cost for `expm`, and slightly greater than that for `expm_new`, and that `exptayns` is more accurate than both methods in the majority of test matrices. Figure 2.3a shows the performance profile [DM02] of the compared functions, where the α coordinate varies between 1 and 5 in steps equal to 0.1, and p coordinate is the probability that the considered method has a relative error lower than or equal to α-times the smallest error over all the methods, where probabilities are defined over all matrices. Figure 2.3b shows the ratio of relative errors $E_{expm\_new}/E_{exptayns}$ using the Taylor maximum orders $m_M = 16$ and 30. Figures 2.3a and 2.3b show that `exptayns` has better accuracy than the other functions in the majority of test matrices. A normwise relative error study [Hig08, p. 252-253] was also made and showed that the three functions performed in a numerically stable way on this test.

To sum up, a new scaling and squaring competitive algorithm has been proposed. It is based on a mixed backward and forward error analysis which uses improved bounds for normal and nonnormal matrices. Applied to the Taylor method, it has shown to be more accurate than existing state-of-the-art algorithms in the majority of matrices in numerical tests, with lower or similar cost. Its extension to IEEE single precision arithmetic is straightforward. Now, we are applying the new scaling and squaring algorithm to the Padé method, however, denominator condition problems are expected as in [AMH09].



(a) (b)

Figure 2.3: (a) Performance profile ($m_M = 16, 20, 25, 30$). (b) Ratio of relative errors $\mathrm{E}_{\texttt{expm\_new}}/\mathrm{E}_{\texttt{exptayns}}$ with Taylor maximum orders $m_M = 16$ and 30.

2.4. New scaling-squaring Taylor algorithms for computing the matrix exponential

Referencia del artículo:
J. Sastre, J. Ibáñez, E. Defez, and P. Ruiz
New scaling-squaring Taylor algorithms for computing the matrix exponential
SIAM Journal on Scientific Computing, Volume 37, Issue 1, February 2015, Pages A439–A455, ISSN 1064-8275, http://dx.doi.org/10.1137/090763202

Abstract

The matrix exponential plays a fundamental role in linear differential equations arising in engineering, mechanics, and control theory. The most widely used, and the most generally efficient, technique for calculating the matrix exponential is a combination of "scaling and squaring"with a Padé approximation. For alternative scaling and squaring methods based on Taylor series, we present two modifications that provably reduce the number of matrix multiplications needed to satisfy the required accuracy bounds, and a detailed comparison of the several algorithmic variants is provided.

2.4.1. Introduction

Many engineering and physics phenomena are governed by systems of linear first-order ordinary differential equations with constant coefficients, whose solution is given in terms of the matrix exponential $\exp(A)$, $A \in \mathbb{C}^{n\times n}$. Thus, the matrix exponential plays an important role in many areas of science and technology: control theory, electrodynamic theory of stratified media, the theory of multimode electric power lines, etc. [HLS98, CM02, KT05, IHAR09]. Numerous methods have been proposed for matrix exponential computation [ML03, Hig08]. Of all the methods, Padé approximation in combination with the scaling and squaring technique is the most popular general method [Hig05]. This paper presents two modifications of a Taylor-based scaling and squaring algorithm that are designed to reduce computational costs while preserving accuracy. A previous unpublished and extended version of this work can be found in [SIDR09].

Throughout this paper $\mathbb{R}^{n\times n}$ and $\mathbb{C}^{n\times n}$ denote the sets of real and complex matrices of size $n \times n$, respectively, and I denotes the identity matrix for both sets. $\mathbb{N}$ denotes the set of positive integers and the matrix norm $\|\cdot\|$ denotes any subordinate matrix norm, and in particular $\|\cdot\|_1$, the 1-norm. $\lceil x \rceil$ denotes the lowest integer not less than x and $\lfloor x \rfloor$ denotes the highest integer not exceeding x. This paper is organized as follows. Section 2.4.2 presents a state of the art on matrix exponential computation. Section 2.4.3 deals with the proposed improvements. Numerical experiments are presented in Section 2.4.4. Finally, the conclusions are given in Section 2.4.5.

2.4.2. State of the art

Given a complex matrix A, $\exp(A)$ can be well approximated by either a Padé or Taylor series approximation if $\|A\|$ is small enough. This suggests exploiting the relation

$$\exp(A) = (\exp(2^{-s}A))^{2^s}, \tag{2.59}$$

where s is a nonnegative integer scaling parameter. Based on (2.59), the idea of a scaling and squaring method is to choose s so that $\|2^{-s}A\|$ is sufficiently small, to use Padé or Taylor series to approximate $\exp(2^{-s}A)$, and finally to square the result s times, in the so-called squaring phase. It is well known that the squaring phase can be badly affected by rounding errors (see [Hig08, p. 248]) and it is therefore desirable to keep s as small as possible.

2.4.2.1. Taylor and Padé series

The Taylor series approximation to $\exp(A)$, discussed in [Wes90, GL96, ML03][PCK91, p. 210] is

$$T_m(A) = \sum_{i=0}^{m} \frac{A^i}{i!}, \tag{2.60}$$

where m is the degree of the matrix polynomial $T_m(A)$, and the total number of terms in the series is $m+1$. In general, a larger value of m will improve accuracy, i.e. reduce the absolute error. From now on, we refer to m as the order of the approximation. A naive use of Taylor series is well known to produce serious cancellation error [GL96, p. 567][ML03, p. 10]. Taylor series approximations, the topic of this paper, are discussed in more detail in Section 2.4.2.2.

Padé approximants [War77, GL96, ML03, Hig05, Hig08] are basic tools for computing the matrix exponential. The $[k/l]$ Padé approximant $r_{kl}(A)$ of the matrix exponential is defined by

$$r_{kl}(A) = p_{kl}(A)\,(q_{kl}(A))^{-1},$$

where

$$p_{kl}(A) = \sum_{j=0}^{k} \frac{(k+l-j)!k!}{(k+l)!(k-j)!j!} A^j, \quad q_{kl}(A) = \sum_{j=0}^{k} \frac{(-1)^j (k+l-j)!l!}{(k+l)!(l-j)!j!} A^j.$$

Diagonal approximants ($k = l$) are preferred because r_{mm} is more accurate than r_{kl} with $k \neq l$ but can be evaluated at the same cost. A Padé-based scaling and squaring algorithm is given in [Hig05].

For reasons discussed in [Hig05] and [HAM10], the most popular techniques for computing the matrix exponential are based on scaling and squaring using a Padé approximation, which, in general, produces acceptable accuracy with less work than a Taylor-series method.

2.4.2.2. Details about Taylor series approximants

Algorithm **GSQT** summarizes the general structure of a Taylor-based scaling and squaring algorithm for computing the matrix exponential, where preprocessing and postprocessing (see [War77]) have been omitted.

Algorithm GSQT (general scaling and squaring Taylor algorithm)
% **Input:** An $n \times n$ matrix A, preprocessed if appropriate;
% K, the maximum allowed number of matrix products; and
% $\{m_k\}, k = 1 : K$, the orders of the associated polynomials, from Table 2.13.
Step 1. Execute Algorithm `Order-scale`, which selects the order and scaling parameter of the Taylor polynomial.
Step 2. Execute Algorithm `Taylor-eval` to evaluate the Taylor polynomial in the scaled matrix.
Step 3. Execute the appropriate number of squaring steps of the Taylor polynomial.

The proposed modifications affect Algorithms `Order-scale` (see Section 2.4.3.1) and `Taylor-eval` (Section 2.4.3.2).

2.4.2.3. Relations between the order and the computational effort

The principles that underlie implementation of the `Order-scale` choice in Algorithm `GSQT` are based on [PS73, GL96, Hig05], and [Hig08, pp. 72–74]. Given a matrix A, a scaling parameter s, and relative machine precision u (typically, IEEE double precision), [Hig05] shows how to define a sequence $\{\theta_m\}$ such that an (m, m) diagonal Padé approximation to $\exp(A)$ produces an acceptable backward error bound if $\|2^{-s}A\| \leq \theta_m$. Applying an analogous procedure to Taylor series, [SIDR09] calculated a sequence $\{\Theta_m\}$ such that, if

$$\|2^{-s}A\| \leq \Theta_m, \tag{2.61}$$

using an order-m Taylor series approximation gives an acceptable relative backward error bound using IEEE double precision. Table 2.12 displays values of $\{\theta_m\}$ and $\{\Theta_m\}$.

Table 2.12: Comparison of Padé and Taylor approximations.

Padé			Taylor		
m	θ_m	π_m	m	Θ_m	Π_m
3	1.5e-2	2	3	1.39e-5	2
4	8.5e-2	3	4	3.40e-4	2
5	2.5e-1	3	5	2.40e-3	3
6	5.4e-1	4	6	9.07e-3	3
7	9.5e-1	4	7	2.38e-2	4
8	1.5e0	5	8	5.00e-2	4
9	2.1e0	5	9	8.96e-2	4
10	2.8e0	6	10	1.44e-1	5
11	3.6e0	6	11	2.14e-1	5
12	4.5e0	6	12	3.00e-1	5

The work associated with approximating the matrix exponential using Padé or Taylor series is measured by convention as the number of matrix products. A matrix polynomial can be evaluated in a straightforward way using Horner's nested multiplication method (see, e.g., [GL96, p. 574]). [Hig05] and [Hig08] show how Horner's technique can be combined with

a method due to Paterson and Stockmeyer [PS73] to produce the smallest number of matrix products, denoted by π_m, needed to calculate the (m,m) diagonal Padé approximant.

As described in [SIDR09], an analogous procedure can be applied to a degree-m Taylor approximation of $\exp(A)$. In the Paterson-Stockmeyer method, a positive integer $q < m$ is chosen and the polynomial (2.60) is written as a degree-r polynomial in A^q with $r = \lfloor m/q \rfloor$:

$$T_m(A) = \sum_{k=0}^{r} B_k \left(A^q\right)^k, \quad r = \lfloor m/q \rfloor, \tag{2.62}$$

where the Paterson-Stockmeyer coefficients $\{B_k\}$, $k = 0, \ldots, r$, are themselves matrix polynomials:

$$B_k = \sum_{j=0}^{q-1} b_{qk+j} A^j, \ k = 0, \ldots, r-1, \text{ and } B_r = b_{qr} I + \cdots + b_m A^{m-qr}. \tag{2.63}$$

Note that, with this form, each B_k, $k = 0, \ldots, r-1$, contains q terms, whereas B_r contains $m - qr + 1$ terms. The Paterson-Stockmeyer form is recursively evaluated as follows with the Horner technique, where F_0 gives the final result.

$F_r = B_r$;
for $j = r-1 : -1 : 0, \quad F_j = B_j + A^q \times F_{j+1}$; **end for**

In general, forming a degree-m Taylor polynomial in this way with the Horner-Paterson-Stockmeyer technique requires $q + r - 1$ matrix products, with $q - 1$ of these used to form $A^2, \ldots, A^q$, plus $r = \lfloor m/q \rfloor$ matrix products to compute the sequence $\{F_j\}$. However, in the special case when q divides m (i.e., when $m = qr$), then only $r - 1$ matrix products are needed to apply Horner's technique because the Paterson-Stockmeyer coefficient $B_r = b_m I$ is a multiple of the identity; hence forming F_{r-1} does not involve a matrix product.

Given $q < m$, the number of matrix products needed to compute T_m using the Horner-Paterson-Stockmeyer procedure is thus given by $q + \lfloor m/q \rfloor - 1 - \psi(q,m)$, where $\psi(q,m) = 1$ if q divides m and 0 otherwise. The value of q for which T_m can be computed with the smallest number of matrix products is obtained by approximately minimizing $q + m/q$, giving $q = \lfloor \sqrt{m} \rfloor$. We use Π_m to denote the associated "optimal" number of matrix multiplications:

$$\Pi_m = \lfloor \sqrt{m} \rfloor + \lfloor m/\sqrt{m} \rfloor.$$

Calculation of the Taylor-based sequences $\{\Theta_m\}$ (2.61) and $\{\Pi_m\}$ is summarized in [SIDR09] and in an appendix of [HAM10], with $\tilde{\theta}_m$ denoting Θ_m and $\tilde{\pi}_m$ denoting Π_m. For selected values of m, Table 2.12 displays θ_m and π_m for Padé approximants, and Θ_m and Π_m for Taylor approximants.

Comparing π_m and Π_m, keeping in mind that $\theta_m > \Theta_m$ for each m, we see that, as noted in [SIDR09] and the Appendix of [HAM10], a Padé approximation requires fewer matrix products except when $\|A\|$ lies in the following three intervals:

$$\begin{aligned} \|A\| &\leq 9.07e-3 \ (\Theta_6), \\ 1.50e-2 \ (\theta_3) \leq \|A\| &\leq 8.96e-2 \ (\Theta_9), \quad \text{and} \\ 2.54e-1 \ (\theta_5) \leq \|A\| &\leq 3.00e-1 \ (\Theta_{12}). \end{aligned} \tag{2.64}$$

Table 2.13: k, m_k, q_k, r_k, Θ_{m_k}, and Π_{m_k}

k	m_k	q_k	r_k	Θ_{m_k}	Π_{m_k}
1	2	1	2	2.58e-8	1
2	4	2	2	3.40e-4	2
3	6	2	3	9.07e-3	3
4	9	3	3	8.96e-2	4
5	12	3	4	3.00e-1	5
6	16	4	4	7.80e-1	6
7	20	4	5	1.44e0	7
8	25	5	5	2.43e0	8
9	30	5	6	3.54e0	9
10	36	6	6	4.97e0	10
11	42	6	7	6.48e0	11

2.4.2.4. Maximizing the order

Combining the analysis of [Hig05] and the results of Table 2.12 when k is a specified number of matrix products, [SIDR09] shows how to define m_k, the highest-order Taylor approximation that can be obtained with k matrix products when $\|A\| \le \Theta_{m_k}$. For $k = 1:11$, Table 2.13 shows m_k, q_k, and r_k (the associated Paterson-Stockmeyer indices), Π_{m_k}, and Θ_{m_k}.

Assuming that $\|A\| \le \Theta_{m_k}$, Table 2.13 shows that for even k, the largest Taylor polynomial order achievable with k matrix products is $m_k = (k+2)^2/4$, with $q_k = r_k = (k+2)/2$ and $m_{k+1} = m_k + q_k$, so that allowing one additional matrix product increases the order of the Taylor polynomial by q_k. For an odd number k of matrix products, where $q_k = (k+1)/2$ and $r_k = q_k + 1$, the order of the Taylor polynomial increases by $q_k + 1$ if $k+1$ matrix products are allowed.

In all cases of interest to us, $m_k = q_k r_k$. This means that the Paterson-Stockmeyer form of T_m (2.62) can be expressed as an equivalent polynomial of degree $r-1$ in A^q, giving the following expression in which the "extra"term $b_0 I$ is added separately:

$$T_m(A) = b_0 I + \sum_{k=0}^{r-1} \bar{B}_k (A^q)^k \quad \text{with} \quad \bar{B}_k = \sum_{j=1}^{q} b_{qk+j} A^j, \ k = 0, \ldots, r-1. \tag{2.65}$$

Note that, as distinct from (2.62), the Paterson-Stockmeyer coefficients are barred, and that every $\bar{B}_k$, $k = 0, \ldots, r-1$, contains q terms.

The following three pseudocode fragments show explicitly how the Taylor polynomial $T_m(A)$ is evaluated using (2.65) when $m = qr$. The reader should note in particular the ranges of the indices k and j.

Algorithm Taylor-eval
% Horner-Paterson-Stockmeyer evaluation of the order-m Taylor polynomial
% of $\exp(A)$ with $m = qr$.

% **Input parameters:** $m;\ q;\ r;\ \{A_j = A^j\}, j = 1:q;\ \{b_i\}, i = 0:m.$

Execute **Algorithm PS-coeff**; % calculate $\{\bar{B}_j\}, j = 0:r-1;$
Execute **Algorithm HPS-eval**; % calculate $T_m(A)$;
end % Taylor-eval

Algorithm PS-coeff
% calculation of the r Paterson-Stockmeyer coefficients $\{\bar{B}_k\}$ in (2.65), where $b_i = 1/i!$.
% **Input:** $q;\ r;\ \{A_j = A^j\}, j = 1:q;\ \{b_i\}, i = 0:qr.$
for $k = 0:r-1$
  $\bar{B}_k = 0;$
  for $j = 1:q$
    $\bar{B}_k = \bar{B}_k + b_{qk+j}A_j;$
  end for j
end for k
end % PS-coeff

Algorithm HPS-eval
% Horner-Paterson-Stockmeyer evaluation of the order-m Taylor polynomial
% of $\exp(A)$ with $m = qr$, from (2.65).
% **Input:** $r;\ b_0;\ A_q = A^q;\ \{\bar{B}_k\}, k = 0:r-1.$
$F = \bar{B}_{r-1};$
for $j = r-1:-1:1$
  $F = \bar{B}_{j-1} + A_q \times F;$
end for j
$F = F + b_0 I;$
end % HPS-eval

2.4.2.5. Estimating the total cost

Based on [Hig05], the following approach can be used to estimate the cost of calculating the matrix exponential as a function of the order m. For Taylor series, if $\|A\| > \Theta_m$ for some order m, the scaling parameter s can be chosen as the smallest value of s such that $\|A/2^s\| \leq \Theta_m$, namely, $s = \lceil \log_2(\|A\|/\Theta_m) \rceil$. After calculating the order-m Taylor approximant of $\exp(A/2^s)$, the result will be squared s times, so that the total number of matrix multiplications required to calculate the order-m Taylor approximation is $\Pi_m + \lceil \log_2(\|A\|/\Theta_m) \rceil$. An approximation of this value that depends only on m, denoted by Γ_m, is obtained by omitting the ceiling operation and the constant $\|A\|$:

$$\Gamma_m = \Pi_m - \log_2(\Theta_m). \tag{2.66}$$

When estimating the total for Padé approximations, [Hig05] rules out $m = 1$ and $m = 2$ because of the unfavorable numerical consequences resulting from the need for a large s to

make $\|A/2^s\|$ small enough. The same argument applies to very low-order Taylor polynomials, and we therefore show the values of Γ_{m_k} (2.66) in Table 2.14 only for $k \geq 2$.

Table 2.14: Overall cost (2.66) as a function of Taylor order m.

k	2	3	4	5	6	7	8	9
m_k	4	6	9	12	16	20	25	30
Γ_{m_k}	13.52	9.79	7.48	6.74	6.36	6.48	6.72	7.18

Table 2.14 shows that, as measured by Γ, six matrix products, corresponding to order $m_6 = 16$, produce the minimum computational cost for a Taylor-based approximation.

The usual practice in Taylor-based methods is to decide in advance on the maximum allowed number of matrix products, denoted by K. Based on this practice, Algorithm `Order-scale-1` summarizes the procedures for choosing the order $\hat{m}$ and the scaling parameter $\hat{s}$ in a standard Taylor-based method when K is given.

Algorithm order-scale-1
% Standard technique for choosing the order $\hat{m}$ and the scaling parameter $\hat{s}$
% to compute a Taylor approximation to $\exp(A)$.
% **Input:** K, the maximum number of matrix products allowed to evaluate the polynomial; $\{m_k\}$ and Θ_{m_k} from Table 2.13.
% **Output:** $\hat{k}$, $\hat{m}$, and $\hat{s}$ (the values needed).
$\hat{m} = m_K$; $\hat{k} = K$;
if $\|A\| > \Theta_{m_K}$ **then**
  $\hat{s} = \lceil (\log_2(\|A\|/\Theta_{m_K}) \rceil$
else
  $\hat{s} = 0$;
  for $k = 1 : K$
    if $\|A\| \leq \Theta_{m_k}$ **then** $\hat{m} = m_k$; $\hat{k} = k$; **break; end if**
  end for k
end if
end % Order-scale-1

In the remainder of this paper, we consider new strategies designed to reduce the number of matrix multiplications while preserving accuracy.

2.4.3. Proposed modifications

We propose two modifications to the standard Taylor-based method presented in Section 2.4.2.2. The first modification, described in Section 2.4.3.1, replaces Algorithm **Order-scale-1** with a method that changes the order and the scaling parameter if another combination of these values reduces the number of matrix products. The second modification, discussed in Section 2.4.3.2, neglects higher-order terms in the Taylor series based on relative error bounds.

2.4.3.1. Modification of the choices for order and scaling

Two modifications are proposed of Algorithm **Order-scale-1** for choosing the order and scaling parameter. The motivation for the first change is based on the logic of Algorithm **Order-scale-1**. If $\|A\| \leq \Theta_{m_K}$, $\hat{k}$ is taken as the largest value such that $\|A\| \leq \Theta_{m_{\hat{k}}}$ with $\hat{m} = m_{\hat{k}}$ and $\hat{s} = 0$; evaluation of the Taylor polynomial then requires $\hat{k}$ matrix products. When $\|A\| > \Theta_{m_K}$, then $\hat{k} = K$, $\hat{m} = m_K$, and

$$\hat{s} = \lceil \log_2(\frac{\|A\|}{\Theta_{m_K}}) \rceil. \tag{2.67}$$

We know in this case that $\hat{s} \geq 1$ and, by definition of $\hat{s}$, that

$$\frac{1}{2}\Theta_{m_K} < \frac{\|A\|}{2^{\hat{s}}} \leq \Theta_{m_K}. \tag{2.68}$$

With these latter choices, a Taylor approximation of order m_K to $\exp(A)$ can be computed with $K + \hat{s}$ matrix products.

But it is possible in some cases to reduce this number. If K, $\hat{s}$, and $\|A\|$ are such that $\|A\|/2^{\hat{s}} \leq \Theta_{m_{K-1}}$, then we can take $k^* = K - 1$, so that the order of the Taylor polynomial is m_{K-1}, keeping $s^* = \hat{s}$. Thus the number of matrix products is $K - 1 + \hat{s}$, one smaller than with the original choices. It follows from (2.68) that this situation is possible only when $\Theta_{m_{K-1}} > \frac{1}{2}\Theta_{m_K}$, which is true only for $K \geq 7$ (see Table 2.13). The strategy just described corresponds to executing Algorithm **Order-scale-2**, shown below, with option = 1.

If one is prepared to accept additional squarings, the idea can be generalized in a limited way for certain values of $\|A\|$ when $K = 8$ and $K = 9$, as implemented in Algorithm **Order-scale-2**. The idea is to find $k^* < K - 1$ and $s^* > \hat{s}$ such that $\|A\|/2^{s^*} < \Theta_{m_{k^*}}$ and $k^* + s^* < K - 1 + \hat{s}$, meaning that, using order m_{k^*} and scaling parameter s^*, the total number of matrix products is smaller than with option = 1. Allowing such changes corresponds to option = 2 in Algorithm **Order-scale-2**.

The pseudocode for Algorithm **Order-scale-2** assumes that $K \leq 9$, since it is straightforward to show (see [SIDR09]) that performing the analogue of option 2 with $K > 9$ does not reduce the number of matrix products.

Algorithm order-scale-2
% Two alternative ways to choose the order m^* of a Taylor approximation
% to $\exp(A)$ and the scaling parameter s^*;
% **Input: option**, set to 0, 1, or 2; K, the maximum number of matrix products
% allowed in evaluating the polynomial; $\{m_k\}$ and $\{\Theta_{m_k}\}$ from Table 2.13.
% **Output:** k^*, m^*, and s^*.
$m^* = m_K$; $k^* = K$;
if $\|A\| \leq \Theta_{m_K}$ **then**
  $s^* = 0$; $\hat{s} = 0$;
  for $k = 1 : K$
    if $\|A\| \leq \Theta_{m_k}$ **then** $m^* = m_k$; $k^* = k$; **break; end if**

Table 2.15: Matrix products needed when using options 0, 1, and 2 in Algorithm **Order-scale-2** for $K \geq 7$ to approximate $\exp(A)$ when $\|A\|/2^{\hat{s}}$ lies in the intervals shown, where $\hat{s}$ is defined by (2.67).

K	option	Interval of $\|A\|/2^{\hat{s}}$	k^*	m^*	s^*	Matrix products
7	0	$(\frac{1}{2}\Theta_{20}, \Theta_{20}] = (0.72, 1.44]$	7	20	$\hat{s}$	$7+\hat{s}$
7	1	$(\frac{1}{2}\Theta_{20}, \Theta_{16}] = (0.72, 0.781]$	6	16	$\hat{s}$	$6+\hat{s}$
7	1	$(\Theta_{16}, \Theta_{20}] = (0.781, 1.44]$	7	20	$\hat{s}$	$7+\hat{s}$
8	0	$(\frac{1}{2}\Theta_{25}, \Theta_{25}] = (1.215, 2.43]$	8	25	$\hat{s}$	$8+\hat{s}$
8	1	$(\frac{1}{2}\Theta_{25}, \Theta_{20}] = (1.215, 1.44]$	7	20	$\hat{s}$	$7+\hat{s}$
8	1	$(\Theta_{20}, \Theta_{25}] = (1.44, 2.43]$	8	25	$\hat{s}$	$8+\hat{s}$
8	2	$(\frac{1}{2}\Theta_{25}, \Theta_{20}] = (1.215, 1.44]$	7	20	$\hat{s}$	$7+\hat{s}$
8	2	$(\Theta_{20}, 2\Theta_{16}] = (1.44, 1.562]$	6	16	$\hat{s}+1$	$7+\hat{s}$
8	2	$(2\Theta_{16}, \Theta_{25}] = (1.562, 2.43]$	8	25	$\hat{s}$	$8+\hat{s}$
9	0	$(\frac{1}{2}\Theta_{30}, \Theta_{30}] = (1.77, 3.54]$	9	30	$\hat{s}$	$9+\hat{s}$
9	1	$(\frac{1}{2}\Theta_{30}, \Theta_{25}] = (1.77, 2.43]$	8	25	$\hat{s}$	$8+\hat{s}$
9	1	$(\Theta_{25}, \Theta_{30}] = (2.43, 3.54]$	9	30	$\hat{s}$	$9+\hat{s}$
9	2	$(\frac{1}{2}\Theta_{30}, \Theta_{25}] = (1.77, 2.43]$	8	25	$\hat{s}$	$8+\hat{s}$
9	2	$(\Theta_{25}, 2\Theta_{20}] = (2.43, 2.88]$	7	20	$\hat{s}+1$	$8+\hat{s}$
9	2	$(2\Theta_{20}, 4\Theta_{16}] = (2.88, 3.124]$	6	16	$\hat{s}+2$	$8+\hat{s}$
9	2	$(4\Theta_{16}, \Theta_{30}] = (3.124, 3.54]$	9	30	$\hat{s}$	$9+\hat{s}$

```
    end for
    if option = 2 then
        if k ≥ 8 and ‖A‖ ≤ 2Θ_{m_{k-2}} then k* = k − 2; m* = m_{k*}; s* = 1;
        else if k = 9 and ‖A‖ ≤ 4Θ_{m_{k-3}} then k* = k − 3; m* = m_{k*}; s* = 2;
        end if
    end if      % end logic for option = 2
else      % here, it must be true that ‖A‖ > Θ_{m_K}
    ŝ = ⌈log2(‖A‖/Θ_{m_K})⌉; s* = ŝ;
    if option > 0 then      % possibly change the order and scale
        if K ≥ 7 and ‖A‖/2^ŝ ≤ Θ_{m_{K-1}} then k* = K − 1; m* = m_{k*};
        else if option = 2 then
            if K ≥ 8 and ‖A‖/2^ŝ ≤ 2Θ_{m_{K-2}} then k* = K − 2; m* = m_{k*}; s* = ŝ + 1;
            else if K = 9 and ‖A‖/2^ŝ ≤ 4Θ_{m_{K-3}} then
                k* = K − 3; m* = m_{k*}; s* = ŝ + 2;
            end if
        end if      % end logic for option = 2
    end if      % end logic for option > 0
end if      % end logic for ‖A‖ > Θ_{m_K}
end      % Order-scale-2
```

When $K \geq 7$, assuming that $\hat{s}$ is defined by (2.67), Table 2.15 shows the number of matrix products needed with option 0 (the standard method), option 1, and option 2 to approximate $\exp(A)$ when $\|A\|/2^{\hat{s}}$ lies in the intervals shown.

For certain intervals of $\|A\|$, the total number of matrix products needed to form the Taylor polynomial with option 2 can be lower than with option 1. However, in those cases the number s^* of squarings is larger, possibly leading to more numerical error. Numerical tests that explore this issue are given in Section 2.4.4.

2.4.3.2. Neglecting higher-order terms of the Taylor polynomial

The motivation for the second modification is that, in some circumstances, the highest-order terms in the Taylor polynomial are negligible relative to $\|\exp(A)\|$, where "negligible" is defined in terms of the level of rounding error.

Recall from Algorithm **Taylor-eval** in Section 2.4.2.2 that, given a number of multiplications k, the Taylor approximation of order m_k is defined by applying the recursive Horner-Paterson-Stockmeyer procedure, where there are r_k Paterson-Stockmeyer coefficients, $\bar{B}_0, \bar{B}_1, \ldots, \bar{B}_{r_k-1}$, each a matrix polynomial in A of degree q_k (see Algorithm **PS-coeff**). It follows from the discussion in Section 2.4.2.4 that reducing the number of matrix multiplications from k to $k-1$ implies a reduction by q_k in order of the Taylor polynomial (see Table 2.13).

Let $\hat{k}$ denote a number of matrix products, with $\hat{m}$, $\hat{q}$, and $\hat{r}$ the associated values, so that $\hat{m} = \hat{q}\hat{r}$. From Algorithms **PS-coeff** and **HPS-eval**, the $\hat{q}$ highest-degree terms in $T_{\hat{m}}(A)$ can be expressed as

$$H_{\hat{q},\hat{r}}(A) = \bar{B}_{\hat{r}-1}(A^{\hat{q}})^{\hat{r}-1}. \tag{2.69}$$

These terms need not be formed or included in the Taylor polynomial if

$$\|e^{-A}H_{\hat{q},\hat{r}}(A)\| \le u, \tag{2.70}$$

where u is unit roundoff, since then

$$\|H_{\hat{q},\hat{r}}(A)\| = \|e^{A}e^{-A}H_{\hat{q},\hat{r}}(A)\| \le \|e^{A}\|u. \tag{2.71}$$

In this case, the desired accuracy can be achieved by omitting these $\hat{q}$ terms, thereby using $\hat{k}-1$ matrix products to form a Taylor polynomial of order $m_{\hat{k}-1} = (\hat{r}-1)\hat{q}$, since, using (2.71), it follows that the difference between $T_{m_{\hat{k}-1}}(A)$ and $T_{m_{\hat{k}}}(A)$ satisfies

$$\|T_{m_{\hat{k}-1}}(A) - T_{m_{\hat{k}}}(A)\|/\|e^{A}\| = \|H_{\hat{q},\hat{r}}(A)\|/\|e^{A}\| \le u, \tag{2.72}$$

and then it is not significant for machine precision. A practical test of whether (2.70) holds can be based on obtaining bounds on the separate factors of $\|H_{\hat{q},\hat{r}}(A)\|$:

$$\|H_{\hat{q},\hat{r}}(A)\| \le \|\bar{B}_{\hat{r}-1}\|\|A^{\hat{q}}\|^{\hat{r}-1}, \tag{2.73}$$

where computing the norm of $\bar{B}_{\hat{r}-1}$ does not involve any matrix products beyond those needed to compute the coefficient itself. Thus satisfaction of the following relationship shows that the desired accuracy can be achieved while reducing the number of matrix products by one:

$$\|e^{-A}\|\|\bar{B}_{\hat{r}-1}\|\|A^{\hat{q}}\|^{\hat{r}-1} \le u. \tag{2.74}$$

If $\|A\| \leq \Theta_{m_{\hat{k}}}$, since $\|-A\| = \|A\|$, then $e^{-A} \approx T_{m_{\hat{k}}}(-A)$. Thus, using (2.65) we can obtain the following bound for $\|e^{-A}\|$, denoted by $b_{\exp}$,

$$\|e^{-A}\| \approx \|T_{m_{\hat{k}}}(-A)\| \leq b_{\exp} = \|b_0 I + \hat{B}_0\| + \sum_{l=1}^{r-1} \|\hat{B}_l\| \|A^{\hat{q}}\|^l, \tag{2.75}$$

where $\hat{B}_l = \sum_{j=1}^{q} (-1)^{ql+j} b_{ql+j} A^j$, $l = 0, \ldots, r-1$. Bound (2.75) can be evaluated with no matrix products reusing the matrix powers A^j, $j = 2, 3, \ldots, \hat{q}$, computed for evaluating $T_{m_{\hat{k}}}(A)$ in the following algorithm.

Algorithm bexp-bound
% Calculation of bound $b_{\exp}$ of $\|\exp(-A)\|$ from (2.75), with $m = qr$.
% **Input:** q; r; $\{A_j = A^j\}, j = 1:q$; $\{b_i = (-1)^i/i!\}, i = 0:qr$.
Execute **Algorithm PS-coeff** with $b_i = (-1)^i/i!$ % calculate $\{\hat{B}_l\}, l = 0:r-1$;
$b_{\exp} = \|\hat{B}_{r-1}\|$;
for $l = r-1:-1:2$
  $b_{\exp} = \|\hat{B}_{l-1}\| + \|A_q\| \times b_{\exp}$;
end for l
$b_{\exp} = \|b_0 I + \hat{B}_0\| + \|A_q\| \times b_{\exp}$;
end % bexp-Bound

Note that, since $\|A\| \leq \Theta_{m_{\hat{k}}}$ then

$$b_{\exp} \leq \sum_{i=0}^{m_{\hat{k}}} \|A\|^i/i! = T_{m_{\hat{k}}}(\|A\|) \approx \mathrm{e}^{\|A\|}, \tag{2.76}$$

and $b_{\exp}$ is strictly lower than $\mathrm{e}^{\|A\|}$ for some matrices, e.g. for matrix

$$A = \begin{pmatrix} 1.25 & 1.25 \\ 1.25 & 1.25 \end{pmatrix}, \tag{2.77}$$

it follows that $\|A\|_1 = 2.5$, $\Theta_{25} < \|A\|_1 < \Theta_{30}$, and then, Algorithm **Order-Scale-2** with $K = 9$ and option= 1 gives $m^* = 30$ and $s^* = 0$. Then, from Table 2.13 it follows that $q = 5$ and $r = 6$. Computing symbolically the exact value of e^{-A} and using the 1-norm in (2.75) gives

$$\|\mathrm{e}^{-A}\|_1 = 1 < b_{\exp} = 1.25 < \mathrm{e}^{\|A\|_1} = 12.18, \tag{2.78}$$

and $b_{\exp}\|\bar{B}_5\|_1 \|A^5\|_1^5 = 7.57e-17 < u \approx 1.11e-16$ (IEEE double precision), showing that (2.74) holds and the number of matrix products can be reduced. However, using bound $\|\mathrm{e}^{-A}\| \leq \mathrm{e}^{\|A\|}$ in (2.74) gives $\mathrm{e}^{\|A\|_1}\|\bar{B}_5\|_1 \|A^5\|_1^5 = 7.39e-16 > u$ and condition (2.74) is not guaranteed. Hence, since bound $b_{\exp}$ is not greater than $\mathrm{e}^{\|A\|}$ and can be lower for some matrices, we use it in (2.74) to give the final condition

$$b_{\exp}\|\bar{B}_{\hat{r}-1}\| \|A^{\hat{q}}\|^{\hat{r}-1} \leq u. \tag{2.79}$$

Similar tests can be devised and applied recursively, eliminating sets of $\hat{q}$ terms each time: if $b_{\exp}\|\bar{B}_{\hat{r}-j}\| \|A^{\hat{q}}\|^{\hat{r}-j} \leq u$ with $j = 2$, then the number of matrix products can be reduced to

Table 2.16: k, m_k, Θ_{m_k}, Θ'_{m_k}, and ϑ_{m_k}

k	m_k	Θ_{m_k}	Θ'_{m_k}	ϑ_{m_k}
1	2	2.5810e-8	8.7334e-6	8.7334e-6
2	4	3.3972e-4	1.6778e-3	1.6778e-3
3	6	9.0657e-3	1.7720e-2	1.7720e-2
4	9	8.9578e-2	1.1354e-1	1.1354e-1
5	12	2.9962e-1	3.2690e-1	3.2690e-1
6	16	7.8029e-1	7.8738e-1	7.8738e-1
7	20	1.4383e0	1.4070e0	1.4383e0
8	25	2.4286e0	2.3392e0	2.4286e0
9	30	3.5397e0	3.3908e0	3.5397e0

$\hat{k}-2$ computing a Taylor polynomial of order $\hat{m} = (\hat{r}-2)\hat{q}$, and so on with $j = 3, 4, \ldots, \hat{r}-1$ computing Taylor polynomials of orders $\hat{m} = (\hat{r}-j)\hat{q}$ with $\hat{k}-j$ matrix products.

The next theorem establishes that (2.74) holds for matrices whose norm lies in certain intervals, so the number of matrix products can be reduced for those matrices.

Theorem 3 *Let $m_k = q_k r_k$ where $q_k = r_k = (k+2)/2$ for even $k > 0$, and $q_k = (k+1)/2$ and $r_k = q_k + 1$ for odd $k > 0$. Let $A \in \mathbb{C}^{n\times n}$, let $H_{q_k,r_k}(A)$ be the q_k highest-degree terms in $T_{m_k}(A) = \sum_{i=0}^{m_k} A^i/i!$, let u be the relative machine precision, and let $\Theta'_{m_{k-1}}$ be the values such that*

$$\Theta'_{m_{k-1}} = \max\{\theta, e^{\theta} H_{q_k,r_k}(\theta) \le u\}. \tag{2.80}$$

Then, if $\|A\| \le \Theta'_{m_{k-1}}$ then condition (2.74) holds and $\|T_{m_k}(A) - T_{m_{k-1}}(A)\|/\|e^A\| \le u$.

Proof. Using (2.74), (2.69), (2.65), and (2.80), since $\|\mathrm{e}^{-A}\| \le \mathrm{e}^{\|A\|}$, if $\|A\| \le \Theta'_{m_{k-1}}$ it follows that

$$\|\mathrm{e}^{-A}\|\|\bar{B}_{r_k-1}\|\|A^{q_k}\|^{r_k-1} \le \mathrm{e}^{\|A\|} H_{q_k,r_k}(\|A\|) \le \mathrm{e}^{\Theta'_{m_{k-1}}} H_{q_k,r_k}(\Theta'_{m_{k-1}}) \le u, \tag{2.81}$$

and condition (2.74) holds. Hence, from (2.69)-(2.71) it follows that (2.72) holds. □

Using a zero finder we computed the values Θ'_{m_k}, $k = 1, \ldots, 9$, presented in Table 2.16. Note that $\Theta'_{m_k} > \Theta_{m_k}$ for $k = 1, \ldots, 6$. Thus, if k and $\|A\|$ are such that $\Theta_{m_{k-1}} < \|A\| \le \Theta'_{m_{k-1}}$, then, Theorem 3 states that we can omit the last q terms of $T_{m_k}(A)$ and compute Taylor polynomial $T_{m_{k-1}}(A)$ instead, reducing the number of matrix products by one. Hence, taking $\vartheta_{m_k} = \max\{\Theta_{m_k}, \Theta'_{m_k}\}$, see Table 2.16, and substituting Θ_{m_k} with ϑ_{m_k} in Algorithm **Order-scale-2**, the number of matrix products is reduced with respect to the original Algorithm **Order-scale-2** for matrices with $\Theta_m < \|A\| \le \Theta'_m$, $m = 2, 4, 6, 9, 12, 16$, and $\Theta_{16} < \|A\|/2^{s^*} \le \Theta'_{16}$, where s^* is the scaling parameter obtained by **Order-scale-2** with the new ϑ_{m_k} values.

Taking into account the new values ϑ_{m_k} from Table 2.16 and proceeding in a way similar to [SIDR09], the intervals of $\|A\|$, where a Taylor approximation requires fewer matrix products than Padé approximation, increase with respect to those given in (2.64): $\|A\| \le 0.11$ (ϑ_9) and 0.25 $(\theta_5) < \|A\| \le 0.33$ (ϑ_{12}).

Note that $\Theta_{m_k} > \Theta'_{m_k}$ for $m_k = 20, 25, 30$. However, using (2.76) it follows that condition (2.79) is less restrictive than (2.81), and the example was given above of the matrix from (2.77) where condition (2.79) holds with $m = 30$ for $\|A\|_1 = 2.5 > \Theta_{25} > \Theta'_{25}$.

Algorithm **HPS-eval-2** is analogous to **HPS-eval**, but implements the performance of bound tests based on (2.79) to reduce the number of matrix products.

Algorithm HPS-eval-2
% Horner-Paterson-Stockmeyer evaluation of the order-m Taylor polynomial
% of $\exp(A)$ with $m = qr$, performing bound tests to check if the higher-order
% terms of the Taylor series can be neglected.
% **Input:** $s;\ q;\ r;\ \{b_i\}, i = 0 : qr;\ \{A_j = A^j\}, j = 1 : q;\ \{\bar{B}_k\}, k = 0 : r-1;\ u.$
% **Output:** $F = T_m(A)$.
Execute **Algorithm bexp-bound**; % Compute bound b_{exp}
$F = \bar{B}_{r-1}$;
for $j = r-1 : -1 : 1$
  if $(b_{exp}\|F\|_1\|A_q\|_1^j \le u)$ **then**
    $F = \bar{B}_{j-1}$; % Reduction of one matrix product
  else
    $F = \bar{B}_{j-1} + A_q \times F$;
  end if
end for j
$F = F + b_0 I$;
end % HPS-eval-2

Taking into account the two proposed modifications, in the next section we test the following versions of the Taylor Algorithm GSQT presented in Section 2.4.2.2:

- TSTD (Taylor standard) uses Algorithm Order-scale-2 with option 0 and Taylor-eval. Cost $k^* + s^*$ matrix products (see Order-scale-2).
- TPS (Taylor Paterson-Stockmeyer) uses Algorithm Order-scale-2 with option 1 and Taylor-eval. Cost $k^* + s^*$ matrix products.
- OTPS (Optimal TPS) uses Algorithm Order-scale-2 with option 2 and Taylor-eval. Cost $k^* + s^*$ matrix products.
- TPSBT(TPS performing Bound Tests) is analogous to TPS except for the use of the new ϑ_m values in Algorithm Order-scale-2, and the Algorithm HPS-eval-2 instead of HPS-eval in Taylor-eval to perform the bound tests. Cost less than or equal to $k^* + s^*$ matrix products.
- OTPSBT (Optimal TPSBS) is analogous to TPSBT using option 2 in Algorithm Order-scale-2. Cost less than or equal to $k^* + s^*$ matrix products.

Finally, we analyze the rounding error for the proposed Taylor algorithms. The effect of rounding errors in the evaluation of the Taylor matrix polynomial can be bounded similarly to the numerator of Padé approximants [Hig05, p. 1185]. Since $e^{-\|A\|} \le \|e^A\|$, using Theorem

2.2 of [Hig05, p. 1184] with $\|A\|_1 \leq \upsilon_m$, where $\upsilon_m = \Theta_m$ for TSTD, TPS and OTPS, and $\upsilon_m = \vartheta_m$ for TPSBT and OTPSBT, and noting that all the coefficients in the Taylor matrix polynomial are positive, it follows that

$$\begin{aligned} \left\|T_m(A) - \hat{T}_m(A)\right\|_1 &\leq \widetilde{\gamma}_{mn} T_m\left(\|A\|_1\right) \approx \widetilde{\gamma}_{mn} e^{\|A\|_1} \leq \widetilde{\gamma}_{mn} \left\|e^A\right\|_1 e^{2\|A\|_1} \\ &\simeq \widetilde{\gamma}_{mn} \left\|T_m(A)\right\|_1 e^{2\|A\|_1} \leq \widetilde{\gamma}_{mn} \left\|T_m(A)\right\|_1 e^{2\upsilon_m}, \end{aligned} \tag{2.82}$$

where $A \in \mathbb{C}^{n\times n}$, $\hat{T}_m(A)$ is the computed Taylor approximation, and $\widetilde{\gamma}_k = cku/(1-cku)$, where c is a small integer constant [Hig02a]. Hence, the relative error is bounded approximately by $\tilde{\gamma}_{mn} e^{2\upsilon_m}$, which is a satisfactory bound taking into account the values of Θ_m and ϑ_m given in Table 2.16 (see [Hig05, p. 1185]), and there is no rounding error contribution of solving multiple linear systems as in Padé methods.

2.4.4. Numerical experiments

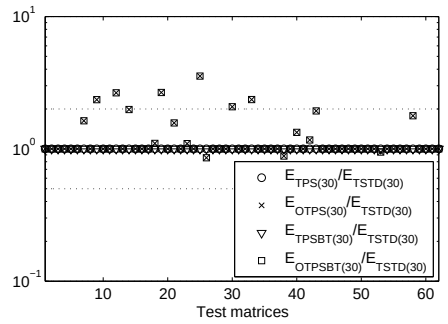
MATLAB implementations of Algorithms TPS, TPSBT, OTPS and OTPSBT with $m_K = 20, 25$, and 30 were compared to the standard Taylor algorithm TSTD, and the MATLAB scaling and squaring Padé function **expm** with maximum orders $m = 13, 17$, and 21; see [Hig05]. Tests were done on an 2.8 GHz Intel Xeon processor with 4 GB main memory and MATLAB 7.7. Accuracy was tested by computing relative error $\mathrm{E} = \|\mathrm{e}^A - \tilde{X}\|_1/\|\mathrm{e}^A\|_1$, where $\tilde{X}$ is the computed solution, and the "exact" value e^A was computed using the Symbolic Math Toolbox of MATLAB and a [33/33] Padé method with scaling and squaring, at 1000-digit decimal arithmetic for Case Study 1 (small matrices), 250-digit decimal arithmetic for Case Study 2 (matrices 50×50), and quadruple precision in Case Study 2 (matrices 500×500). An extra output parameter P was added to all tested functions to return the number of matrix products. 4/3 products were added in function **expm** for each solution of a multiple right-hand side linear system [BD99].

2.4.4.1. Case Study 1

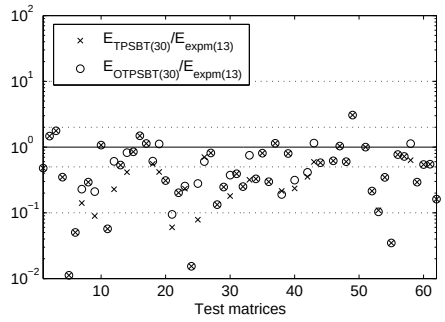
In this test we considered the same test matrices as in [Hig05], except for three matrices where **expm** gave infinite results in MATLAB, i.e. matrices 17 "ipjfact", 42 "invhilb", and 44 "pascal" with size 8×8, from the **matrix** function given in the Matrix Function Toolbox [Hig08, Appendix D], and matrix 43 "magic", which was repeated as matrix 49. Table 2.17 shows the total number of matrix products P computed by each function to evaluate the matrix exponential of the 62 test matrices. It shows that the proposed Taylor algorithms, and especially TPSBT and OTPSBT, need fewer matrix products than the standard Taylor algorithm TSTD, and between 5.08 % and 6.50 % more matrix products than **expm**. The cost of OTPSBT is very similar for $m_K = 20, 25, 30$, as expected; see Algorithm **Order-scale-2**. Thus, from now on we consider only $m_K = 30$ for OTPSBT. Figure 2.4a shows the relative error ratios of all proposed Taylor functions with respect to the standard TSTD for $m_K = 30$. This figure shows that the TPS and TPSBT errors are similar to that of the standard TSTD for all matrices. However, OTPS and OTPSBT errors are greater for some matrices, confirming that the extra squaring in both algorithms can lead to more numerical error; see subsection 2.4.3.1. Figure 2.4a also shows that TPS error and TPSBT error are similar for all matrices. The same happens with OTPS and OTPSBT. This fact is

Table 2.17: Total number of matrix products P computed by each function to evaluate the exponentials of all the test matrices from Case Study 1.

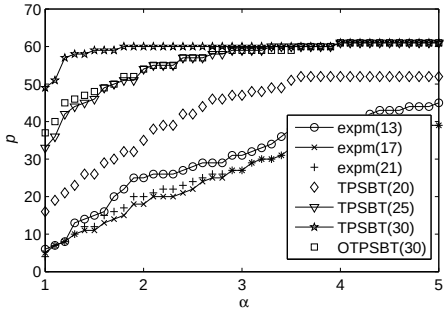
m_K	TSTD	TPS	OTPS	TPSBT	OTPSBT	max. m **expm**	expm
20	724	719	719	668	668	13	635.67
25	739	724	719	673	672	17	637.67
30	757	739	719	677	672	21	649.67



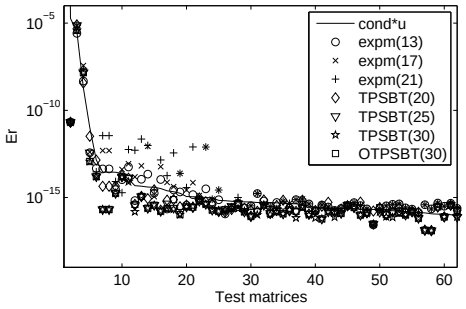
(a) Relative error ratios



(b) Relative error ratios



(c) Performance profile



(d) Normwise relative error

Figure 2.4: Comparison results, Case Study 1.

Table 2.18: Total number of matrix products P computed by each function to evaluate the exponentials of all 50×50 matrices from Case Study 2.

m_K	TSTD	TPS	OTPS	TPSBT	OTPSBT	max. **expm** order	expm
20	469	465	465	430	430	13	408.67
25	479	469	465	432	431	17	412.33
30	487	479	465	429	431	21	416.33

supported by (2.72), which shows that the norm of the neglected terms in functions TPSBT and OTPSBT, relative to the exact value $\exp(A)$, is not significant for machine precision.

Figure 2.4b presents the relative error ratios for the most efficient Taylor functions TPSBT and OTPSBT with $m_K = 30$, and **expm** with optimal maximum order $m = 13$ [Hig05]. Figure 2.4b shows that OTPSBT and TPSBT displayed error comparable to **expm**. Figure 2.4c shows the performances [DM02] of TPSBT and OTPSBT, and **expm** with maximum orders $m = 13, 17, 21$ (see [Hig05]), where the α coordinate varies between 1 and 5 in steps equal to 0.1, and the p coordinate is the probability that the considered algorithm has a relative error lower than or equal to α-times the smallest error over all the methods, where probabilities are defined over all matrices. Figure 2.4c shows that in this case study the most accurate function was TPSBT with $m_K = 30$ followed by OTPSBT with $m_K = 30$ confirming that the extra squaring in OTPSBT can yield a lower accuracy.

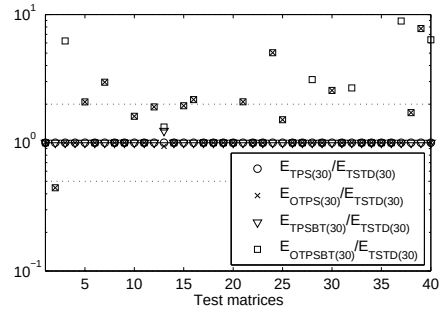
To test numerical stability we plotted the normwise relative errors of the considered functions. Figure 2.4d shows the relative errors of all implementations, and a solid line that represents the unit roundoff multiplied by the relative condition number of the exponential function at X [Hig08, p. 56],

$$\mathrm{cond}_{\exp}(X) = \lim_{\varepsilon \to 0} \sup_{\|E\| \leq \varepsilon} \frac{\left\| e^{X+E} - e^X \right\|}{\varepsilon} \frac{\|X\|}{\|e^X\|}.$$

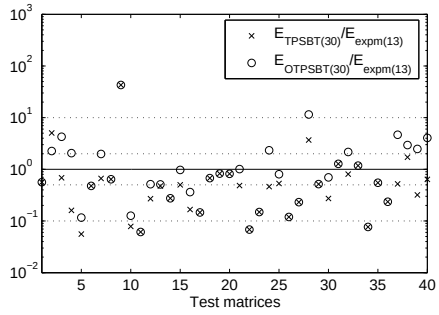
Relative condition number was computed using the MATLAB function **expm_cond** from the Matrix Function Toolbox [Hig08, Appendix D] (`http://www.ma.man.ac.uk/~higham/mftoolbox`). For a method to perform in a backward and forward stable manner, its error should lie not far above this line on the graph [Hig05, p. 1188]. Figure 2.4d shows that all functions performed in a numerically stable way.

2.4.4.2. Case Study 2

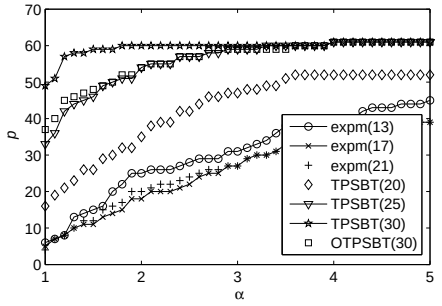
In this case study 39 matrices of dimension $n = 50$ and 36 matrices of dimension $n = 500$ from MATLAB function **matrix** in the Matrix Computation Toolbox were used as the test battery (matrices whose exponential cannot be represented in double precision because overflow errors were excluded from the 52 total possible matrices). Fortran versions of functions TSTD, TPS, OTPS, TPSBT, OTPSBT with $m_K = 30$ and **expm** with maximum order $m = 13$ [Hig05] were implemented, and made available online in [FOR], to measure execution times for the 500×500 matrices. Tables 2.18 and 2.19 present the total number of matrix products P computed to evaluate the exponential of all matrices, and Table 2.19 also presents the mean and standard deviation of the total execution time t in seconds to compute the exponential of all 500×500 matrices with 100 repetitions of the experiment.



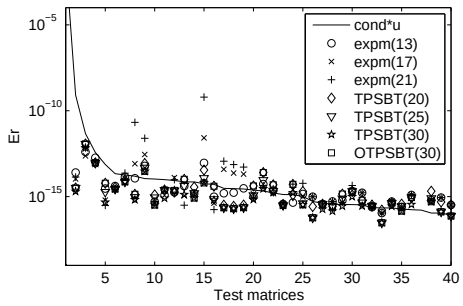
(a) Relative error ratios, $n = 50$



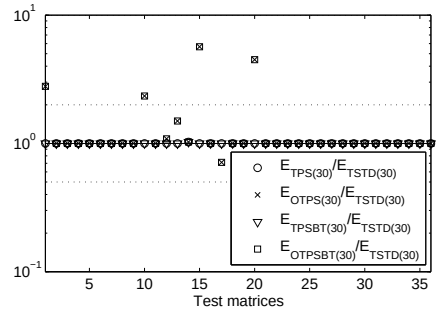
(b) Relative error ratios, $n = 50$



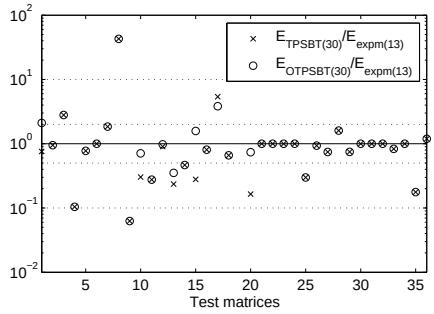
(c) Performance profile, $n = 50$



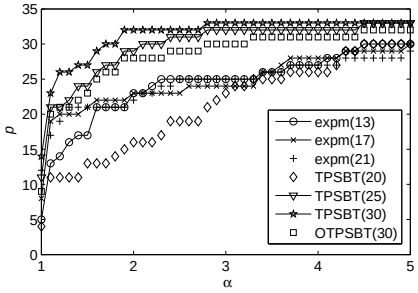
(d) Normwise relative error, $n = 50$



(e) Relative error ratios, $n = 500$



(f) Relative error ratios, $n = 500$



(g) Performance profile, $n = 500$

Figure 2.5: Comparison results, Case Study 2.

Table 2.19: Total number of matrix products P to compute the exponential of all 500×500 matrices from Case Study 2. Mean and standard deviation of total execution time t in seconds for 100 repetitions of the experiment, with $m_K = 30$ in Taylor functions, and the optimal maximum order $m = 13$ in **expm**; see [Hig05].

	`TSTD`	`TPS`	`OTPS`	`TPSBT`	`OTPSBT`	`expm`
P	506	487	479	440	439	425
mean(t) (seconds)	829.97	787.39	782.67	692.28	694.59	762.58
standard deviation(t)	2.51	2.31	2.33	2.11	1.99	2.26

We omitted plotting the normwise relative errors for the 500×500 matrices because they were too large to compute the relative condition number.

Similar conclusions to those from Case Study 1 are obtained from Tables 2.18 and 2.19 and Figure 2.5: TPSBT and OTPSBT with $m_K = 30$ had a lower cost than TSTD, and between 3.29 % and 5.46 % more matrix products than **expm** with maximum order $m = 13$, and TPSBT and OTPSBT execution times for the 500×500 matrices were 9.21 % and 8.91 % lower than **expm**, respectively. TPS and TPSBT errors were similar to TSTD error in all cases. OTPS and OTPSBT errors were greater than TSTD error in some cases, as expected because of the extra squaring. In the majority of tests TPS and OTPS had similar errors to TPSBT and OTPSBT, respectively, as supported by (2.72). The performance profiles show that TPSBT with $m_K = 30$ was the most accurate function in both the 50×50 and the 500×500 matrices.

2.4.5. Conclusions

This work developed four new Taylor algorithms TPS, OTPS, TPSBT, and OTPSBT to compute the matrix exponential based on two modifications to the standard algorithm that reduce computational cost and preserve accuracy. The first modification changes the order and the scaling parameter of the standard scaling algorithm if another combination of these values reduces the number of matrix products. The second modification neglects higher-order terms in the Taylor series based on relative error bounds. Finally, a detailed comparison of the several algorithmic variants was provided.

2.5. Accurate and efficient matrix exponential computation

Referencia del artículo:
J. Sastre, J. Ibáñez, P. Ruiz and E. Defez
Accurate and efficient matrix exponential computation
International Journal of Computer Mathematics, Volume 914, Issue 1, 2014, Pages 97–112, ISSN 0020-7160, http://dx.doi.org/10.1080/00207160.2013.791392

Abstract

This work gives a new formula for the forward relative error of matrix exponential Taylor approximation and proposes new bounds for it depending on the matrix size and the Taylor approximation order, providing a new efficient scaling and squaring Taylor algorithm for the matrix exponential. A Matlab version of the new algorithm is provided and compared with Padé state-of-the-art algorithms obtaining higher accuracy in the majority of tests at similar or even lower cost.

2.5.1. Introduction

Matrix exponential plays a fundamental role in linear systems arising in many areas of science, and a large number of methods for its computation have been proposed [ML03, Hig08]. This work improves the scaling and squaring algorithm presented in [SIDR11a] providing a competitive scaling and squaring algorithm for matrix exponential computation. The new algorithm employs an improved version of Theorem 1 from [SIDR11a] to bound the norm of matrix power series. A new formula for the forward relative error of Taylor approximation in exact arithmetic and new sharp bounds for forward and backward relative errors are given. Moreover, taking into account that the roundoff error in the computation of Taylor matrix polynomial tends to increase with the matrix size and the approximation order, we propose increasing the allowed bounds for the error in exact arithmetic with both parameters. A Matlab version of the new algorithm is given. Numerical tests showed that it provided higher accuracy than Padé algorithms from [Hig05, AMH09] at similar or even lower cost.

Throughout this paper $\mathbb{C}^{n\times n}$ denotes the set of complex matrices of size $n\times n$, I denotes the identity matrix for this set, $\rho(A)$ is the spectral radius of matrix A, and $\mathbb{N}$ denotes the set of positive integers. The matrix norm $\|\cdot\|$ denotes any subordinate matrix norm, and $\|\cdot\|_\infty$ and $\|\cdot\|_1$ denote the ∞-norm and the 1-norm, respectively. Both norms are simple to compute, so they have been very used in the matrix function computation literature; particularly, the 1-norm is used in recent studies on matrix exponential computation [Hig05, AMH09], and [HT00] provides an algorithm for its estimation which will be used in this paper.

This paper is organized as follows. Section 2 presents the scaling and squaring error analysis and the improved algorithm. Section 3 deals with numerical tests and gives some conclusions. The following theorem will be used to bound the norm of matrix power series, see [SIDR11a, Th. 1].

Theorem 4 *Let $h_l(x) = \sum_{k\geq l} b_k x^k$ be a power series with radius of convergence R, and let $\tilde{h}_l(x) = \sum_{k\geq l} |b_k| x^k$. For any matrix $A \in \mathbb{C}^{n\times n}$ with $\rho(A) < R$, if a_k is an upper bound for $\|A^k\|$ ($\|A^k\| \leq a_k$), $p \in \mathbb{N}$, $1 \leq p \leq l$, $p_0 \in \mathbb{N}$ is the multiple of p with $l \leq p_0 \leq l + p - 1$, and*

$$\alpha_p = \max\{a_k^{\frac{1}{k}} : k = p, l, l+1, l+2, \ldots, p_0 - 1, p_0 + 1, p_0 + 2, \ldots, l + p - 1\}, \tag{2.83}$$

then $\|h_l(A)\| \leq \tilde{h}_l(\alpha_p)$.

Proof. Since p_0 is a multiple of p, then $p_0/p \in \mathbb{N}$ and $\|A^{p_0}\| = \|A^{pp_0/p}\| \leq \|A^p\|^{p_0/p}$. Hence, it follows that

$$\begin{aligned}
\|h_l(A)\| &\leq \sum_{k\geq l} |b_k| \|A^k\| \leq \sum_{j\geq 0} \sum_{i=l}^{l+p-1} |b_{i+jp}| \|A^p\|^j \|A^i\| \\
&\leq \sum_{j\geq 0} \left[\sum_{i=l}^{p_0-1} |b_{i+jp}| \|A^p\|^j \|A^i\| + |b_{p_0+jp}| \|A^p\|^{j+p_0/p} + \sum_{i=p_0+1}^{l+p-1} |b_{i+jp}| \|A^p\|^j \|A^i\| \right] \\
&\leq \sum_{j\geq 0} \sum_{i=l}^{l+p-1} |b_{i+jp}| \alpha_p^{pj+i} = \sum_{k\geq l} |b_k| \alpha_p^k = \tilde{h}_l(\alpha_p).
\end{aligned} \tag{2.84}$$

Theorem 4 unifies the two cases in which Theorem 1 from [SIDR11a, p. 1835] is divided, and avoids needing a bound for $\|A^{p_0}\|$ to obtain α_p, see (2.83).

2.5.2. Taylor Algorithm

Taylor approximation of order m of exponential of matrix $A \in \mathbb{C}^{n\times n}$ can be expressed as the matrix polynomial $T_m(A) = \sum_{k=0}^{m} A^k/k!$. The scaling and squaring algorithms in Taylor approximations are based on the approximation $e^A = \left(e^{2^{-s}A}\right)^{2^s} \approx (T_m(2^{-s}A))^{2^s}$ [ML03], where the nonnegative integers m and s are chosen with the aim of achieve full machine accuracy at minimal cost. Similarly, this method is applied in Padé approximation.

In [Hig08, p. 241], the author states that Padé approximations are preferred to Taylor series approximations in the context of scaling and squaring methods because they provide a given accuracy with lower computational cost. However, in [SIDR11a] the authors presented a scaling and squaring Taylor algorithm based on an improved mixed backward and forward error analysis, which was more accurate than existing state-of-the-art Padé algorithms [Hig05, AMH09] in the majority of test matrices with a lower or similar cost. Moreover, modifications to Padé algorithm had to be carried out in [AMH09, p. 983] to improve the denominator conditioning, whereas Taylor algorithms have no denominator.

In [Hig08, p. 247-248], an analysis about rounding errors and numerical stability of the scaling and squaring methods are performed. The author states that the overall effect of rounding errors in the computation by repeated squaring may be large relative to the computed exponential. This may or may not indicate instability of the algorithm, depending on the conditioning of the e^A problem at the matrix A. If A is normal, then the scaling and squaring method is guaranteed to be forward stable; hence, the square phase is innocuous

and the error in the computed exponential is consistent with the conditioning of the problem. Another case where the scaling and squaring method is forward stable corresponds to matrices with nonnegative nondiagonal entries as shown in[ACF96].

The scaling and squaring method has also a weakness when applied to block triangular matrices [DP00, DP01]. The exponential of a block 2×2 block triangular matrix A can be computed as

$$\exp\left(\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}\right) = \begin{bmatrix} e^{A_{11}} & \int_0^1 e^{A_{11}(1-s)} A_{12} e^{A_{22}s} ds \\ 0 & e^{A_{22}} \end{bmatrix},$$

where matrices A_{11} and A_{22} are square matrices. However, since matrix A_{12} appears only in the (1,2) block of e^A, then e^A depends linearly of A_{12}, and the accuracy of computing e^A should be unaffected by $\|A_{12}\|$ and should depend only on $\|A_{11}\|$ and $\|A_{22}\|$. Since s depends on $\|A\|$, when $\|A_{12}\| >> \max\{\|A_{11}\|, \|A_{22}\|\}$ the diagonal blocks A_{11} and A_{22} are overscaled with regard to the computation of $e^{A_{11}}$ and $e^{A_{22}}$, and this can affect the accuracy of computing e^A.

In [DP01] L. Diecci and A. Papini obtain improved error bounds for Padé approximations to e^A when A is block triangular. As a result, improved scaling strategies ensue which avoid some common overscaling difficulties. Later, [AMH09] presents an algorithm that reduces the overscaling problem choosing parameter s, based on the norms of low powers of matrix A instead of in $\|A\|$, and computes the diagonal elements in the squaring phase as exponentials instead of from powers of the diagonal Padé approximation for the case of triangular matrices. In [SIDR11a], estimations of norms of higher powers of matrix A (greater than or equal to m+1) are used to obtain the scaling parameter s, and similar ideas to those in [AMH09] can be used in the case of Taylor approximations to compute the diagonal elements for triangular matrices.

Algorithm 2.3 presents a general scaling and squaring Taylor algorithm for computing the matrix exponential, where the maximum allowed value of m is denoted by m_M.

Algorithm 2.3 Given a matrix $A \in \mathbb{C}^{n\times n}$ and a maximum order m_M, this algorithm computes $C = e^A$ by a Taylor approximation of order $m \leqslant m_M$.

1: Preprocessing of matrix A.
2: SCALING PHASE: Choose $m \leqslant m_M$, and an adequate scaling parameter $s \in \mathbb{N} \cup \{0\}$ for the Taylor approximation with scaling.
3: Compute $B = T_m(A/2^s)$ using (2.85)
4: **for** $i = 1 : s$ **do**
5: $\quad B = B^2$
6: **end for**
7: Postprocessing of matrix B.

The preprocessing and postprocessing steps are based on applying transformations to reduce the norm of matrix A, see [Hig08], and will not be studied in this paper.

In Step 2, the scaling phase, the optimal order of Taylor approximation $m_k \leqslant m_M$ and the scaling parameter s are chosen.

Table 2.20: Values of q_k depending on the selection of m_M.

k	0	1	2	3	4	5	6	7		
$m_M \setminus m_k$	1	2	4	6	9	12	16	20		
16	1	2	2	3	3	4	4			
20	1	2	2	3	3	4	4	4		
25	1	2	2	3	3	4	4	5	5	
30	1	2	2	3	3	4	4	5	5	5

For the evaluation of $T_m\left(2^{-s}A\right)$ in Step 3 we use the modified Horner and Paterson–Stockmeyer's method proposed in [SIDR11a, p. 1836-1837]. From [SIDR11a, p. 6454-6455] matrix polynomial $T_m(2^sA)$ can be computed optimally for m in the set $\mathbb{M} = \{1, 2, 4, 6, 9, 12, 16, 20, 25, 30, \ldots\}$, where we denote the elements of $\mathbb{M}$ as m_0, m_1, $m_2, \ldots$, respectively, by using Paterson-Stockmeyer's method [PS73], see [Hig08, p. 72–74] for a complete description. First, matrix powers A^2, $A^3, \cdots, A^q$ are computed, where $q = \lceil\sqrt{m_k}\rceil$ or $\lfloor\sqrt{m_k}\rfloor$, both values dividing m_k and giving the same cost [Hig08, p. 74].Then, the truncated Taylor series is computed using (10) of [SIDR11a, p. 1837], which includes the matrix scaling in Taylor series coefficients and saves some divisions of matrix A by scalar in the next way

$$
\begin{aligned}
T_m\left(2^{-s}A\right) = \Bigg\{\Bigg\{\cdots\Bigg\{\Bigg(\frac{A_q}{2^s m} + A_{q-1}\Bigg)\Big/\left[2^s(m-1)\right] + A_{q-2}\Bigg\}\Big/\left[2^s(m-2)\right] + \cdots A_2\Bigg\}\Big/\left[2^s(m-q+2)\right] + A \\
+ 2^s(m-q+1)I\Bigg\}\frac{A_q}{2^{2s}(m-q+1)(m-q)} + A_{q-1}\Bigg\}\Big/\left[2^s(m-q-1)\right] + A_{q-2}\Bigg\} \\
\Big/\left[2^s(m-q-2)\right] + \cdots + A_2\Bigg\}\Big/\left[2^s(m-2q+2)\right] + A + 2^s(m-2q+1)I\Bigg\} \\
\times \frac{A_q}{2^{2s}(m-2q+1)(m-2q)} + \cdots + A_2\Bigg\}\Big/\left[2^s(q+2)\right] + A + 2^s(q+1)I\Bigg\} \\
\times \frac{A_q}{2^{2s}(q+1)q} + A_{q-1}\Bigg\}\Big/\left[2^s(q-1)\right] + \cdots + A_2\Bigg\}\Big/\left[2^s 2\right] + A\Bigg\}\Big/2^s + I.
\end{aligned}
\tag{2.85}
$$

Analogously to Sastre *et al.* [SIDR11a], in the proposed scaling algorithm it will be necessary that the same powers of A are used for the two last orders m_{M-1} and m_M, i.e. $A^i, i = 2, 3, \ldots, q$. For each value of m_M Table 2.20 shows the selected optimal values of q for orders m_k, $k = 0, 1, 2, \ldots, M$, denoted by q_k. For example, if $m_M = 20$ and $m_4 = 9$ is the optimal order obtained in the scaling phase, then $q_4 = 3$.

Taking into account Table 4.1 from [Hig08, p. 74], the total cost of evaluating $T_{m_k}(2^{-s}A)$ in terms of matrix products for $k = 0, 1, \ldots$, denoted by Π_{m_k}, is $\Pi_{m_k} = k$. Finally, after the evaluation of $T_m\left(2^{-s}A\right)$, s repeated squarings are applied in Steps 4-6. Thus, the computational cost of Algorithm 2.3 in terms of matrix products is $\mathrm{Cost}(m_k, s) = k + s$.

2.5.2.1. Roundoff error analysis

The main contributions of this paper are concerned with the selection of m and s in the scaling phase, where the roundoff error in the computation of (2.85) will play an important role. The roundoff error can be studied by using a componentwise analysis [Fas93, pp. 18-19]. If we denote $|A| = (|a_{ij}|)_{n\times n}$, then

1. $|fl(A+B)-(A+B)| < u\,|A+B|\,,\ A,B \in \mathbb{C}^{n\times n}$,
2. $|fl(AB)-AB| < nu\,|A||B|$,
3. $\left|fl\left(\sum_{k=0}^{m} p_k A^k\right) - \sum_{k=0}^{m} p_k A^k\right| \dot{\leq}\ m(n+1)\sum_{k=0}^{m} |p_k|\,|A|^k$,

where $\dot{\leq}$ denotes the inequality avoiding the terms of order greater than or equal to u^2, where $u=2^{-53}$ is the unit roundoff in IEEE double precision arithmetic. Taking into account these properties, it is straightforward to prove that the roundoff error for computing $T_m\left(2^{-s}A\right)$ by using (2.85) verifies $|fl\left(T_m(2^{-s}A)\right) - T_m(2^{-s}A)| \dot{\leqslant} \varphi(m,n)uT_m(2^{-s}|A|)$, where u is the unit roundoff, $m \in \mathbb{M}$, and for large n, asymptotically it follows that

$$\varphi(m,n) = mn, m \geq 2. \tag{2.86}$$

Hence if we use the 1–norm, then

$$\frac{\|fl\left(T_m(2^{-s}A)\right) - T_m(2^{-s}A)\|_1}{T_m(2^{-s}\|A\|_1)} \leqslant \varphi(m,n)u. \tag{2.87}$$

This is a worst-case bound. If we denote the actual roundoff error in computing (2.85) from (2.87) as $\phi(m,n)u$, and noting that minimum roundoff errors of value u are expected, by (2.86), for large n in the majority of cases we can assume that

$$1 \leq \phi(m,n) \leq mn. \tag{2.88}$$

From [Hig02a, p. 52] a well-known rule of thumb to obtain realistic error estimates in (2.87) is that if the bound is $f(n)u$ then the error will be typically of order $\sqrt{f(n)}u$, and this rule of thumb can be supported by assuming that the rounding errors are independent random variables and applying the central limit theorem. By (2.86) and (2.87) the application of this rule gives

$$\phi(m,n) \approx \sqrt{mn}. \tag{2.89}$$

Rounding errors do not necessarily behave like independent random variables [Hig02a, p. 52] and there will be cases where the application of this rule will be pessimistic and others where it will be optimistic. However, in numerical results we will see that the use of (2.89) allows to reduce the cost of Algorithm 2.3 with no important effects in accuracy in the majority of cases.

2.5.2.2. Analysis of truncation error

Following [SIDR11a, p. 1836], if we denote the remainder of the truncated exponential Taylor series of $A \in \mathbb{C}^{n\times n}$ as

$$R_m(A) = \sum_{k\geq m+1} A^k/k!, \tag{2.90}$$

for a scaled matrix $2^{-s}A$, $s \in \mathbb{N}\cup\{0\}$, we can write

$$\left(T_m(2^{-s}A)\right)^{2^s} = e^A\left(I + g_{m+1}(2^{-s}A)\right)^{2^s} = e^{A+2^s h_{m+1}\left(2^{-s}A\right)}, \tag{2.91}$$

where

$$g_{m+1}(2^{-s}A) = -e^{-2^{-s}A}R_m(2^{-s}A), \tag{2.92}$$

$$h_{m+1}(2^{-s}A) = \log(I+g_{m+1}(2^{-s}A)), \tag{2.93}$$

where log denotes the principal logarithm, $h_{m+1}(X)$ is defined in the set

$$\Omega_m = \left\{X \in \mathbb{C}^{n\times n} : \rho\left(e^{-X}T_m(X) - I\right) < 1\right\}, \tag{2.94}$$

and both $g_{m+1}(2^{-s}A)$ and $h_{m+1}(2^{-s}A)$ are holomorphic functions of A in Ω_m and then commute with A. Using scalar Taylor series in (2.92) and (2.93) one gets

$$g_{m+1}(x) = \sum_{k\geq m+1} b_k^{(m)}x^k = \left(e^{h_{m+1}(x)} - 1\right) = \sum_{k\geq 1}\left(h_{m+1}(x)\right)^k/k!, \tag{2.95}$$

$$h_{m+1}(x) = \sum_{k\geq 1}\frac{(-1)^{k+1}(g_{m+1}(x))^k}{k} = \sum_{k\geq m+1} c_k^{(m)}x^k, \tag{2.96}$$

where $b_k^{(m)}$ and $c_k^{(m)}$ depend on order m. From (2.95) and (2.96) it follows that

$$b_k^{(m)} = c_k^{(m)}, \qquad k = m+1, m+2, \ldots, 2m+1, \tag{2.97}$$

and if $\|h_{m+1}(2^{-s}A)\| \ll 1$ then

$$g_{m+1}(2^{-s}A) = h_{m+1}(2^{-s}A) + (h_{m+1}(2^{-s}A))^2/2 + \cdots \approx h_{m+1}(2^{-s}A), \tag{2.98}$$

and, similarly, if $\|g_{m+1}(2^{-s}A)\| \ll 1$

$$h_{m+1}(2^{-s}A) = g_{m+1}(2^{-s}A) + (g_{m+1}(2^{-s}A))^2/2 + \cdots \approx g_{m+1}(2^{-s}A). \tag{2.99}$$

Using (2.90) and the exponential Taylor series in (2.92) it follows that

$$\begin{aligned} g_{m+1}(x) = \frac{-x^{m+1}}{(m+1)!}\Bigg\{(-1)^0\frac{1}{0!} + (-1)^1\left[\frac{1}{1!} - \frac{1}{0!(m+2)}\right]x + (-1)^2\Bigg[\frac{1}{2!} \\ - \frac{1}{1!(m+2)} + \frac{1}{0!(m+2)(m+3)}\Bigg]x^2 + \cdots + (-1)^k a_k^{(m)}x^k + \cdots, \end{aligned} \tag{2.100}$$

where by induction the general term of coefficient $a_k^{(m)}$ is

$$\begin{aligned} a_k^{(m)} = \frac{1}{k!} - \frac{1}{(k-1)!(m+2)} + \frac{1}{(k-2)!(m+2)(m+3)} - \cdots \\ + \frac{(-1)^{k-3}}{3!(m+2)(m+3)\cdots(m+k-2)} + \frac{(-1)^{k-2}}{2!(m+2)(m+3)\cdots(m+k-1)} \\ + \frac{(-1)^{k-1}}{1!(m+2)(m+3)\cdots(m+k)} + \frac{(-1)^k}{0!(m+2)(m+3)\cdots(m+k+1)}. \end{aligned} \tag{2.101}$$

Summing from the last term to the initial terms of $a_k^{(m)}$ it is easy to show that the sum of the last i terms of $a_k^{(m)}$ becomes

$$\frac{(-1)^{k-(i-1)}}{(i-1)!(m+2)(m+3)\cdots(m+k-(i-1))(m+k+1)} \tag{2.102}$$

and then it follows that

$$a_k^{(m)} = \frac{m+1}{k!\,(m+1+k)}. \tag{2.103}$$

Hence, using (2.100) it follows that

$$g_{m+1}(x) = -\sum_{k\geqslant 0} \frac{(-1)^k x^{m+1+k}}{k!m!\,(m+1+k)}, \tag{2.104}$$

and then

$$\frac{b_{m+k+1}^{(m)}}{b_{m+k}^{(m)}} = \frac{-1}{k\left(1+\frac{1}{m+k}\right)}. \tag{2.105}$$

This expression confirms the observation from [SIDR11a, p. 1838], where for the orders m_k that were proposed in the algorithm presented therein and the first 1000 series terms, using Matlab's Symbolic Math Toolbox we checked experimentally that the following expression holds

$$\frac{1}{k+1} < \left|\frac{b_{m+k+1}^{(m)}}{b_{m+k}^{(m)}}\right| < \frac{1}{k}, \quad k \geqslant 1. \tag{2.106}$$

Once obtained the closed form (2.104) for $g_{m+1}(x)$, $h_{m+1}(x)$ can be obtained using (2.96), and now we set the basis for the scaling algorithm. If we choose s so that $2^{-s}A \in \Omega_m$, see (2.94), then from (2.91) one gets that

$$\Delta A = 2^s h_{m+1}\left(2^{-s}A\right), \tag{2.107}$$

represents the backward absolute error in exact arithmetic from the approximation of e^A by Taylor series truncation with the scaling and squaring technique. If the minimum value of s is chosen so that

$$\|h_{m+1}\left(2^{-s}A\right)\| \leq \|2^{-s}A\|\,u, \tag{2.108}$$

or

$$\|g_{m+1}\left(2^{-s}A\right)\| \leq \phi(m,n)u. \tag{2.109}$$

where $\phi(m,n)u$ is the roundoff error in computing (2.85), see Section 2.5.2.1.

- If the minimum value of s is given by (2.108), then from (2.107) it follows that $\Delta A \leq \|A\|\,u$ and using (2.91) it follows that

$$\left(T_m\left(2^{-s}A\right)\right)^{2^s} = e^{A+\Delta A} \approx e^A. \tag{2.110}$$

- If the minimum value of s is given by (2.109), using (2.91), (2.92) and the exponential Taylor series it follows that

$$\|R_m\left(2^{-s}A\right)\| = \left\|e^{2^{-s}A} g_{m+1}\left(2^{-s}A\right)\right\| \leq \left\|e^{2^{-s}A}\right\| \phi(m,n)u, \tag{2.111}$$

  and, taking into account that a relative roundoff error $\phi(m,n)u$ will be introduced in the numerical evaluation of the matrix polynomial $T_m\left(2^{-s}A\right)$, there is no point in increasing the scaling parameter s or the approximation order m to reduce further the norm of Taylor remainder $R_m\left(2^{-s}A\right)$.

Table 2.21: Maximal values Θ_m such that $\tilde{h}_{m+1}(\Theta_m) \leq \Theta_m u$, maximal values $\tilde{\Theta}_m$ such that $\tilde{g}_{m+1}(\tilde{\Theta}_m) \leq \phi(m,n)u$ for $\phi(m,n) = 1$, and values $\vartheta_m = \max\{\Theta_m, \tilde{\Theta}_m\}$.

m	Θ_m	$\tilde{\Theta}_m$ for $\phi(m,n) = 1$	ϑ_m
1	2.220446049250264e-16	1.490116111983279e-8	1.490116111983279e-8
2	2.580956802971767e-8	8.733457513635361e-6	8.733457513635361e-6
4	3.397168839976962e-4	1.678018844321752e-3	1.678018844321752e-3
6	9.065656407595101e-3	1.773082199654024e-2	1.773082199654024e-2
9	8.957760203223343e-2	1.137689245787824e-1	1.137689245787824e-1
12	2.996158913811581e-1	3.280542018037257e-1	3.280542018037257e-1
16	7.802874256626574e-1	7.912740176600240e-1	7.912740176600240e-1
20	1.438252596804337	1.415070447561532	1.438252596804337
25	2.428582524442827	2.353642766989427	2.428582524442827
30	3.539666348743690	3.411877172556771	3.539666348743690

Using (2.96) and (2.104), Matlab's Symbolic Math Toolbox, high precision arithmetic, 200 series terms and a zero finder we obtained the maximal values Θ_m of $\Theta = \|2^{-s}A\|$, shown in Table 2.21, such that, using notation of Theorem 4

$$\|h_{m+1}(2^{-s}A)\| \leq \tilde{h}_{m+1}(\Theta) = \sum_{k\geq m+1} |c_k^{(m)}|\Theta^k \leq \Theta u. \tag{2.112}$$

In a similar way, for a given value of $\phi(n,m)$ the maximal values $\tilde{\Theta}_m$ such that

$$\|g_{m+1}(2^{-s}A)\| \leq \tilde{g}_{m+1}(\Theta) = \sum_{k\geq m+1} b_k^{(m)}\Theta^k \leq \phi(m,n)u, \tag{2.113}$$

can be easily obtained. Table 2.21 shows $\tilde{\Theta}_m$ values for the more restrictive case from (2.88), i.e. when $\phi(m,n) = 1$, which is the case used in the error analysis of [SIDR11a, p. 1836]. Hence, if $\|2^{-s}A\| \leq \vartheta_m$ where $\vartheta_m = \max\{\Theta_m, \tilde{\Theta}_m\}$ then (2.108) or (2.109) hold. For the values of m where $\Theta_m > 1$ in Table 2.21, i.e. $m = 20$, 25 and 30, it follows that $\vartheta_m = \Theta_m$. We recall bound (9) of [SIDR11a, p. 1836], which holds for those values of m and will be needed in the scaling algorithm:

$$\|h_{m+1}(2^{-s}A)\| \leq \tilde{h}_{m+1}(\|2^{-s}A\|) = \tilde{h}_{m+1}(\Theta) \leq \Theta u, \quad 0 \leq \Theta \leq \Theta_m. \tag{2.114}$$

2.5.2.3. Scaling algorithm

The new proposed scaling algorithm has some improvements with respect to that proposed in [SIDR11a, p. 1837-1838] that we describe in this section. For all norms appearing in the scaling algorithm we will use the 1–norm, and from Algorithm 2.3 recall that m_M is the maximum allowed Taylor order. Using the same bounds and a similar process that we use in the proposed scaling algorithm described below, we will first verify if any of the lower Taylor optimal orders $m_k = 1, 2, 4, \ldots, m_{M-1}$ satisfy (2.108) or (2.109) without scaling, i.e. with $s = 0$. If not, the optimal scaling parameter $s \geq 0$ for order m_M is computed. This computation has two phases: Calculation of an initial value of the scaling parameter s_0, and the refinement of this value to test if it can be reduced. In this paper the main improvement with respect to the algorithm from [SIDR11a] is applied to the refinement of the scaling parameter, where roundoff error and function $\phi(m,n)$ from (2.89) will play an important role.

The calculation of the initial scaling parameter s_0 remains practically unchanged except for the use of the new Theorem 4 instead of Theorem 1 from [SIDR11a], and it is described for clarity in the following section.

2.5.2.4. Calculation of initial scaling s_0

First, the 1–norm estimate of $\|A^{m_M+1}\|$ is computed using the block 1–norm estimation algorithm of [HT00]. For a $n \times n$ matrix this algorithm carries out a 1-norm power iteration whose iterates are $n \times t$ matrices, where t is a parameter that has been taken to be 2, see [AMH09, p. 983]. Hence, the estimation algorithm has $O(n^2)$ computational cost, negligible compared to matrix products, whose cost is $O(n^3)$.

Similarly to [SIDR11a, p. 1837], the upper bounds a_k for $\|A^k\|$ needed to apply Theorem 4 in (2.112) and (2.113) are obtained using products of norms of matrix powers estimated for current and previous tested orders, i.e. $\|A^{m_k+1}\|$, $k = 0, 1, 2, \ldots, M$, and the powers of A computed for evaluation of $T_{m_M}(2^{-s}A)$, A^i, $i = 1, 2, \ldots, q$, as

$$\begin{aligned} \left\|A^k\right\| \le a_k \;=\; \min\Big\{ & \|A\|^{i_1} \left\|A^2\right\|^{i_2} \cdots \|A^q\|^{i_q} \left\|A^{m_1+1}\right\|^{i_{m_1+1}} \left\|A^{m_2+1}\right\|^{i_{m_2+1}} \cdots \\ & \times \left\|A^{m_M+1}\right\|^{i_{m_M+1}} : i_1 + 2i_2 + \cdots + qi_q + (m_1+1)i_{m_1+1} \\ & + (m_2+1)i_{m_2+1} + \cdots + (m_M+1)i_{m_M+1} = k \Big\}, \end{aligned} \tag{2.115}$$

and a simple Matlab function was provided in [SIDR11a] to obtain a_k, see nested function **powerbound** from **exptayns.m** available at `http://personales.upv.es/~jorsasma/Software/exptayns.m`.

Then, we seek for the minimum value of α_p from Theorem 4 with $l = m_M + 1$, denoted by $\alpha_{\min}$, obtaining successively α_p for $p = 2, 3, \ldots, q, m_1+1, m_2+1, \ldots, m_M+1$, stopping the process for that value of p such that

$$a_p^{1/p} \le \max\{a_k^{1/k} : k = m+1, m+2, \ldots, p_0-1, p_0+1, p_0+2, \ldots, m+p\}, \tag{2.116}$$

where p_0 is the multiple of p with $m+1 \le p_0 \le m+p$. In the following we show that if condition (2.116) holds, then for $p' > p$ it follows that $\alpha_{p'} \ge \alpha_p$:

By (2.115) one gets $a_{p_0} \leqslant a_p^{p_0/p}$, and then using (2.116) it follows that

$$a_{p_0}^{1/p_0} \le a_p^{1/p} \le \max\{a_k^{1/k} : k = m+1, m+2, \ldots, p_0-1, p_0+1, p_0+2, \ldots, m+p\}, \tag{2.117}$$

and then

$$\begin{aligned} \alpha_p &= \max\{a_k^{1/k} : k = m+1, m+2, \ldots, p_0-1, p_0+1, p_0+2, \ldots, m+p\} \\ &= \max\{a_k^{1/k} : k = m+1, m+2, \ldots, m+p\}. \end{aligned} \tag{2.118}$$

Hence, for $p' > p$, if

$$a_{p'}^{1/p'} \le \max\{a_k^{1/k} : k = m+1, m+2, \ldots, p_0'-1, p_0'+1, p_0'+2, \ldots, m+p'\}, \tag{2.119}$$

where p_0' is a multiple of p', then in a similar way it follows that

$$\begin{aligned} \alpha_{p'} &= \max\{a_k^{1/k} : k = m+1, m+2, \ldots, m+p'\} \\ &\geq \max\{a_k^{1/k} : k = m+1, m+2, \ldots, m+p\} = \alpha_p. \end{aligned} \tag{2.120}$$

Otherwise, if (2.119) is not verified, as $a_{p'}^{1/p'} \geq a_{p_0'}^{1/p_0'}$ it follows that

$$\begin{aligned} \alpha_{p'} &= a_{p'}^{1/p'} > \max\{a_k^{1/k} : k = m+1, m+2, \ldots, m+p'\} \\ &\geq \max\{a_k^{1/k} : k = m+1, m+2, \ldots, m+p\} = \alpha_p, \end{aligned} \tag{2.121}$$

and then, by (2.120) and (2.121) $\alpha_{p'} \geq \alpha_p$.

Once obtained $\alpha_{\min}$, we take the appropriate initial minimum scaling parameter $s_0 \geq 0$ so that $2^{-s_0}\alpha_{\min} \leq \vartheta_{m_M}$, i.e.

$$s_0 = \begin{cases} 0, & \text{if } \alpha_{\min} \leq \vartheta_{m_M}, \\ \lceil \log_2(\alpha_{\min}/\vartheta_{m_M}) \rceil, & \text{if } \alpha_{\min} > \vartheta_{m_M}. \end{cases} \tag{2.122}$$

Then, if $\vartheta_{m_M} = \max\{\Theta_{m_M}, \tilde{\Theta}_{m_M}\} = \tilde{\Theta}_{m_M}$ using Theorem 4 and (2.113), taking for simplicity from now on $m = m_M$, it follows that

$$\|g_{m+1}\left(2^{-s_0}A\right)\| \leq \tilde{g}_{m+1}\left(2^{-s_0}\alpha_{\min}\right) \leq \tilde{g}_{m+1}\left(\tilde{\Theta}_m\right) \leq u, \tag{2.123}$$

and (2.109) holds. Taking into account that $\|A^k\|^{1/k} \leq \|A\|$, from (2.115) it follows that $a_k^{1/k} \leq (\|A\|^k)^{1/k} = \|A\|$. Thus, $\alpha_{\min}$ from Theorem 4 satisfies $\alpha_{\min} \leq \|A\|$. Hence, if $\vartheta_m = \Theta_m$ from Table 2.21 it follows that $m = 20, 25$ or 30, and using (2.114) one gets

$$\|h_{m+1}\left(2^{-s_0}A\right)\| \leq \tilde{h}_{m+1}\left(2^{-s_0}\alpha_{\min}\right) \leq 2^{-s_0}\alpha_{\min}\, u \leq 2^{-s_0}\|A\|u, \tag{2.124}$$

and (2.108) holds.

After obtaining the initial value of the scaling parameter s_0, it will be refined as described in the following section.

2.5.2.5. Scaling refinement

In this section bounds for $\|g_{m+1}(2^sA)\|$ and $\|h_{m+1}(2^{-s}A)\|$ of the same type of (14) and (15) from [SIDR11a, p. 1838] are used for the scaling parameter refinement. Both series from (2.96) and (2.104) will be truncated with the same number of terms as in Section 3 of [SIDR11a, p. 1839], i.e. $q+2$ where q takes the values from Table 2.20. Taking into account (2.97) one gets that the first $m+1$ coefficients of $g_{m+1}(2^sA)$ and $h_{m+1}(2^{-s}A)$ are equal, and if q takes the values from Table 2.20 it follows that $m+1 \geq q+2$ for $m \geq 4$. Then, using (2.95)–(2.97) we take

$$\|g_{m+1}(2^sA)\| \approx \left\|\sum_{k=m+1}^{m+q+2} b_k^{(m)}(2^sA)^k\right\| = \left\|\sum_{k=m+1}^{m+q+2} c_k^{(m)}(2^sA)^k\right\| \approx \|h_{m+1}(2^sA)\|, \tag{2.125}$$

and the refinement will be considered for $m \geq 4$.

Hence, once the initial value s_0 of the scaling parameter is obtained, if $s_0 \geq 1$ we test if (2.108) or (2.109) hold with the reduced scaling parameter $s = s_0 - 1$, using the bounds for $\left\|A^k\right\| \leq a_k$ from (2.115) and testing if bound

$$\sum_{k=m+1}^{m+q+2} \left|c_k^{(m)}\right| a_k / 2^{sk} \leq \max\left\{\phi(m,n), \left\|2^{-s}A\right\|\right\} u, \tag{2.126}$$

holds taking $\phi(m,n) = \sqrt{nm}$ from (2.89). Note that we stop the series summation if the first term does not verify the bound. If the sum of one or more terms is lower than the bound but the complete truncated series sum is not, we can estimate $\|A^{m+2}\|$ to improve the bound a_{m+2} and check if (2.126) holds then, see [SIDR11a]. If (2.126) does not hold with $s = s_0 - 1$, then we check if next bound holds

$$\frac{\left\|A^{m+1}\right\|}{2^{s(m+1)}} \left\| \sum_{k=m+1}^{m+q+1} c_k^{(m)} \left(2^{-s}A\right)^{k-m-1} \right\| + \left|c_{m+q+2}^{(m)}\right| \frac{a_{m+q+2}}{2^{(m+q+2)s}} \leq \max\left\{\phi(m,n), \left\|2^{-s}A\right\|\right\} u, \tag{2.127}$$

where $\phi(m,n) = \sqrt{nm}$, matrix powers A^i, $i = 2,3,\ldots,q$ from the computation of $T_m\left(2^{-s}A\right)$ by (2.85) are reused, and we can divide by the coefficient of A to save the product of matrix A by a scalar. In a similar way to [SIDR11a, p. 1838], lower bounds for expression (2.127) to avoid its unnecessary evaluation can be easily obtained.

If any of both bounds (2.126) or (2.127) holds with $s_0 - 1$ then repeat the process with $s = s_0 - 2, s_0 - 3, \ldots$ Note that computational cost of evaluating (2.126) and (2.127) is $O(n^2)$, negligible when compared to matrix product costs, which are $O(n^3)$.

Note that from Table 2.21 for $m_k \geq 20$ it follows that $\vartheta_{m_k}/2 < \vartheta_{m_{k-1}}$. Thus, if the last value of s where (2.126) or (2.127) hold is $s \geq 1$ and $\vartheta_{m_M}/2 < \vartheta_{m_{M-1}}$ it is possible that order m_{M-1} also verifies (2.126) or (2.127) with the same value of scaling s, see [SIDR11a]. Hence, if final resulting scaling is $s \geq 1$ bounds (2.126) and/or (2.127) should be tested with the same scaling parameter s and order m_{M-1}. Finally, the algorithm returns s and the minimum order m_M or m_{M-1} satisfying (2.126) or (2.127). It is possible to evaluate $T_m(2^{-s}A)$ with both orders with optimal number of matrix products at this point because we set in its evaluation that both last orders used the same matrix powers of A [SIDR11a].

Summarizing, the complete scaling algorithm from Step 2 in Algorithm 2.3 consists of:

1. Check if one of optimal orders $m = 4, 6, \ldots, m_{M-1}$, $m \in \mathbb{M}$ satisfies (2.126) or (2.127) with $s = 0$ using the 1–norm estimate of $\|A^{m+1}\|$ from [HT00] and reusing the matrix powers A^i, $i = 2,3,\ldots,q$ needed to compute Taylor matrix polynomial for each tested value of m, returning $s = 0$ and the order m which satisfies (2.126) or (2.127) if it exists.

2. If there is no value of $m \leq m_{M-1}$ that satisfies (2.126) or (2.127) with $s = 0$ then obtain an initial value s_0 of the scaling parameter with order m_M as described in Section 2.5.2.4.

3. If $s_0 > 0$ refine the selection of s_0 testing if (2.126) or (2.127) hold with $s = s_0 - 1$, $s_0 - 2, \ldots$, returning the lowest value of s that verifies (2.126) or (2.127).

4. If the final selection of s is not zero, test if (2.126) or (2.127) hold with s and m_{M-1}, returning m_{M-1} if (2.126) or (2.127) hold with it, or otherwise returning m_M.

Maximum order $m_M = 30$ is recommended as it obtained the highest accuracy results in [SIDR11a]. For more details about the implementation we made available online a fully commented Matlab version of the complete Algorithm 2.3, denoted by **exptaynsv2**, in `http://personales.upv.es/~jorsasma/Software/exptaynsv2.m`, where changes in the selection of order m and scaling parameter s are avoided when no cost reductions are achieved with respect to the original algorithm **exptayns** from [SIDR11a].

2.5.2.6. New bounds for $\|g_{m+1}(2^s A)\|$ and $\|h_{m+1}(2^{-s} A)\|$

This section provides bounds for the complete series of $\|g_{m+1}(2^{-s} A)\|$ without truncation. These bounds are used to provide bounds for $\|h_{m+1}(2^{-s} A)\|$. From (2.104), if $B \in \mathbb{C}^{n\times n}$ and $r \in \mathbb{N} \cup \{0\}$, then

$$\|g_{m+1}(B)\| \leq \frac{1}{m!} \left\| \sum_{k=0}^{k=r} \frac{(-1)^k B^{m+1+k}}{k!\,(m+1+k)} \right\| + \frac{\left\|B^{m+r+2}\right\|}{m!(r+1)!(m+r+2)} + \|f_{m+2+r}(B)\|, \tag{2.128}$$

where

$$\|f_{m+2+r}(B)\| = \frac{1}{m!} \left\| \sum_{k\geq r+2} \frac{(-1)^k B^{m+1+k}}{k!\,(m+1+k)} \right\| \leq \frac{1}{m!(m+r+3)} \sum_{k\geq r+2} \frac{\left\|B^{m+k}\right\|}{k!\left(\frac{m+1+k}{m+r+3}\right)}. \tag{2.129}$$

If

$$\vartheta_{m_M} = \max\{\Theta_{m_M}, \tilde{\Theta}_{m_M}\} = \tilde{\Theta}_{m_M}, \tag{2.130}$$

using α_{min} from Section 2.5.2.4, it follows that

$$\|f_{m+2+r}(2^{-s} A)\| \leq \frac{1}{m!(m+r+3)} \sum_{k\geq r+2} \frac{(2^{-s}\alpha_{min})^{m+k}}{k!\left(\frac{m+1+k}{m+r+3}\right)} \tag{2.131}$$

$$\leq \frac{(2^{-s}\alpha_{min})^{m+r+2}}{m!(m+r+3)(r+2)!} \left[1 + \frac{2^{-s}\alpha_{min}}{(r+3)\left(1+\frac{1}{m+r+3}\right)} + \frac{(2^{-s}\alpha_{min})^2}{(r+3)(r+4)\left(1+\frac{2}{m+r+3}\right)} + \cdots \right]$$

$$\leq \frac{(2^{-s}\alpha_{min})^{m+r+2}}{m!(m+r+3)(r+2)!} \sum_{j\geq 0} \beta^j = \frac{(2^{-s}\alpha_{min})^{m+r+2}}{m!(m+r+3)(r+2)!} \frac{1}{1-\beta}, \tag{2.132}$$

where

$$\beta = \frac{2^{-s}\alpha_{min}}{(r+3)\left(1+\frac{1}{m+r+3}\right)}, \tag{2.133}$$

must verify $\beta < 1$, and the binomial theorem has been used taking into account $(1+x)^j \geq 1+jx$ for $x > 0$. If a sharper bound is needed, using (2.131) note that

$$\|f_{m+2+r}(2^{-s} A)\| \leq \frac{1}{m!(m+r+3)} \sum_{k\geq r+2} \frac{(2^{-s}\alpha_{min})^{m+k}}{k!\left(\frac{m+1+k}{m+r+3}\right)}$$

$$\leq \frac{(2^{-s}\alpha_{min})^{m+1}}{m!(m+r+3)} \sum_{k\geq r+2} \frac{(2^{-s}\alpha_{min})^k}{k!}$$

$$\leq \frac{(2^{-s}\alpha_{min})^{m+1}}{m!(m+r+3)} \left[e^{2^{-s}\alpha_{min}} - \sum_{k=0}^{r+1} \frac{(2^{-s}\alpha_{min})^k}{k!} \right], \tag{2.134}$$

and an accurate enough approximation to this bound can be obtained using for instance Matlab's exponential function **exp** if r is not large and $2^{-s}\alpha_{min}$ is not very small. Taking $r = q$, as in Section 2.5.2.5, and using Table 2.20, it follows that $q \leq 5$. On the other hand, for $m_M \geq 16$, taking into account (2.122), (2.130) and Table 2.21 it follows that

$$2^{-s}\alpha_{min} \geq \tilde{\Theta}_{16}/2 \approx 0.3956, \tag{2.135}$$

which is not so small to produce problems in the evaluation of bound (2.131). For instance, with the maximum value of q and the minimum value of $2^{-s}\alpha_{min}$ using Matlab we obtained that

$$\exp\left(\tilde{\Theta}_{16}/2\right) - \sum_{k=0}^{6} \left(\tilde{\Theta}_{16}/2\right)^k /k! = 3.16626365 \times 10^{-7}, \tag{2.136}$$

obtaining the same result for the 9 decimal digits shown using Matlab's Symbolic Math Toolbox and high precision arithmetic.

If $\vartheta_{m_M} = \max\{\Theta_{m_M}, \tilde{\Theta}_{m_M}\} = \Theta_{m_M}$ then using (2.96), for $\|g_{m+1}(2^{-s}A)\| \ll 1$ it follows that

$$\|h_{m+1}(2^{-s}A)\| \leq \|g_{m+1}(2^{-s}A)\| + O\left[\|g_{m+1}(2^{-s}A)\|^2\right] \approx \|g_{m+1}(2^{-s}A)\|, \tag{2.137}$$

If $\|g_{m+1}(2^{-s}A)\| < 1$ is not small, using Taylor series of function $\log(1-x)$ and (2.96) it follows that

$$\|h_{m+1}(2^{-s}A)\| \leq -\log(1 - \|g_{m+1}(2^{-s}A)\|). \tag{2.138}$$

2.5.3. Numerical experiments and conclusions

This section compares the Matlab implementation of the proposed Taylor algorithm, denoted by **extaynsv2**, with the original function **exptayns** from [SIDR11a] (`http://personales.upv.es/~jorsasma/Software/exptayns.m`), and functions **expm** and **expm_new** which implement Padé algorithms from [Hig05] and [AMH09], respectively. Algorithm accuracy was tested by computing the relative error $\mathrm{E} = \|e^A - \tilde{Y}\|_1 / \|e^A\|_1$, where $\tilde{Y}$ is the computed approximation. Cost was given in terms of matrix products. The asymptotic cost in terms of matrix products for solving the multiple right-hand side linear system appearing in Padé algorithms was taken 4/3 [BD99]. We were interested in testing a wide range of matrices (diagonalizable and nondiagonalizable matrices) with a considerable dimension (between 128 and 1024), and whose exponentials could be computed accurately using orthogonal transformations. For this reason we chose the sets of matrices (1) and (2). The matrices of set (3) appear in the state of the art in the exponential matrix computation [Hig05, AMH09]. These sets of matrices are described below:

1. 3 sets of one hundred diagonalizable matrices of sizes 128, 256 and 1024, respectively. These matrices have the form $V^T DV$, where D is a diagonal matrix whose diagonal elements are random values between $-k$ and k with different integer values of k, and V is an orthogonal matrix obtained as $V = H/16$, where H is the Hadamard matrix.

2. 3 sets of one hundred matrices with multiple eigenvalues of sizes 128, 256 and 1000, respectively. These matrices have the form $V^T DV$, where D is a block diagonal matrix whose diagonal blocks are Jordan blocks with random dimension and random

Table 2.22: Comparison of functions **exptaynsv2** (labelled 2) and **exptayns** (labelled 1) with maximum order $m_M = 30$: Maximum and minimum values of the matrix norm for each matrix set, maximum and minimum relative error ratios E_2/E_1, maximum and minimum relative error ratio E_1/u where u is the unit roundoff, number N of matrices where **exptaynsv2** cost is one matrix product lower than **exptayns**.

	diag 128	diag 256	diag 1024	Jord 128	Jord 256	Jord 1024
máx$\{\|A\|_1\}$	6.07e1	3.75e2	7.28e2	2.14e2	2.43e2	1.91e6
mín$\{\|A\|_1\}$	4.65e1	3.59	7.21	5.27	5.93	1.20e3
máx$\{E_2/E_1\}$	1.02	1.01	1.00	1.12	1.06	1
mín$\{E_2/E_1\}$	0.99	0.98	1.00	0.99	0.89	1
máx$\{E_1\}/u$	2.28e1	6.34e1	2.13e2	1.54e3	6.89e3	7.22e11
mín$\{E_1\}/u$	9.82	1.14	8.71	2.38	3.53	4.03e2
$N(\%)$	67	10	6	11	12	0

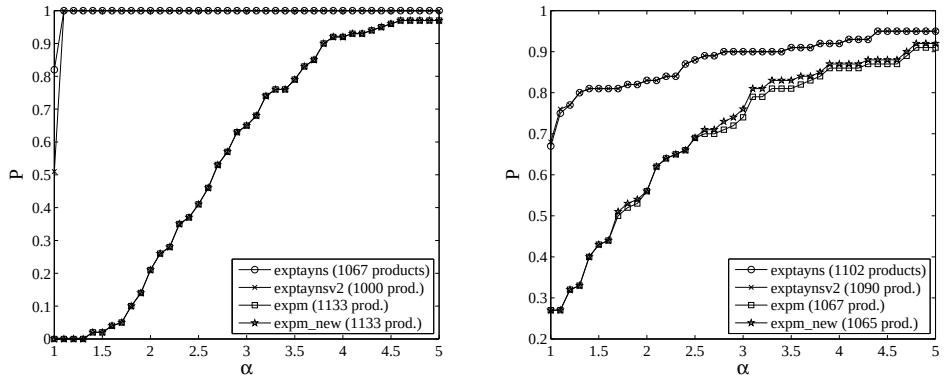
eigenvalues between -50 and 50, and V is an invertible matrix with random values in $[-0.5, 0.5]$ for size 1000, and an orthogonal matrix obtained as $V = H/16$, where H is the Hadamard matrix for sizes 128 and 256.

3. 43 25×25 matrices, 39 100×100 and 32 1000×1000 from the function **matrix** from the Matrix Computation Toolbox [Hig02b]. Matrices whose exponential cannot be represented in double precision due to overflow were excluded from all the matrices given by function **matrix** for each size.

The "exact" value of matrix exponential e^A was computed using Matlab's Symbolic Math Toolbox or quadruple precision in Fortran, by using the following methods:

- For matrix sets 1 and 2: $e^A = V^T e^D V$, where $V^T e^D V$ was computed by using the **vpa** function of Matlab with 32 decimal digit precision.
- For matrix set 3: Taylor method with different orders and scaling parameters for each matrix to check the result correctness, using quadruple precision for 1000×1000 matrices, and 128 decimal digit precision for the remaining matrices.

Table 2.22 presents the results for sets (1) and (2) showing that maximum and minimum **exptaynsv2** and **exptayns** relative error ratios E_2/E_1, where E_2 is the relative error with function **exptaynsv2** and E_1 is the relative error for **exptayns**, are very close to unity for all matrix sets. This is verified even for the first set, whose norm range is such that **exptaynsv2** saves one matrix product for 67 % matrices. In the remaining sets, which have a greater norm variability, the percentage of matrices where **exptaynsv2** saves products is lower, between 0 and 12 %. Table 2.22 also shows that the relative error tends to increase with the matrix size, as expected. Figures 2.6a and 2.6b show the costs in terms of matrix products and the performance profiles [DM02] of **exptaynsv2**, **exptayns**, **expm_new** and **expm** for the two sets of the test matrices, where α coordinate varies between 1 and 5 in steps equal to 0.1, and p coordinate is the probability that the considered method has a relative error lower than or equal to α-times the smallest error over all the methods, where probabilities are defined over all matrices. Taylor functions had the highest accuracies with costs similar or even lower than **expm_new**.



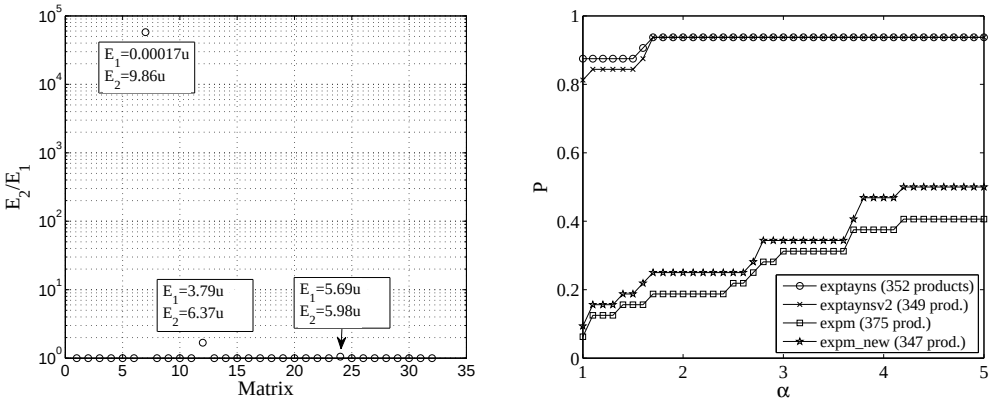
(a) Diagonalizable matrices $n = 128$. (b) Jordan matrices $n = 256$.

Figure 2.6: Performance profile and cost in terms of matrix products for the 128×128 diagonalizable and 256×256 Jordan matrices from sets 1 and 2, and $m_M = 30$ in Taylor functions.

Figure 2.7a presents the relative error ratios E_2/E_1 for the 1000×1000 matrices from set 3, showing that there are three matrices (9.38 %) where **exptaynsv2** saved one matrix product. It also shows that there is one matrix where E_2/E_1 was significant. It can be explained because for that matrix $E_1 = 0.00017u$ was much lower than the minimum expected error, i.e. the roundoff error u. In the remaining two cases E_2 and E_1 had the same order. Figure 2.7b gives the same conclusions as Figures 2.6a and 2.6b.

In order to test the most critical cases for function **exptaynsv2** with respect to error, we multiplied each matrix A_i from the 25×25 matrices from test set 3, $1 \leq i \leq 43$, by a different constant $t_i \geq 1$ such that **exptaynsv2** cost for matrix $t_i \times A_i$ was one matrix product lower than the cost of the same function with matrix $(t_i + 0.01) \times A_i$. The same was done with the 100×100 matrices from the same set. For the majority of the new matrices $t_i \times A_i$, **exptaynsv2** cost was one matrix product lower than **exptayns** cost, showing that the new proposed bounds based on $\phi(n, m)u = \sqrt{mn}u$ were verified, while the original bounds from [SIDR11a] based on u were not. Maximum relative error differences between both functions were then expected for those matrices. Figure 2.8 presents the results for the modified sets of matrices sized 25 and 100, and similar conclusions to those from Figures 2.7a and 2.7b can be obtained, with the only difference that in the new matrix sets there were some cases where E_2 was significantly lower than E_1.

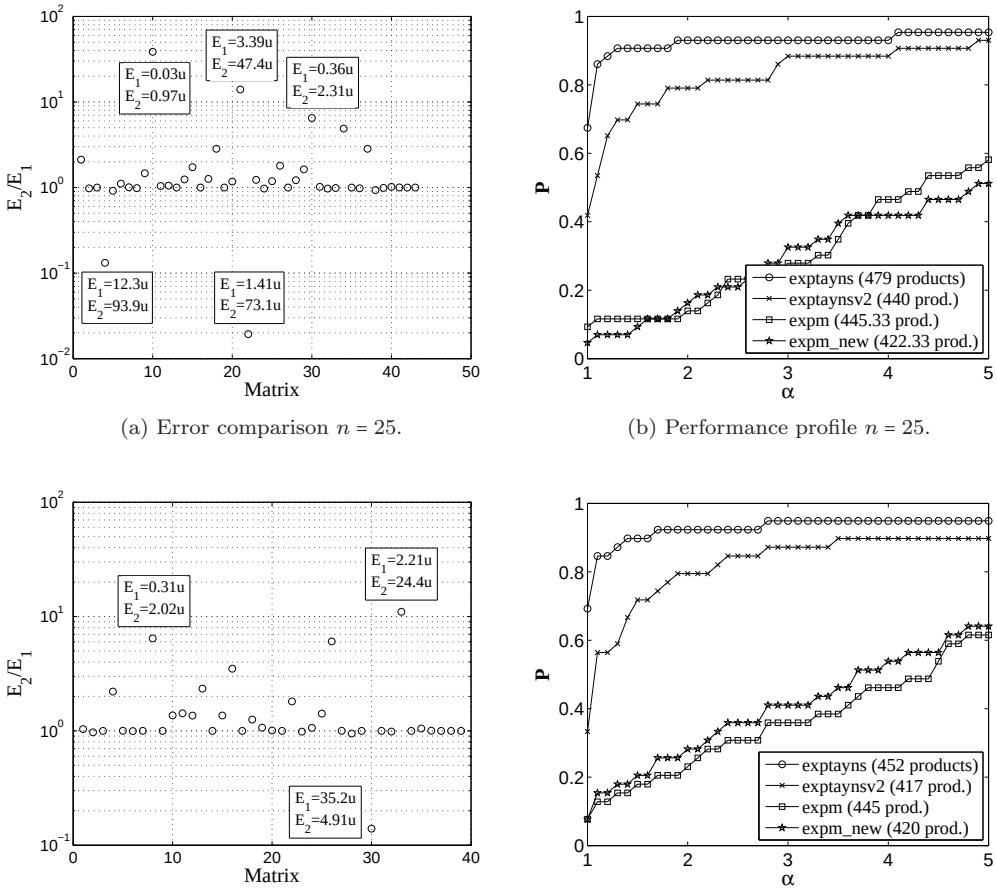
To sum up, a competitive modification of the Taylor algorithm from [SIDR11a] has been proposed based on increasing the allowed forward error bound depending on the matrix size and Taylor order. The proposed modification reduces the cost with a small impact on accuracy, being more accurate than Padé existing state-of-the-art algorithms in the majority of tests with similar or even lower cost.



(a) Error comparison $n = 1000$.

(b) Performance profile $n = 1000$.

Figure 2.7: Relative error ratios E_2/E_1 for functions **exptayns** (E_1) and **exptaynsv2** (E_2), performance profile and cost in terms of matrix products, for $m_M = 30$ and the 1000×1000 matrices from the Matrix Computation Toolbox.



(a) Error comparison $n = 25$.

(b) Performance profile $n = 25$.

(c) Error comparison $n = 100$.

(d) Performance profile $n = 100$.

Figure 2.8: Relative error ratios E_2/E_1 for functions **exptayns** (E_1), **exptaynsv2** (E_2), performance profiles and cost in terms of matrix products, for $m_M = 30$ and matrices with sizes 25 and 100 from the Matrix Computation Toolbox multiplied by constants t_i, see text.

2.6. High perfomance computing of the matrix exponential

Referencia del artículo:
P. Ruiz, J. Sastre, J. Ibáñez and E. Defez
High perfomance computing of the matrix exponential
Journal of Computational and Applied Mathematics, Volume 291, 1 January 2016, Pages 370–379, ISSN 0377-0427, http://dx.doi.org/10.1016/j.cam.2015.04.001

Abstract

This work presents a new algorithm for matrix exponential computation that significantly simplifies a Taylor scaling and squaring algorithm presented previously by the authors, preserving accuracy. A Matlab version of the new simplified algorithm has been compared with the original algorithm, providing similar results in terms of accuracy, but reducing processing time. It has also been compared with two state-of-the-art implementations based on Padé approximations, one commercial and the other implemented in Matlab, getting better accuracy and processing time results in the majority of cases.

2.6.1. Introduction

Matrix function computation has received remarkable attention in the last decades due to its usefulness in a great variety of engineering problems. Especially noteworthy is the matrix exponential, which emerge in the solution of systems of linear differential equations in numerous applications and a large number of methods for its computation have been proposed [ML03, Hig08]. Moreover, in many cases, the resolution of these systems involve large matrices, so, not only accurate, but also efficient methods are needed. In this sense, the authors presented in [SIDR15] two modifications of a Taylor-based scaling and squaring algorithm to reduce computational costs while preserving accuracy.

In [SIDR11a] the authors presented a scaling and squaring Taylor algorithm based on an improved mixed backward and forward error analysis, which was more accurate than existing state-of-the-art algorithms for matrix exponential such as that in [AMH09], in the majority of test matrices with a lower or similar cost. Subsequently, in [SIDR14], the authors gave a new formula for the forward relative error of matrix exponential Taylor approximation and proposed to increase the allowed forward error bound depending on the matrix size and the Taylor approximation order. This algorithm reduces the computational cost in exchange for a small impact in accuracy. In this work, we present a new algorithm that significantly simplifies the one presented in [SIDR11a] providing a competitive scaling and squaring algorithm for matrix exponential computation in comparison with both previous algorithms and the state-of-the-art implementations based on Padé approximations from [AMH09] and [NAG02].

Throughout this paper $\mathbb{C}^{n\times n}$ denotes the set of complex matrices of size $n\times n$, I denotes the identity matrix for this set, $\rho(A)$ is the spectral radius of matrix A, and $\mathbb{N}$ denotes

the set of positive integers. The matrix norm $\|\cdot\|$ denotes any subordinate matrix norm; in particular $\|\cdot\|_1$ and $\|\cdot\|_2$ are the 1-norm and the 2-norm, respectively. The symbols $\lceil\cdot\rceil$ and $\lfloor\cdot\rfloor$ denote the smallest following and the largest previous integer, respectively. This paper is organized as follows: Section 2.6.2 presents a general scaling and squaring Taylor algorithm; Section 2.6.3 presents the scaling and squaring error analysis; the new algorithm is given in Section 2.6.4; finally, Section 2.6.5 shows numerical results and Section 2.6.6 gives some conclusions. Next Theorem 5 from [SIDR14] and the new Theorem 6 will be used in section 2.6.3 to bound the norm of matrix power series.

Theorem 5 *Let $h_l(x) = \sum_{k\geq l} b_k x^k$ be a power series with radius of convergence R, and let $\tilde{h}_l(x) = \sum_{k\geq l} |b_k| x^k$. For any matrix $A \in \mathbb{C}^{n\times n}$ with $\rho(A) < R$, if a_k is an upper bound for $\|A^k\|$ ($\|A^k\| \leq a_k$), $p \in \mathbb{N}$, $1 \leq p \leq l$, $p_0 \in \mathbb{N}$ is the multiple of p with $l \leq p_0 \leq l+p-1$, and*

$$\alpha_p = \max\{a_k^{\frac{1}{k}} : k = p, l, l+1, l+2, \ldots, p_0-1, p_0+1, p_0+2, \ldots, l+p-1\}, \tag{2.139}$$

then $\|h_l(A)\| \leq \tilde{h}_l(\alpha_p)$.

Theorem 6 *Let $l \in \mathbb{N}$, $l \geq 1$, and let $q \in \mathbb{N}$ be the minimum value with $1 \leq q \leq l$ such that*

$$\|A^q\|^{\frac{1}{q}} \leq \max\{\|A^k\|^{\frac{1}{k}} : k = l, l+1, \ldots, q_0-1, q_0+1, q_0+2, \ldots, l+q-1\}, \tag{2.140}$$

where $q_0 \in \mathbb{N}$ is the multiple of q with $l \leq q_0 \leq l+q-1$. Then if

$$\|A^{k_0}\|^{\frac{1}{k_0}} = \max\{\|A^k\|^{\frac{1}{k}} : k = l, l+1, \ldots, q_0-1, q_0+1, q_0+2, \ldots, l+q-1\} \tag{2.141}$$

then

$$\max\{\|A^k\|^{\frac{1}{k}} : \ k \geq l\} = \|A^{k_0}\|^{\frac{1}{k_0}} \tag{2.142}$$

Proof. Since q_0 is a multiple of q, then $q_0/q \in \mathbb{N}$ and using (2.140) and (2.141) one gets

$$\|A^{q_0}\|^{1/q_0} = \|A^{qq_0/q}\|^{1/q_0} \leq \|A^q\|^{q_0/(qq_0)} = \|A^q\|^{1/q} \leq \|A^{k_0}\|^{1/k_0}. \tag{2.143}$$

For any integer $k \geq l+q$ we can write $k = l+i+jq$ for positive integers i and j with $0 \leq i \leq q-1$ and $j = [k-(l+i)]/q$, and then using (2.140), (2.141) and (2.143) it follows that

$$\|A^k\|^{\frac{1}{k}} \leq \left[\|A^{l+i}\|\|A^q\|^j\right]^{\frac{1}{k}} \leq \left[\|A^{k_0}\|^{\frac{l+i}{k_0}}\|A^{k_0}\|^{\frac{jq}{k_0}}\right]^{\frac{1}{k}} = \|A^{k_0}\|^{\frac{k}{k_0 k}} = \|A^{k_0}\|^{\frac{1}{k_0}}.\square \tag{2.144}$$

2.6.2. Taylor algorithm

Taylor approximation of order m of exponential of matrix $A \in \mathbb{C}^{n\times n}$ can be expressed as the matrix polynomial $T_m(A) = \sum_{k=0}^{m} A^k/k!$. The scaling and squaring algorithms with Taylor approximations are based on the approximation $e^A = \left(e^{2^{-s}A}\right)^{2^s} \approx \left(T_m(2^{-s}A)\right)^{2^s}$ [ML03], where the nonnegative integers m and s are chosen to achieve full machine accuracy at a minimum cost.

A general scaling and squaring Taylor algorithm for computing the matrix exponential is presented in Algorithm 2.4, where m_M is the maximum allowed value of m.

Algorithm 2.4 General scaling and squaring Taylor algorithm for computing $B = e^A$, where $A \in \mathbb{C}^{n\times n}$ and m_M is the maximum approximation order allowed.

1: Preprocessing of matrix A.
2: Choose $m_k \leqslant m_M$, and an adequate scaling parameter $s \in \mathbb{N} \cup \{0\}$ for the Taylor approximation with scaling.
3: Compute $B = T_{m_k}(A/2^s)$ using (2.145)
4: **for** $i = 1 : s$ **do**
5: $\quad B = B^2$
6: **end for**
7: Postprocessing of matrix B.

The preprocessing and postprocessing steps (1 and 7) are based on applying transformations to reduce the norm of matrix A, see [Hig08, AMH11], and will not be discussed in this paper.

In Step 2, the optimal order of Taylor approximation $m_k \leqslant m_M$ and the scaling parameter s are chosen. Matrix polynomial $T_m(2^s A)$ can be computed optimally in terms of matrix products using values for m in the set $m_k = \{1, 2, 4, 6, 9, 12, 16, 20, 25, 30, \ldots\}$, $k = 0, 1, \ldots$, respectively, see [Hig08, p. 72–74]. The choice of s is fully described in section 2.6.3.

After that, in Step 3, we compute the matrix exponential approximation of the scaled matrix by using the modified Horner and Paterson–Stockmeyer's method proposed in [SIDR11a, p. 1836-1837]. Note that this modified method has the same optimal values for m as the original one:

$$
\begin{aligned}
T_m\left(2^{-s}A\right) = \Bigg\{\cdots\Bigg\{\frac{A_q}{2^s m} + A_{q-1}\Bigg\}/[2^s(m-1)] + A_{q-2}\Bigg\}/[2^s(m-2)] + \cdots + A_2\Bigg\}/[2^s(m-q+2)] + A \\
+ 2^s(m-q+1)I\Bigg\}\frac{A_q}{2^{2s}(m-q+1)(m-q)} + A_{q-1}\Bigg\}/[2^s(m-q-1)] + A_{q-2}\Bigg\} \\
/\,[2^s(m-q-2)] + \cdots + A_2\Bigg\}/[2^s(m-2q+2)] + A + 2^s(m-2q+1)I\Bigg\} \\
\times \frac{A_q}{2^{2s}(m-2q+1)(m-2q)} + \cdots + A_2\Bigg\}/[2^s(q+2)] + A + 2^s(q+1)I\Bigg\} \\
\times \frac{A_q}{2^{2s}(q+1)q} + A_{q-1}\Bigg\}/[2^s(q-1)] + \cdots + A_2\Bigg\}/[2^s 2] + A\Bigg\}/2^s + I,
\end{aligned}
\tag{2.145}
$$

where matrix powers $A_i = A^i, i = 1, 2, \ldots, q$ are computed, with $q = \lceil\sqrt{m_k}\rceil$ or $\lfloor\sqrt{m_k}\rfloor$, both values dividing m_k and giving the same cost [Hig08, p. 74]. Table 2.23 shows some optimal values of q for orders m_k, $k = 0, 1, 2, \ldots, M$, with $m_M = 20, 25, 30$, denoted by q_k.

Finally, after the evaluation of $T_m\left(2^{-s}A\right)$, s repeated squarings are applied in Steps 4-6 and the postprocessing is applied in Step 7 to obtain the matrix exponential approximation of the original matrix A. The computational cost of Algorithm 2.4 in terms of matrix products is $\mathrm{Cost}(m_k, s) = k + s$.

Table 2.23: Values of q_k depending on the selection of m_M.

k	0	1	2	3	4	5	6	7	8	9
$m_M \setminus m_k$	1	2	4	6	9	12	16	20	25	30
20	1	2	2	3	3	4	4	4		
25	1	2	2	3	3	4	4	5	5	
30	1	2	2	3	3	4	4	5	5	5

2.6.3. Error analysis

Following [SIDR11a, SIDR14], denoting the remainder of the Taylor series as $R_m(A) = \sum_{k\geq m+1} A^k/k!$, for a scaled matrix $2^{-s}A$, $s \in \mathbb{N}\cup\{0\}$, we can write

$$\left(T_m(2^{-s}A)\right)^{2^s} = e^A\left(I + g_{m+1}(2^{-s}A)\right)^{2^s} = e^{A+2^s h_{m+1}(2^{-s}A)}, \tag{2.146}$$

$$g_{m+1}(2^{-s}A) = -e^{-2^{-s}A}R_m(2^{-s}A),\ h_{m+1}\left(2^{-s}A\right) = \log\left(I+g_{m+1}(2^{-s}A)\right), \tag{2.147}$$

where log denotes the principal logarithm, $h_{m+1}(X)$ is defined in the set $\Omega_m = \left\{X \in \mathbb{C}^{n\times n} : \rho\left(e^{-X}T_m(X) - I\right) < 1\right\}$, and both $g_{m+1}(2^{-s}A)$ and $h_{m+1}\left(2^{-s}A\right)$ are holomorphic functions of A in Ω_m and then commute with A. As showed in [SIDR11a], $h_{m+1}(2^{-s}A)$ and $g_{m+1}(2^{-s}A)$ are related with the backward and forward errors in exact arithmetic from the approximation of e^A by the Taylor series with scaling and squaring, respectively. Choosing s so that

$$\left\|h_{m+1}\left(2^{-s}A\right)\right\| \leq \max\left\{1, \left\|2^{-s}A\right\|\right\} u, \tag{2.148}$$

where $u = 2^{-53}$ is the unit roundoff in IEEE double precision arithmetic, then: if $2^{-s}\|A\| \geq 1$, then the backward error $\Delta A \leq \|A\|\, u$ and using (2.146) one gets $\left(T_m\left(2^{-s}A\right)\right)^{2^s} = e^{A+\Delta A} \approx e^A$, and if $2^{-s}\|A\| < 1$, using (2.146)-(2.148) and the Taylor series one gets

$$\left\|R_m\left(2^{-s}A\right)\right\| \approx \left\|T_m\left(2^{-s}A\right)\right\| u. \tag{2.149}$$

Hence, as Algorithm 2.4 evaluates explicitly $T_m\left(2^{-s}A\right)$, by (2.149) one gets $e^{2^{-s}A} = T_m\left(2^{-s}A\right) + R_m\left(2^{-s}A\right) \approx T_m\left(2^{-s}A\right)$, and there is no need to increase m or the scaling parameter s to try to get a higher accuracy. Using scalar Taylor series in (2.147) one gets

$$g_{m+1}(x) = \sum_{k\geq m+1} b_k^{(m)} x^k, \quad h_{m+1}(x) = \sum_{k\geq 1} \frac{(-1)^{k+1}(g_{m+1}(x))^k}{k} = \sum_{k\geq m+1} c_k^{(m)} x^k, \tag{2.150}$$

where $b_k^{(m)}$ and $c_k^{(m)}$ depend on the order m. Moreover, $b_k^{(m)} = c_k^{(m)}, k = m+1, m+2, \ldots, 2m+1$ and if $\|h_{m+1}(2^{-s}A)\| \ll 1$ or if $\|g_{m+1}(2^{-s}A)\| \ll 1$, then $h_{m+1}(2^{-s}A) \approx g_{m+1}(2^{-s}A)$, see [SIDR14]. Using MATLAB symbolic Math Toolbox, high precision arithmetic, 200 series terms and a zero finder we obtained the maximal values Θ_m of $\Theta = \|2^{-s}A\|$, shown in Table 2.24, such that, using the notation of Theorem 5

$$\left\|h_{m+1}\left(2^{-s}A\right)\right\| \leq \tilde{h}_{m+1}(\Theta) = \sum_{k\geq m+1} |c_k^{(m)}|\Theta^k \leq \max\{1,\Theta\}\, u. \tag{2.151}$$

Table 2.24: Maximal values $\Theta_m = \|2^{-s}A\|$ such that $\tilde{h}_{m+1}(\Theta_m) \le \text{máx}\{1, \Theta_m\}u$, coefficient ratios $c_{m+1}^{(m)}/c_{m+2}^{(m)}$, values $u/|c_{m+2}^{(m)}|$, maximal values $\hat{\Theta}_m$ using only the first 2 terms in series $\tilde{h}_{m+1}(\hat{\Theta}_m)$, and values $\tilde{h}_{m+1}(\hat{\Theta}_m)/(\text{máx}\{1, \hat{\Theta}_m\}u)$ considering 200 series terms.

m	Θ_m	$c_{m+1}^{(m)}/c_{m+2}^{(m)}$	$u/\lvert c_{m+2}^{(m)}\rvert$	$\hat{\Theta}_m$	$\frac{\tilde{h}_{m+1}(\hat{\Theta}_m)}{\text{máx}\{1,\hat{\Theta}_m\}u}$
1	1.490116111983279e-8	-3/2	3.3e-16	1.490e-8	1.00
2	8.733457513635361e-6	-4/3	8.9e-16	8.733e-6	1.00
4	1.678018844321752e-3	-6/5	1.6e-14	1.678e-3	1.00
6	1.773082199654024e-2	-8/7	6.4e-13	1.777e-2	1.02
9	1.137689245787824e-1	-11/10	4.4e-10	1.150e-1	1.11
12	3.280542018037257e-1	-14/13	7.5e-7	3.358e-1	1.37
16	7.912740176600240e-1	-18/17	4.2e-2	8.269e-1	2.19
20	1.438252596804337	-22/21	5.9e03	1.474	1.70
25	2.428582524442827	-27/26	4.7e10	2.538	3.36
30	3.539666348743690	-32/31	9.4e17	3.771	8.34

Hence, if $\|2^{-s}A\| \le \Theta_m$ then (2.148) holds. For the cases where $\Theta_m > 1$, note that $f(\Theta) = \tilde{h}_{m+1}(\Theta) - \Theta u$ is a continuous function in $[0, \Theta_m]$ and $f(\Theta_m) = 0$, $f(0) = 0$. For $m = 20, 25, 30$ we have checked that there are no other zeros in $[0, \Theta_m]$, and $f(\Theta) < 0$, $\Theta \in]0, \Theta_m[$. Thus, for those orders the next bound holds

$$\|h_{m+1}(2^{-s}A)\| \le \tilde{h}_{m+1}(\|2^{-s}A\|) = \tilde{h}_{m+1}(\Theta) \le \Theta u, \quad 0 \le \Theta \le \Theta_m. \tag{2.152}$$

2.6.4. New Taylor algorithm

2.6.4.1. New scaling algorithm

In this section a new scaling algorithm is proposed, being a simplification of that presented in [SIDR11a, p. 1837-1838]. For all norms appearing in the scaling algorithm we will use the 1-norm, and m_M will be the maximum allowed Taylor order. Using the bounds and a similar process that we describe below, we will first check if any of the Taylor optimal orders $m_k = 1, 2, 4, \ldots, m_{M-1}$ satisfy (2.148) without scaling, i.e. with $s = 0$. If not, we will calculate the optimal scaling s for order m_M in two phases: first, we will calculate an initial value of the scaling parameter, s_0, and then we will try to refine it, testing if it can be reduced. In this paper we have simplified both phases with respect to the algorithms from [SIDR11a] and [SIDR14], avoiding costly and complex checks that rarely allow to reduce the scaling parameter.

We begin estimating the 1–norm of $\|A^{m+1}\|$ using the block 1–norm estimation algorithm of Higham and Tisseur [HT00]. For a $n \times n$ matrix this algorithm carries out a 1-norm power iteration whose iterates are $n \times t$ matrices, where t is a parameter that has been taken to be 2, see [AMH09, p. 983]. Hence, the estimation algorithm has $O(n^2)$ computational cost, negligible compared to the cost of a matrix product, i.e. $O(n^3)$.

In [SIDR11a, p. 1837], the upper bounds a_k for $\|A^k\|$ needed to apply Theorem 5 in (2.151) were obtained using products of norms of matrix powers estimated for current and previous tested orders, i.e. $\|A^{m_k+1}\|$, $k = 0, 1, 2, \ldots, M$, and the powers of A computed for evaluation of $T_{m_M}(2^{-s}A)$, A^i, $i = 1, 2, \ldots, q$, as

$$\begin{aligned}\left\|A^k\right\| \le a_k \;=\; \min\Big\{ & \|A\|^{i_1} \left\|A^2\right\|^{i_2} \cdots \|A^q\|^{i_q} \left\|A^{m_1+1}\right\|^{i_{m_1+1}} \left\|A^{m_2+1}\right\|^{i_{m_2+1}} \cdots \\ & \times \left\|A^{m_M+1}\right\|^{i_{m_M+1}} : i_1 + 2i_2 + \cdots + qi_q + (m_1+1)i_{m_1+1} \\ & + (m_2+1)i_{m_2+1} + \cdots + (m_M+1)i_{m_M+1} = k\Big\}, \end{aligned} \tag{2.153}$$

where the minimum was desirable, but not necessary. Then, in [SIDR11a], α_p values from Theorem 5 with $l = m_M + 1$, see (2.139), were obtained successively for $p = 2, 3, \ldots, q, m_1 + 1, m_2 + 1, \ldots, m_M + 1$, stopping the process when $(a_p)^{1/p} \le \max\{(a_k)^{1/k} : k = m+1, m+2, \ldots, m+p\}$, since if this condition holds, then for $p' > p$ it follows that $\alpha_{p'} \ge \alpha_p$, see [SIDR14, p. 105] for a demonstration. Then, the minimum value among all values α_p, denoted by α_{min}, was selected.Finally, the initial minimum scaling parameter $s_0 \ge 0$ so that $2^{-s_0}\alpha_{min} \le \Theta_{m_M}$ was computed as follows: if $\alpha_{min} \le \Theta_{m_M}$ then $s_0 = 0$, and otherwise $s_0 = \lceil \log_2(\alpha_{min}/\Theta_{m_M}) \rceil$. In [SIDR11a] it was also shown that taking $s = s_0$ then (2.148) holds.

In this work, we have simplified all that process by directly approximating

$$\alpha_{min} \approx \max\{a_{m+1}^{1/(m+1)}, a_{m+2}^{1/(m+2)}\}, \tag{2.154}$$

where a_{m+1} and a_{m+2} are the 1–norm estimation of $\|A^{m+1}\|$ and $\|A^{m+2}\|$, respectively, using the block 1–norm estimation algorithm of Higham and Tisseur [HT00]. Theorem 6 establishes that $\max\{\|A^k\|^{\frac{1}{k}} : k \ge l\}$ is obtained for $l \le k \le l + q - 1$, where $1 \le q \le l$. If we use maximum Taylor order $m_M = 30$ for the computation of a matrix exponential then from Theorem 5 and 6 it follows that $l = m_M + 1 = 31$ and $l + q - 1 \le 61$. For normal matrices, since $\|A^k\|_2 = \|A\|_2^k$, then $\max\{\|A^k\|_2^{\frac{1}{k}} : k \ge l\} = \max\{\|A\|_2^{\frac{k}{k}} : k \ge l\} = \|A\|_2$ and (2.154) will be a good approximation (it would be exact if we used the 2-norm and the exact values of $\|A^{m+1}\|_2$ and $\|A^{m+2}\|_2$). For the case of nonnormal matrices Figure 2.9 shows the values $\{\|A^k\|_2^{1/k}\}_{k=1}^{61}$ for 103 matrices A from the matrix exponential literature with $\|A\|_2 = 1$ and sizes from 2×2 to 100×100, see [AMH09, p. 973]. Since $\|A^k\|^{1/k} \to \rho(A)$ as $k \to \infty$ [AMH09, p. 972], in the majority of matrices the values $\|A^k\|^{1/k}$ tend to be decreasing and tend to have less variations for higher matrix powers, so (2.154) will be a good approximation for $\max\{\|A^k\|^{\frac{1}{k}} : k \ge m_M + 1\}$. In fact, for those nonnormal matrices with monotonically decreasing sequence $\|A^k\|^{\frac{1}{k}}$, $k = 1, 2, \ldots$ it follows that $\max\{\|A^k\|^{\frac{1}{k}} : k \ge m_M + 1\} = \|A^{m_M+1}\|^{\frac{1}{m_M+1}}$ and then the value α_{min} from (2.154) gives the best possible selection. The fact of taking at least two matrix powers in (2.154) comes from the example matrix (3.8) from [AMH11]

$$A = \begin{bmatrix} 1 & a \\ 0 & -1 \end{bmatrix}, \; |a| >> 1, \|A^{2k}\|_1 = 1, \|A^{2k+1}\|_1 = 1 + |a|, \tag{2.155}$$

where the norm of the odd powers is much greater than the norm of the even powers. Note that in this case

$$\max\{\|A^k\|^{\frac{1}{k}} : k \ge m + 1\} = \max\{\|A^{m_M+1}\|^{\frac{1}{m_M+1}}, \|A^{m_M+2}\|^{\frac{1}{m_M+2}}\}, \tag{2.156}$$

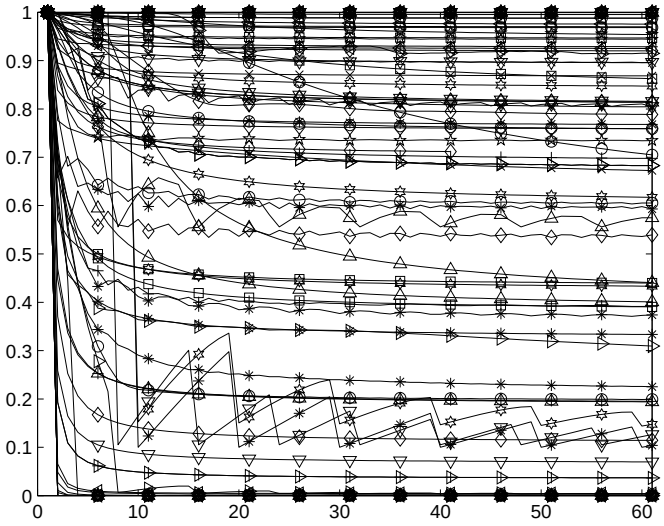


Figure 2.9: $\{\|A^k\|_2^{1/k}\}_{k=1}^{61}$ for 103 matrices A with $\|A\|_2 = 1$ and sizes from 2×2 to 100×100.

and it is necessary to check at least two matrix powers to obtain the maximum. Numerical tests confirmed that using only two terms in (2.154) made no noticeable difference in the accuracy results in the majority of test matrices.

Once obtained s_0, if $s_0 \geq 1$ we check if (2.148) holds reducing the scaling $s = s_0 - 1$, and using the bounds for $\left\|A^k\right\| \leq a_k$ to test if bound

$$\frac{\|h_{m+1}\left(2^{-s}A\right)\|}{|c_{m+2}^{(m)}|} \leq \sum_{k=m+1}^{m+2} \left|\frac{c_k^{(m)}}{c_{m+2}^{(m)}}\right| \frac{a_k}{2^{sk}} \leq \max\left\{1, \|2^{-s}A\|\right\} \frac{u}{|c_{m+2}^{(m)}|}, \tag{2.157}$$

holds, truncating the series. Note that we will stop the series summation if after summing the first term, the sum is greater than $\max\{1, \|2^{-s}A\|\}u/|c_{m+2}^{(m)}|$. Table 2.24 presents some values of $c_k^{(m)}/c_{m+2}^{(m)}$, and the values $u/|c_{m+2}^{(m)}|$. In (2.157) we have simplified the process of determining the error bound with respect to those in [SIDR11a] and [SIDR14], using only the two first terms of the error series, instead of the q terms used in algorithms from [SIDR11a] and [SIDR14]. Table 2.24 shows the maximal values $\hat{\Theta}_m$ such that $\tilde{h}_{m+1}(\hat{\Theta}_m) \leq \max\{1, \hat{\Theta}_m\}u$ using only the first 2 terms in series $\tilde{h}_{m+1}(\hat{\Theta}_m)$, and the values $\tilde{h}_{m+1}(\hat{\Theta}_m)/(\max\{1, \hat{\Theta}_m\}u)$ considering then 200 series terms. These values show that for normal matrices, considering only two series terms makes little relative difference, (≤ 8.34) in the worst case, with respect to considering all the series terms.

For nonnormal matrices the value of the remaining series terms will depend on the values $\|A^k\|^{1/k}$, $k \geq m+3$ and their ratio with $\max\{\|A^{m+1}\|^{\frac{1}{m+1}}, \|A^{m+2}\|^{\frac{1}{m+2}}\}$. Given the results from Figure 2.9, typically the terms of the power series of function $\|h_{m+1}\left(2^{-s}A\right)\|$ will be decreasing, so the first terms tend to determine the error bound.

Taking all the previous into account we decided to reduce the number of terms of the error series to be used and we checked empirically that using more terms rarely modified the final result. Again, taking into account matrix (2.155) we use the first two terms of the power series of $\|h_{m+1}(2^{-s}A)\|$.

In this step, we have also removed a complex and costly test that previous versions of the new Taylor algorithm do when expression (2.157) does not hold with $s = s_0 - 1$, see (15) from [SIDR11a] and (45) from [SIDR14]. We have found empirically that when (2.157) does not hold, then very rarely those tests are satisfied.

In the next subsection, a detailed description of the new algorithm is given.

2.6.4.2. Taylor algorithm

The new proposed algorithm is presented in Algorithm 2.5. For all norms appearing in the scaling algorithm we use the 1-norm. The maximum allowed Taylor order, m_M, is an input parameter. In our experience, the optimal values for m_M are $20, 25$ or 30, increasing slightly the cost in tests with increasing m_M but also improving the accuracy, see Fig. 1 and Table 2 of [SIDR11a]. For clarity in the algorithm, we have considered that the maximum value for m_M is 30. However, extending the algorithm to higher values for m_M is straightforward.

In Steps 3–27, Algorithm 2.5 checks if any of the Taylor optimal orders $m_k = 1, 2, 4, \ldots, m_M$ satisfies (2.148) without scaling ($s = 0$), using the bounds provided in the previous section. As mentioned above, we compute the 1–norm estimate of $\|A^{m+1}\|$ and $\|A^{m+2}\|$ using the block 1–norm estimation algorithm of [HT00].

If no value of $m_k \leq m_M$ satisfies (2.148), the algorithm computes α_{min} using only the 1-norm estimate of the matrix powers $\|A^{m+1}\|$ and $\|A^{m+2}\|$, and determines the initial scaling parameter s_0 in Steps 29–30. Then, if $s_0 > 0$, the algorithm checks in Steps 31–37 if the initial scaling parameter can be reduced, testing if (2.148) holds with $s = s_0 - 1$.

Then, in Steps 38–42, similarly to the algorithms proposed in [SIDR11a] and [SIDR14], Algorithm 2.5 tests if (2.148) holds with s and m_{M-1}; Taylor order $m = m_{M-1}$ will be used if (2.148) holds, or $m = m_M$ otherwise.

Finally, in Step 43 we use (2.145) to compute the exponential approximation of the scaled matrix, and in Steps 44–46 s squaring steps are done to obtain the matrix exponential approximation of the original matrix A.

Algorithm 2.5 Given a matrix $A \in \mathbb{C}^{n\times n}$ and a maximum order m_M, this algorithm computes $B = e^A$ by a Taylor approximation of order $m \leqslant m_M$.

Inputs: $A \in \mathbb{C}^{n\times n}$ and maximun order approximation m_M
Output: $B = e^A$

1: Set Θ_m, $c_{m+1}^{(m)}/c_{m+2}^{(m)}$ and $u/|c_{m+2}^{(m)}|$ values from Table 2.24
2: $s \leftarrow 0$

Algorithm 2.5 (continued)

if $\|A\| < \Theta_1$ **then**
  $m \leftarrow 1$
  $B \leftarrow A + I_n$
  quit
end if
for $m \in [2, 4, 6, \ldots, m_M]$ **do** ▷ Optimal values for m, from 2 to m_M
  if $m = 2$ **then**
    Compute and save A^2
  else if $m = 6$ **then**
    Compute and save A^3
  else if $m = 12$ **then**
    Compute and save A^4
  else if $m = 20$ **then**
    Compute and save A^5
  end if
  $b \leftarrow \max\{1, \|A\|\} \cdot u / |c^{(m)}_{m+2}|$
  $a(m+1) \leftarrow \|A^{m+1}\|$ ▷ Estimate value of $\|A^{m+1}\|$
  if $|c^{(m)}_{m+1}/c^{(m)}_{m+2}| \cdot a(m+1) \leq b$ **then**
    $a(m+2) \leftarrow \|A^{m+2}\|$ ▷ Estimate value of $\|A^{m+2}\|$
    if $|c^{(m)}_{m+1}/c^{(m)}_{m+2}| \cdot a(m+1) + a(m+2) \leq b$ **then**
      $B \leftarrow T_m(A)$ ▷ Evaluate $T_m(A)$ using (2.145) with $s = 0$
      quit
    end if
  end if
end for
$m \leftarrow m_M$ ▷ Maximum order selected
$\alpha_{min} \leftarrow \max\{a(m+1)^{1/(m+1)}, a(m+2)^{1/(m+2)}\}$
$s_0 \leftarrow \lceil \log_2(\alpha_{min}/\Theta_m) \rceil$
if $s_0 > 0$ **then** ▷ Check if (2.148) holds reducing the scaling $s = s_0 - 1$
  $s \leftarrow s_0 - 1$
  $b \leftarrow \max\{1, \|A\|/2^s)\} u / |c^{(m)}_{m+2}|$
  if $|c^{(m)}_{m+1}/c^{(m)}_{m+2}| \cdot a(m+1)/2^{(m+1)s} + a(m+2)/2^{(m+2)s} > b$ **then**
    $s \leftarrow s_0$ ▷ (2.148) does not hold, then $s = s_0$
  end if
end if
$m \leftarrow m_{M-1}$ ▷ Test if scaled matrix allows using m_{M-1}
$b \leftarrow \max\{1, \|A\|/2^s\} \cdot u / |c^{(m)}_{m+2}|$
if $|c^{(m)}_{m+1}/c^{(m)}_{m+2}| \cdot a(m+1)/2^{(m+1)s} + a(m+2)/2^{(m+2)s} > b$ **then**
  $m \leftarrow m_M$ ▷ Scaled matrix does not allow using m_{M-1}, then $m = m_M$
end if
Compute $B = T_m(A/2^s)$ ▷ Evaluate $T_m(2^{-s}A)$ using (2.145)
for $i = 1 : s$ **do** ▷ Squaring phase
  $B = B^2$
end for

2.6.5. Numerical experiments and conclusions

In this section we compare a Matlab implementation of the new algorithm, denoted by **exptaynsv3**, with the functions **exptayns** and **exptaynsv2** from [SIDR11a] and [SIDR14], respectively, and also a Fortran version of **exptaynsv3** with one of the main commercial software available for computing matrix exponentials: the NAG library [NAG02]. The four functions are available at:

http://personales.upv.es/jorsasma/Software/exptayns.m

http://personales.upv.es/jorsasma/Software/exptaynsv2.m

http://personales.upv.es/jorsasma/Software/exptaynsv3.m

http://personales.upv.es/jorsasma/Software/exptaynsv3fortran.zip

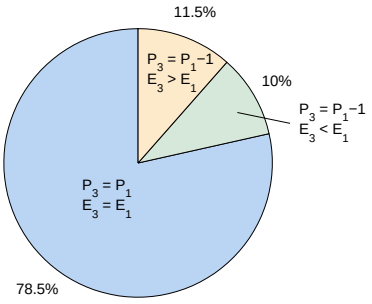
We have also included a comparison of **exptaynsv3** with MATLAB function **expm_new** from [AMH09], that implements a scaling squaring Padé algorithm to compute matrix exponential. The accuracy was tested by computing the relative error $\mathrm{E} = \|e^A - \tilde{X}\|_1 / \left\|e^A\right\|_1$, where $\tilde{X}$ is the computed approximation and the cost is given in terms of matrix products. We used the following sets of matrices for testing:

1. One hundred diagonalizable matrices of size 1024. These matrices have the form $V^T D V$, where D is a diagonal matrix whose diagonal elements are random values between $-k$ and k with different integer values of k, and V is an orthogonal matrix obtained as $V = H/16$, where H is the Hadamard matrix.
2. One hundred matrices with multiple eigenvalues of size 1000. These matrices have the form $V^T D V$, where D is a block diagonal matrix whose diagonal blocks are Jordan blocks with random dimension and random eigenvalues between -50 and 50, and V is an invertible matrix with random values in $[-0.5, 0.5]$.
3. 32 matrices 1000×1000 from the function `matrix` from the Matrix Computation Toolbox [Hig02b]. Matrices whose exponential cannot be represented in double precision due to overflow were excluded from all the matrices given by function `matrix`. These matrices appear in the state of the art in the exponential matrix computation [AMH09, Hig05].

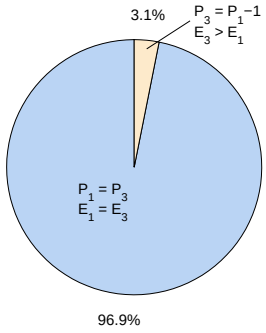
The "exact" value of matrix exponential for matrix sets 1 and 2 was computed by using transformations $e^A = V^T e^D V$, where $V^T e^D V$ was computed using `vpa` function from Matlab's Symbolic Math Toolbox with 32 decimal digit precision. For matrix set 3, we used quadruple precision Taylor algorithm in Fortran with different orders and scaling parameters for each matrix to check the result correctness. The maximum order used for Taylor approximation in all cases was $m_M = 30$.

Figure 2.10 presents the comparison of functions `exptaynsv3` and `exptayns` in terms of matrix products and relative errors. The new algorithm saved one matrix product in 21.5 % of cases from the matrix sets 1 and 2, and one matrix product in 3.1 % of cases from the matrix set 3. Both algorithms achieved very similar accuracy results, with $\max(|E_1 - E_3|/|E_1|) = 0.0194$ for matrix sets 1 and 2, and $\max(|E_1 - E_3|/|E_1|) = 0.0038$ for matrix set 3.

Similarly, Figure 2.11 shows the results of the comparison between the functions `exptaynsv3` and `exptaynsv2`. In this case considering matrix sets 1 and 2, the new algorithm saved one matrix product in 18.5 % of cases. With respect to matrix set 3, it saved one matrix product in 3.1 % of cases, whereas it performed one more matrix product in 9.4 % of cases. In terms of accuracy, see Figure 2.12, results are very similar to those obtained in the previous case when considering matrix sets 1 and 2, obtaining $\max(|E_2 - E_3|/|E_2|) = 0.0194$. Considering matrix set 3, see Figure 2.12b, the only case where the error difference is significant is for matrix 7, where `exptaynsv2` gives a 10^{-15} order accuracy, while `exptaynsv3` obtains a higher accuracy of order 10^{-20}. For the remaining cases $\max(|E_2 - E_3|/|E_2|) = 0.41$.

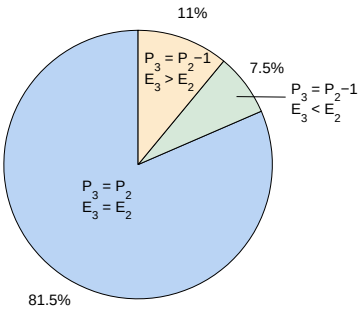


(a)

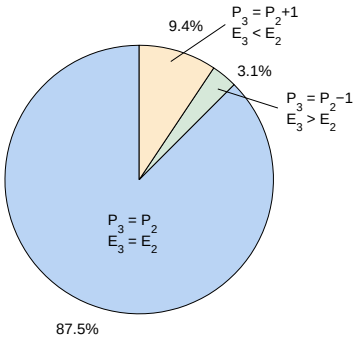


(b)

Figure 2.10: Comparison of the cost in terms of matrix products (P) and the relative error (E) between `exptaynsv3` (P_3 and E_3) and `exptayns` (P_1 and E_1) with matrix sets 1 and 2 (a), and matrix set 3 (b).



(a)



(b)

Figure 2.11: Comparison of the cost in terms of matrix products (P) and the relative error (E) between `exptaynsv3` (P_3 and E_3) and `exptaynsv2` (P_1 and E_1) with matrix sets 1 and 2 (a), and matrix set 3 (b).

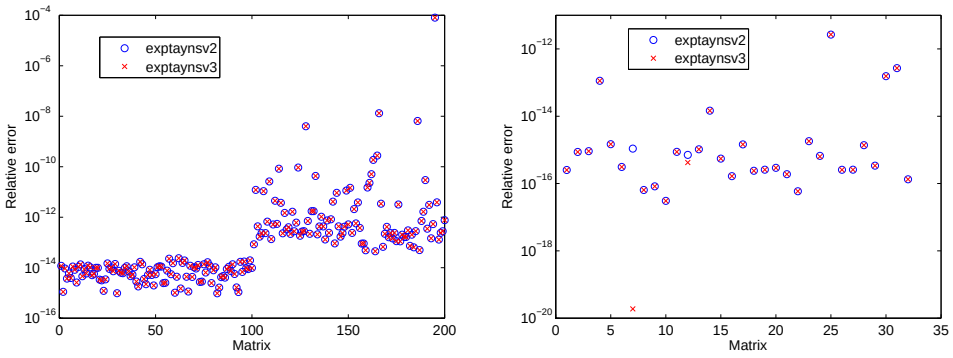


Figure 2.12: Relative error comparison between `exptaynsv3` and `exptaynsv2` with matrix sets 1 and 2 (a) and matrix set 3 (b). Taylor maximum order $m_M = 30$.

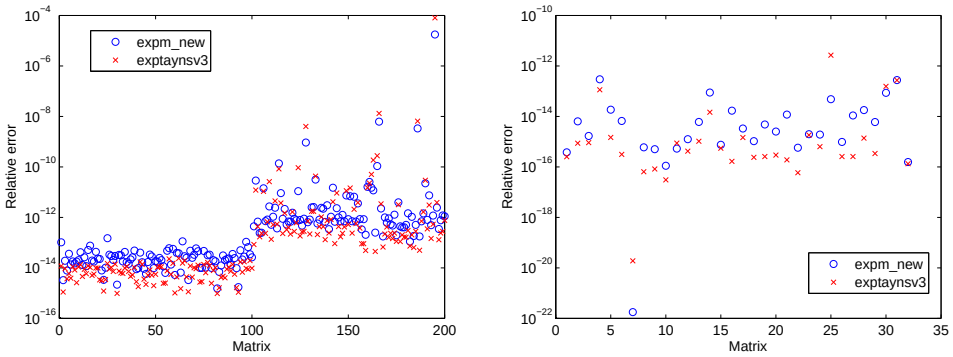


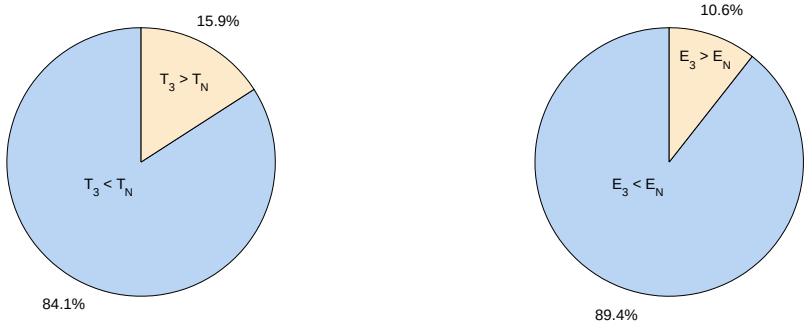
Figure 2.13: Relative error comparison between `exptaynsv3` and `expm_new` with matrix sets 1 and 2 (a) and matrix set 3 (b). Taylor maximum order $m_M = 30$.

The accuracy comparison between the functions `exptaynsv3` and `expm_new` is presented in Figure 2.13. Function `exptaynsv3` achieved a higher accuracy in 88.5 % of the matrices from matrix sets 1 and 2, and in 87.5 % of the matrices from matrix set 3. The number of matrix products performed by `exptaynsv3` to compute all the matrix exponentials of matrix sets 1, 2 and 3 were 1115, 700 and 351, respectively, whilst `expm_new` carried out 1338, 1596 and 345.6 matrix products, respectively.

Although matrix set 3 includes some ill-conditioned matrices, we found interesting to check the behaviour of the functions with a very nonnormal and ill-conditioned matrix, such as the one generated by the command `gallery('triw',20,4.1)` of Matlab [AMH11]. We computed the "exact" exponential of this matrix using `vpa` Matlab's function with 500 decimal digit precision and the relative error obtained was $1.8163 \cdot 10^{-16}$ for the three Taylor functions and $1.1408 \cdot 10^{-15}$ for `expm_new`. The number of matrix products performed were 10 for the Taylor functions and 7.3 for `expm_new`.

Table 2.25: Execution time comparison in seconds between the functions `exptayns`, `exptaynsv2` and `exptaynsv3` in Matlab.

	`exptayns`	`exptaynsv2`	`exptaynsv3`
Matrix set 1	262.83	266.94	227.69
Matrix set 2	178.25	179.54	136.37
Matrix set 3	171.93	170.81	165.47



(a) (b)

Figure 2.14: Cost in seconds (T) (a) and relative error comparison (E) (b) between `exptaynsv3` (T_3 and E_3) and `f01ecc` from NAG Library (T_N and E_N) with matrix sets 2 and 3.

We have also included an execution time comparative of the three Taylor functions in Matlab. Although execution time in Matlab is not always reliable, since the three functions are similar the results can be useful. Table 2.25 shows the total time in seconds taken by each function to compute all matrix exponentials of each matrix set. As shown, `exptaynsv3` computes the exponential faster than the other two previous versions in all cases.

Finally, we have compared a Fortran version of the new algorithm with one of the main commercial software packages that allows the computation of matrix exponentials. The function `f01ecc` from NAG Library, see [NAG02], computes the matrix exponential of a real square matrix, using the algorithm based on Padé approximants and the scaling and squaring method described in [Hig05] and [Hig08].

The tests have been done in a Linux system with an Intel processor, using the Math Kernel Libraries from Intel. We have used the 32 matrices from Matlab Toolbox and the 100 random Jordan matrices from previous tests. Results are shown in Figures 2.14 and 2.15. In this case, execution time instead of matrix products was used to evaluate the cost of both functions, and function `exptaynsv3` obtained better results than the NAG routine in both accuracy, 89.4 % of cases, and execution time, 84.1 % of cases. The total time taken by `exptaynsv3` to compute all matrix exponentials was 521 seconds, versus 866 seconds taken by `f01ecc` function. Moreover, `exptaynsv3` was significantly more accurate in the majority of cases, see Figure 2.15.

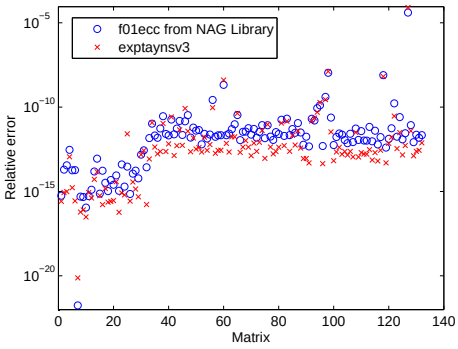


Figure 2.15: Relative error comparison between `exptaynsv3` and function `f01ecc` from NAG Software.

2.6.6. Conclusions

A competitive modification of the Taylor algorithm from [SIDR11a] has been proposed based on a simplification of the calculation of the error bounds used to select the order of the approximation and the scaling parameter. These modifications were based on theoretical results for normal matrices and empirical results for nonnormal matrices, leading to a new simplified algorithm that obtained similar accuracy as previous algorithms from the authors in the majority of test matrices with a lower processing time, and also better results than state-of-the-art algorithms based on Padé approximations.

Bibliografía

[ACF96] M. Arioli, B. Codenotti, and C. Fassino. The Padé method for computing the matrix exponential. *Linear Algebra Appl.*, 240:111–130, 1996.

[AMH09] A. H. Al-Mohy and N. J. Higham. A new scaling and squaring algorithm for the matrix exponential. *SIAM J. Matrix Anal. Appl.*, 31(3):970–989, 2009.

[AMH11] A. H. Al-Mohy and N. J. Higham. Computing the action of the matrix exponential, with an application to exponential integrators. *SIAM J. Sci. Comput.*, 33(2):488–511, 2011.

[BD99] S. Blackford and J. Dongarra. Installation guide for LAPACK, LAPACK Working Note 411. Technical report, Department of Computer Science University of Tennessee, Knoxville, TN, 1999.

[CM02] S. M. Cox and P. C. Matthews. Exponential time differencing for stiff systemsy. *J. Comput. Phys.*, 176:430–455, 2002.

[DH03] P. I. Davies and N. J. Higham. A Schur–Parlett algorithm for computing matrix functions. *SIAM J. Matrix Anal. Appl.*, 25(2):464–485, 2003.

[DJ98] E. Defez and L. Jódar. Some applications of Hermite matrix polynomials series expansions. *Journal of Computational and Applied Mathematics*, 99:105–117, 1998.

[DM02] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Programming*, 91:201–213, 2002.

[DP00] L. Dieci and A. Papini. Padé approximation for the exponential of a block triangular matrix. *Linear Algebra Appl.*, 308:183–202, 2000.

[DP01] L. Dieci and A. Papini. Conditioning of the exponential of a block triangular matrix. *Numerical Algorithms*, 28:137–150, 2001.

[DS57] N. Dunford and J. Schwartz. *Linear Operators, Part I: General Theory.* Interscience, New York, 1957.

[Fas93] C. Fassino. *Computation of Matrix Functions.* PhD thesis, Università di Pisa, Genova, 7 1993.

[FDC46] R. A. Frazer, W. J. Duncan, and A. R. Collar. *Elementary Matrix and Some Applications to Dynamics and Differential Equations.* Macmillan, New York, 1946.

[FOR] Fortran versions of functions TSTD, TPS, OTPS, TPSBT, OTPSBT and **expm**. http://personales.upv.es/~jorsasma/FORTRAN.zip.

[Fun04] T. C. Fung. Computation of the matrix exponential and its derivatives by scaling and squaring. *International Journal for Numerical Methods in Engineering*, 59:1273–1286, 2004.

[GL96] G. H. Golub and C. Van Loan. *Matrix Computations.* Johns Hopkins Studies in Mathematical Sciences. The Johns Hopkins University Press, third edition, 1996.

[HAM10] N. J. Higham and A. H. Al-Mohy. Computing matrix functions. *Acta Numerica*, 19:159–208, 2010.

[Hig93] N. J. Higham. The Test Matrix Toolbox for MATLAB. Numerical Analysis Report No. 237, Manchester Centre for Computational Mathematics, Manchester Centre for Computational Mathematics, Manchester, England, December 1993.

[Hig02a] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA, USA, second edition, 2002.

[Hig02b] N. J. Higham. The Matrix Computation Toolbox. http://www.ma.man.ac.uk/~higham/mctoolbox, 2002.

[Hig05] N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl.*, 26(4):1179–1193, 2005.

[Hig08] N. J. Higham. *Functions of Matrices: Theory and Computation*. SIAM, Philadelphia, PA, USA, 2008.

[HLS98] M. Hochbruck, C. Lubich, and H. Selhofer. Exponential integrators for large systems of differential equations. *SIAM J. Sci. Comput.*, 19(5):1552–1574, 1998.

[HT00] N. J. Higham and F. Tisseur. A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra. *SIAM J. Matrix Anal. Appl.*, 21:1185–1201, 2000.

[IHAR09] J. Ibáñez, V. Hernández, E. Arias, and P. Ruiz. Solving initial value problems for ordinary differential equations by two approaches: BDF and piecewise-linearized methods. *Computer Physics Communications*, 180(5):712–723, 2009.

[JC96] L. Jódar and R. Company. Hermite matrix polynomials and second order matrix differential equations. *Journal Approximation Theory Application*, 12(2):20–30, 1996.

[KL98] C. S. Kenney and A. J. Laub. A Schur–Fréchet algorithm for computing the logarithm and exponential of a matrix. *SIAM J. Matrix Anal. Appl.*, 19(3):640–663, 1998.

[KT05] A. K. Kassam and Ll. N. Trefethen. Fourth-order time-stepping for stiff PDEs. *SIAM J. Sci. Comput.*, 26(4):1214–1233, 2005.

[Lu03] Y. Y. Lu. Computing a matrix function for exponential integrators. *J. Comput. Appl. Math.*, 161:203–216, 2003.

[ML78] C. B. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review*, 20(4):801–836, 1978.

[ML03] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*. *SIAM Review*, 45:3–49, 2003.

[NAG02] NAG Library Function Document. http://www.nag.co.uk/numeric/cl/nagdoc_cl23/html/F01/f01ecc.html, 2002.

[NH95] I. Najfeld and T. F. Havel. Derivatives of the matrix exponential and their computation. *Advances in Appl. Math.*, 16:321–375, 1995.

[PCK91] P. Hr. Petkov, N. D. Christov, and M. M. Konstantinov. *Computational methods for linear control systems*. Prentice Hall International Ltd., Hertfordshire, UK, 1991.

[PS73] M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, 2(1):60–66, 1973.

[RSID16] P. Ruiz, J. Sastre, J. Ibáñez, and E. Defez. High performance computing of the matrix exponential. *Journal of Computational and Applied Mathematics*, 291:370 – 379, 2016. Mathematical Modeling and Computational Methods.

[SIDR09] J. Sastre, J. Ibáñez, E. Defez, and P. Ruiz. Efficient scaling-squaring taylor method for computing the matrix exponential. http://personales.upv.es/~jorsasma/076320.pdf, 2009.

[SIDR11a] J. Sastre, J. Ibáñez, E. Defez, and P. Ruiz. Accurate matrix exponential computation to solve coupled differential models in engineering. *Math. Comput. Model.*, 54:1835–1840, 2011.

[SIDR11b] J. Sastre, J. Ibáñez, E. Defez, and P. Ruiz. Efficient orthogonal matrix polynomial based method for computing matrix exponential. *Appl. Math. Comput.*, 217(14):6451–6463, 2011.

[SIDR14] J. Sastre, J. Ibáñez, E. Defez, and P. Ruiz. Accurate and efficient matrix exponential computation. *Int. J. Comput. Math.*, 91(1):97–112, 2014.

[SIDR15] J. Sastre, J. Ibáñez, E. Defez, and P. Ruiz. New scaling-squaring Taylor algorithms for computing the matrix exponential. *SIAM J. Sci. Comput.*, 37(1):A439–A455, 2015.

[Smi85] G. D. Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford University Press, third edition, 1985.

[SZ71] S. Saks and A. Zygmund. *Analytic Functions*. Elsevier Science Publishers, Amsterdam, The Netherlands, 1971.

[War77] R. C. Ward. Numerical computation of the matrix exponential with accuracy estimate. *SIAM J. Numer. Anal.*, 14(4):600–610, 1977.

[Wes90] D. Westreich. A practical method for computing the exponential of a matrix and its integral. *Communications in Applied Numerical Methods*, 6:375–380, 1990.

[Wri02] T. G. Wright. Eigtool. http://www.comlab.ox.ac.uk/pseudospectra/eigtool/, 2002.

Capítulo 3

Coseno y seno de una matriz

3.1. Introducción

Este capítulo está dedicado al cálculo de las funciones seno y coseno matriciales. El trabajo realizado durante el desarrollo de esta tesis ha dado lugar a los siguientes tres artículos relacionados con estas funciones:

- *"Computing matrix functions solving coupled differential models"* (sección 3.2, [DSIR09]): En este primer trabajo se presenta un algoritmo que permite el cálculo simultáneo del seno y el coseno de una matriz. El cálculo se realiza mediante el uso de series de matrices polinomiales de Hermite y está basado en una modificación del método propuesto en [DJ98]. La implementación en MATLAB de este algoritmo consiguió muy buenos resultados al compararlo con la función `funm` de MATLAB, que permite calcular el seno y coseno matricial (usando un algoritmo de Shur–Parlett [DH03]).
- *"Computing matrix functions arising in engineering models with orthogonal matrix polynomials"* (sección 3.3, [DSIR13]): en este artículo se desarrolla un nuevo algoritmo para el cálculo del coseno que incluye mejoras muy significativas con respecto al presentado en el artículo anterior. Por una parte, se incluye escalado de la matriz, basándose en la fórmula del doble ángulo (ver sección 1.2.2): $cos(2A) = 2cos^2(A) - I$. Además, se utiliza el método de Horner y Paterson–Stockmeyer [PS73] para calcular el polinomio matricial de Hermite y también se establece una nueva cota para el error absoluto y se realizan comprobaciones de precisión para establecer si se pueden despreciar los términos de orden más alto de la serie polinómica sin que afecte a la precisión final. Una vez implementado en MATLAB, las pruebas numéricas realizadas demuestran que el algoritmo obtenido resulta muy competitivo al compararlo con otras funciones del estado del arte.
- *"Efficient computation of the matrix cosine"* (sección 3.4, [SIRD13]): por último, en este artículo se presenta el algoritmo más competitivo de todos. Se ajustan significativamente las cotas del error absoluto de tipo `forward` y se utiliza un algoritmo del mismo tipo que el aplicado a la exponencial en "Accurate matrix exponential compu-

tation to solve coupled differential models in Engineering" (sección 2.3). En las pruebas experimentales se compara este algoritmo en MATLAB con el presentado en el artículo anterior y con la función `cosm`, que es una implementación en MATLAB del algoritmo basado en aproximantes de Padé presentado en [HH05]; los resultados obtenidos mejoran tanto en precisión como en tiempos de ejecución a ambos algoritmos en la mayoría de los casos.

En las siguientes secciones de este capítulo se pueden consultar los artículos mencionados al completo.

3.2. Computing matrix functions solving coupled differential models [1]

Referencia del artículo:
E. Defez, J. Sastre, J. Ibáñez and P.A. Ruiz
Computing matrix functions solving coupled differential models
Mathematical and Computer Modelling, Volume 50, Issues 5-6, September 2009, Pages 831–839, ISSN 0895-7177, http://dx.doi.org/10.1016/j.mcm.2009.05.012

Abstract

In this paper a modification of the method proposed in [DJ98] for computing matrix sine and cosine based on Hermite matrix polynomial expansions is presented. An algorithm and illustrative examples demonstrate the performance of the new proposed method.

3.2.1. Introduction

It is well known that the wave equation

$$v^2 \frac{\partial^2 \psi}{\partial x^2} = \frac{\partial^2 \psi}{\partial t^2} \ , \tag{3.1}$$

plays an important role in many areas of engineering and applied sciences. The matrix differential problem

$$Y''(t) + AY(t) = 0 \ , \ Y(0) = Y_0 \ , \ Y'(0) = Y_1 \ , \tag{3.2}$$

where A is a matrix and Y_0 and Y_1 are vectors, arises from spatially semi-discretization of the wave equation (3.1), see [SB80]. Matrix problem (3.2) has the exact solution

$$Y(t) = \cos\left(\sqrt{A}t\right)Y_0 + \left(\sqrt{A}\right)^{-1} \sin\left(\sqrt{A}t\right)Y_1, \tag{3.3}$$

where $\sqrt{A}$ denotes any square root of a non-singular matrix A (see *e.g.* equation 1.2 of [HH05]). More general problems of type (3.2), with a forcing term $F(t)$ on the right-hand side arise from mechanical systems without damping, and their solutions can be expressed in terms of integrals involving the matrix sine and cosine [Ser79]. Thus, trigonometric matrix functions play an important role in second order differential systems, similar to matrix exponentials in first order differential problems.

A general algorithm for computing the matrix cosine which uses rational approximations and the double-angle formula $\cos(2A) = 2\cos^2(A) - I$ was proposed by Serbin and

[1] This work has been partially supported by the *Generalitat Valenciana* GV/2007/009.

Blalock [SB80]. Higham in [HH05, HS03, Hig08] developed a particular version of this algorithm based on the Padé approximation including truncation and rounding error analysis.

In this paper, that may be regarded as a continuation of [DJ98], we use Hermite matrix polynomial expansions of the matrix cosine and sine in order to perform a very accurate and competitive method for computing them compared to the results given by the function *funm* of MATLAB. The implementations have been tested on an Intel Core 2 Duo T5600 with 2 GB main memory, using 7.5 (R2007b) MATLAB version.

This paper is organized as follows. Section 3.2.2 summarizes previous results of Hermite matrix polynomials and includes a new Hermite series expansion of the matrix sine and cosine. Section 3.2.3 deals with the Hermite matrix polynomial series expansion of $\cos(At)$ and $\sin(At)$ for an arbitrary matrix as well as with its finite series truncation with a prefixed accuracy in a bounded domain, and an algorithm of the method is given. Section 3.2.4 deals with a selection of examples in order to investigate the accuracy of the new method proposed here. Finally, conclusions are presented in section 3.2.5.

Throughout this paper, $[x]$ denotes the integer part of x. The matrices I_r and $\theta_{r\times r}$ in $\mathbb{C}^{r\times r}$ denote the matrix identity and the null matrix of order r, respectively. Following [GL96], for a matrix A in $\mathbb{C}^{r\times r}$, its infinite-norm will be denoted by $\|A\|_\infty$ and its 2-norm will be denoted by $\|A\|_2$. Finally, if $A(k,n)$ are matrices in $\mathbb{C}^{r\times r}$ for $n\geq 0, k\geq 0$, from [DJ98] it follows that

$$\sum_{n\geq 0}\sum_{k\geq 0} A(k,n) = \sum_{n\geq 0}\sum_{k=0}^{n} A(k,n-k)\ . \tag{3.4}$$

3.2.2. Hermite matrix polynomials series expansions of matrix sine and matrix cosine

For the sake of clarity in the presentation of the following results we recall some properties of Hermite matrix polynomials which have been established in [DJ98] and [JC96]. From (3.4) of [JC96, p. 25] the nth Hermite matrix polynomial satisfies

$$H_n\left(x,\frac{1}{2}A^2\right) = n!\sum_{k=0}^{[\frac{n}{2}]}\frac{(-1)^k\,(xA)^{n-2k}}{k!(n-2k)!}\ , \tag{3.5}$$

for an arbitrary matrix A in $\mathbb{C}^{r\times r}$. Taking into account the three-term recurrence relationship (3.12) of [JC96, p. 26], it follows that

$$\left.\begin{array}{l} H_n\left(x,\frac{1}{2}A^2\right) = xAH_{n-1}\left(x,\frac{1}{2}A^2\right) - 2(n-1)H_{n-2}\left(x,\frac{1}{2}A^2\right)\quad,\quad n\geq 1 \\ H_{-1}(x,\frac{1}{2}A^2) = \theta_{r\times r}\quad,\quad H_0(x,\frac{1}{2}A^2) = I_r \end{array}\right], \tag{3.6}$$

and from its generating function in (3.1) and (3.2) [JC96, p. 24] one gets

$$e^{xtA-t^2I} = \sum_{n\geq 0} H_n\left(x,\frac{1}{2}A^2\right)t^n/n!,\quad |t|<\infty, \tag{3.7}$$

where $x, t \in \mathbb{C}$. The nth scalar Hermite polynomial is given by [Leb72, p. 60]

$$H_n(x) = n! \sum_{k=0}^{\left[\frac{n}{2}\right]} \frac{(-1)^k (2x)^{n-2k}}{k!(n-2k)!} \ , \ n \geq 0 \ , \tag{3.8}$$

which coincide with the $n-$th matrix Hermite polynomial (3.5) when $r = 1$ and $A = 2$.

Taking $y = tx$ and $\mu = 1/t$ in (3.7) it follows that

$$e^{Ay} = \mathrm{e}^{\frac{1}{\mu^2}} \sum_{n\geq 0} \frac{1}{\mu^n n!} H_n\left(\mu y, \frac{1}{2}A^2\right), \ \mu \in \mathbb{C}, \quad y \in \mathbb{C}, \quad A \in \mathbb{C}^{r\times r} \ . \tag{3.9}$$

Now, we look for the Hermite matrix polynomials series expansion of the matrix cosine $\cos(Ax)$. Given an arbitrary matrix $A \in \mathbb{C}^{r\times r}$, with

$$\cos(Ay) = \frac{e^{iAy} + e^{-iAy}}{2}$$

and using (3.9) in combination with [JC96, p. 25], it follows that

$$H_n(-x, A) = (-1)^n H_n(x, A) \ .$$

Thus, one gets

$$\cos(Ay) \ = e^{\frac{1}{\mu^2}} \sum_{n\geq 0} \frac{1}{\mu^{2n}(2n)!} H_{2n}\left(iy\mu, \frac{1}{2}A^2\right) \ . \tag{3.10}$$

Taking $\lambda = i\mu$ in (3.10), we obtain the looked for expression:

$$\cos(Ay) \ = e^{-\frac{1}{\lambda^2}} \sum_{n\geq 0} \frac{(-1)^n}{\lambda^{2n}(2n)!} H_{2n}\left(y\lambda, \frac{1}{2}A^2\right) \ . \tag{3.11}$$

In a similar form, taking into account that

$$\sin(Ay) = \frac{e^{iAy} - e^{-iAy}}{2i} \ ,$$

it follows that

$$\sin(Ay) \ = e^{-\frac{1}{\lambda^2}} \sum_{n\geq 0} \frac{(-1)^n}{\lambda^{2n+1}(2n+1)!} H_{2n+1}\left(y\lambda, \frac{1}{2}A^2\right). \tag{3.12}$$

Remark 3.2.1 *Observe that when $\lambda = 1$, expressions (3.11) and (3.12) are formulae (19) and (20) of [DJ98, p. 109].*

Denoting by $C_N(A, \lambda)$ the Nth partial sum of series (3.11) for $y = 1$, one gets

$$C_N(\lambda, A) = e^{-\frac{1}{\lambda^2}} \sum_{n=0}^{N} \frac{(-1)^n}{\lambda^{2n}(2n)!} H_{2n}\left(\lambda, \frac{1}{2}A^2\right) \approx \cos(A), \ \lambda \in \mathbb{C}, \quad A \in \mathbb{C}^{r\times r}. \tag{3.13}$$

Observe that the case $\lambda = 1$ corresponds with the matrix cosine approximation $C(A; 1; N)$ given in [DJ98]. Denoting by $S_N(A, \lambda)$ the Nth partial sum of series (3.12) for $y = 1$, one gets

$$S_N(\lambda, A) = e^{-\frac{1}{\lambda^2}} \sum_{n=0}^{N} \frac{(-1)^n}{\lambda^{2n+1}(2n+1)!} H_{2n+1}\left(\lambda, \frac{1}{2}A^2\right) \approx \sin(A), \ \lambda \in \mathbb{C}, \quad A \in \mathbb{C}^{r\times r}. \tag{3.14}$$

In Section 4 we shall see that the introduction of the additional parameter λ will improve the results given in [DJ98].

3.2.3. Accurate and error bounds for cosine and sine approximation. Algorithm

By (3.5) and (3.8), it follows that

$$\left\| H_n\left(x, \frac{1}{2}A^2\right)\right\|_2 \leq \sum_{k=0}^{[n/2]} \frac{n!\left(|x|\,\|A\|_2\right)^{n-2k}}{k!(n-2k)!}\ , \tag{3.15}$$

and thus

$$\left\| H_{2n}\left(\lambda, \frac{1}{2}A^2\right)\right\|_2 \leq \sum_{k=0}^{n} \frac{(2n)!\left(\lambda\,\|A\|_2\right)^{2(n-k)}}{k!(2(n-k))!}\ . \tag{3.16}$$

Using (3.4), the following expression holds

$$\sum_{n\geq 0}\sum_{k=0}^{n} \frac{\|A\|_2^{2(n-k)}}{\lambda^{2k}k!\,(2(n-k))!} \;=\; \cosh\left(\|A\|_2\right)e^{\frac{1}{\lambda^2}}\ . \tag{3.17}$$

Taking the approximate value $C_N(\lambda, A)$ given by (3.13) and taking into account (3.16), it follows that

$$\begin{aligned}
\|\cos(A) - C_N(\lambda, A)\|_2 &\leq \mathrm{e}^{-\frac{1}{\lambda^2}} \sum_{n\geq N+1} \frac{1}{\lambda^{2n}(2n)!}\left\| H_{2n}\left(\lambda, \frac{1}{2}A^2\right)\right\|_2 \\
&\leq \mathrm{e}^{-\frac{1}{\lambda^2}} \sum_{n\geq N+1}\sum_{k=0}^{n} \frac{\|A\|_2^{2(n-k)}}{\lambda^{2k}k!(2(n-k))!} \\
&= \mathrm{e}^{-\frac{1}{\lambda^2}}\left[\sum_{n\geq 0}\sum_{k=0}^{n} \frac{\|A\|_2^{2(n-k)}}{\lambda^{2k}k!(2(n-k))!} - \sum_{n=0}^{N}\sum_{k=0}^{n} \frac{\|A\|_2^{2(n-k)}}{\lambda^{2k}k!(2(n-k))!}\right].
\end{aligned}$$

Considering the previous expression, one gets an error bound for approximation (3.13):

$$\|\cos(A) - C_N(\lambda, A)\|_2 \leq \mathrm{e}^{-\frac{1}{\lambda^2}}\left[\cosh\left(\|A\|_2\right)e^{\frac{1}{\lambda^2}} - \sum_{n=0}^{N}\sum_{k=0}^{n} \frac{\|A\|_2^{2(n-k)}}{\lambda^{2k}k!(2(n-k))!}\right]. \tag{3.18}$$

Now, let $\varepsilon > 0$ be an *a priori* error bound. Using (3.18), if N is the first positive integer so that

$$\sum_{n=0}^{N}\sum_{k=0}^{n} \frac{\|A\|_2^{2(n-k)}}{\lambda^{2k}k!(2(n-k))!} \geq \cosh\left(\|A\|_2\right)e^{\frac{1}{\lambda^2}} - \varepsilon\,\mathrm{e}^{\frac{1}{\lambda^2}}\ , \tag{3.19}$$

from (3.18) and (3.19) one gets,

$$\|\cos(A) - C_N(\lambda, A)\|_2 \;\leq\; \varepsilon\ .$$

Summarizing, the next result, similar to theorem 3.1 of [DJ98], has been proved:

Theorem 7 *Let A be a matrix in $\mathbb{C}^{r\times r}$ and let $\lambda > 0$. Let $\varepsilon > 0$. If N is the first positive integer so that inequality (3.19) holds. Then*

$$\|\cos(A) - C_N(\lambda, A)\|_2 \;\leq\; \varepsilon\ . \tag{3.20}$$

Furthermore, using that relation $\sin(A) = \cos\left(A - \frac{\pi}{2}I\right)$, it is possible avoid the computation of the matrix sine. On the other hand, we can obtain a similar result to theorem 7 for the case of the matrix sine:

Theorem 8 *Let A be a matrix in $\mathbb{C}^{r\times r}$ and let $\lambda > 0$. Let $\varepsilon > 0$. If N is the first positive integer so that inequality*

$$\sum_{n=0}^{N}\sum_{k=0}^{n}\frac{\|A\|_2^{2(n-k)+1}}{\lambda^{2k}k!(2(n-k)+1)!} \geq \sinh\left(\|A\|_2\right)e^{\frac{1}{\lambda^2}} - \varepsilon\, e^{\frac{1}{\lambda^2}} \ ,$$

holds. Then, approximation $S_N(\lambda, A)$ given by (3.14) satisfies

$$\|\sin(A) - S_N(\lambda, A)\|_2 \ \leq \ \varepsilon \ . \tag{3.21}$$

Starting with expressions (3.13) and (3.14), it is possible to simultaneously compute the matrix cosine and sine using the following algorithm 3.1.

Algorithm 3.1 computes sine and cosine of a matrix by means of Hermite approximants.

Function $[C, S] = sincosher(A, N, \lambda)$
Inputs: Matrix $A \in \mathbb{R}^{r\times r}$; $2N+1$ is the order of the Hermite approximation ($N \in \mathbb{N}$) of sine/cosine function; parameter $\lambda \in \mathbb{R}$
Output: Matrices $C = \cos(A) \in \mathbb{R}^{r\times r}$ and $S = \sin(A) \in \mathbb{R}^{r\times r}$

$H_0 = I_r$
$H_1 = \lambda A$
$C = H_0$
$S = H_1/\lambda$
$aux = 1/\lambda$
for $n = 2 : 2N+1$ **do**
  $H = \lambda A H_1 - 2(n-1)H_0$
  $H_0 = H_1$;
  $H_1 = H$
  $aux = aux/(\lambda n)$
  if $\mathrm{mod}(n, 4) < 2$ **then**
    if $\mathrm{mod}\,(n, 2) == 0$ **then**
      $C = C + auxH$;
    else
      $C = C - auxH$;
    end if
  else
    if $\mathrm{mod}\,(n, 2) == 0$ **then**
      $S = S + auxH$;
    else
      $S = S - auxH$;
    end if
  end if
end for
$C = e^{-1/l^2}C$
$S = e^{-1/l^2}S$

3.2.4. Numerical examples

In this section we provide results for numerical experimentation of the computational method based on expansion (3.11) compared with the results given by the function *funm* of MATLAB. This function allows to compute general matrix functions by the Schur-Parlett algorithm, [DH03], and it is the only function that MATLAB has to compute matrix sine and cosine. The implementations have been tested on an Intel Core 2 Duo T5600 with 2 GB main memory, using 7.5 (R2007b) MATLAB version.

In the first example, we apply the computation of the matrix cosine of a matrix A treated in [DJ98] using the expansion (3.11). Note that there are different possible choices for the parameter λ.

Example 3.2.1 *Let A be a matrix defined by*

$$A = \begin{pmatrix} 3 & -1 & 1 \\ 2 & 0 & 1 \\ 1 & -1 & 2 \end{pmatrix}, \tag{3.22}$$

with $\sigma(A) = \{1, 2\}$. Matrix A is non-diagonalizable. Using the minimal theorem [DS57, p. 571], see also [DJ98], the exact value of $\cos(A)$ is

$$\cos(A) = \begin{pmatrix} \cos(2) - \sin(2) & \sin(2) & -\sin(2) \\ -\cos(1) + \cos(2) - \sin(2) & \cos(1) + \sin(2) & -\sin(2) \\ -\cos(1) + \cos(2) & \cos(1) - \cos(2) & \cos(2) \end{pmatrix}$$
$$= \begin{pmatrix} -1.325444263372824 & 0.909297426825682 & -0.909297426825682 \\ -1.865746569240964 & 1.449599732693821 & -0.909297426825682 \\ -0.956449142415282 & 0.956449142415282 & -0.4161468365471424 \end{pmatrix}.$$

In [DJ98], for an admissible error $\varepsilon = 10^{-5}$, we need $N = 15$ to provide the required accuracy. In practice, the number of terms required to obtain a prefixed accuracy uses to be smaller than the one provided by Theorem 3.1 of [DJ98]. So for instance, taking $N = 9$ one gets:

$$C_9(1, A) = \begin{pmatrix} -1.3254444650245485 & 0.9092974459509594 & -0.9092974459509594 \\ -1.8657468968644513 & 1.4495998777908623 & -0.9092974459509594 \\ -0.9564494509134919 & 0.9564494509134919 & -0.4161470190735891 \end{pmatrix},$$

and

$$\|\cos(A) - C_9(1, A)\|_2 = 7.995228661905607 \times 10^{-7}.$$

We will compare these results obtained letting $\lambda = 1$ in Theorem 3.1 of [DJ98] with the new Theorem 7. Taking $\lambda = 2000$, using Theorem 7 we need $N = 10$ to obtain the same prefixed accuracy. Again, the number of terms required to obtain a prefixed accuracy uses to be smaller than the one provided by (3.20). For instance, taking $N = 7$ one gets

$$C_7(2000, A) = \begin{pmatrix} -1.3254442633775207 & 0.9092974268299509 & -0.9092974268299509 \\ -1.8657465692456603 & 1.4495997326980907 & -0.9092974268299509 \\ -0.9564491424157093 & 0.9564491424157093 & -0.41614683654756973 \end{pmatrix},$$

and

$$\|\cos(A) - C_7(2000, A)\|_2 = 7.717270333884585 \times 10^{-8}.$$

The choice of parameter λ *can still be refined. For example, taking* $\lambda = 4.1$ *one gets*

$$\|\cos(A) - C_7(4.1, A)\|_2 = 7.098351906265066 \times 10^{-10} .$$

Figure 3.1 presents the error 2-norm of approximation (3.11) for $N = 8$ *fixed and* $\lambda \in]0, 25]$.

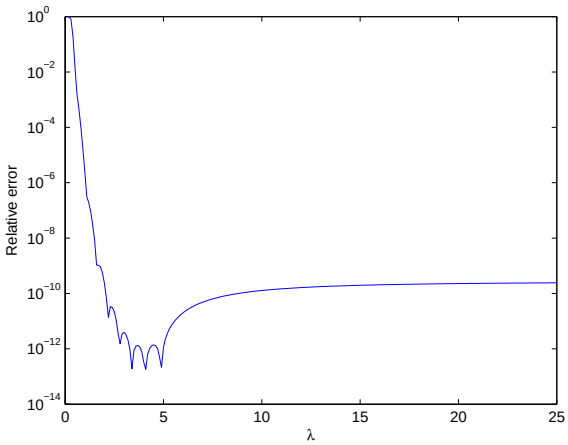


Figura 3.1: For $N = 8$ fixed and varying λ.

This figure illustrates how the error norm depends on the varying parameter λ *and it becomes evident that an adequate choice of* λ *may provide results with higher accuracy.*

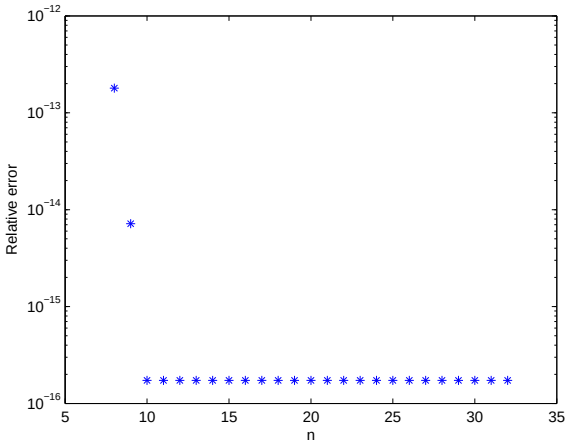


Figura 3.2: Relative error of Hermite series (3.11) for example 3.2.1 for $\lambda = 4.1$.

Figure 3.2 shows the 2–norm error bound of $C_N(\lambda, A)$ for the fixed value of $\lambda = 4.1$ varying N. For $N = 10$, we obtain

$$\|\cos(A) - C_{10}(4.1, A)\|_2 = 1.7763568394002505 \times 10^{-15} .$$

Example 3.2.2 *In this experiment we consider* 100 *random matrices of the form*

$$A = PDP^{-1}, \tag{3.23}$$

where D *is a diagonal matrix with uniform random values in the interval* $[-5, 5]$ *and* P *is a matrix with uniform random values in the same interval. The dimensions of all matrices are* 100×100. *We have computated the approximation of the matrix cosine* $C_N(\lambda, A)$ *and sine* $S_N(\lambda, A)$ *with* $N = 20$ *and the experimental value of* λ *was* $\lambda = 0.7936$.

It is well known that the exact solutions are

$$\cos(A) = P\cos(D)P^{-1} \quad , \quad \sin(A) = P\sin(D)P^{-1} \ .$$

In the experiment each exact solution has been obtained at 256*−digit precision using MATLAB's Symbolic Math Toolbox.*

Figure 3.3 shows the comparison between the relative errors of function `funm` *of MATLAB and series (3.11) with* $\lambda = 0.7936$ *using the infinite norm:*

$$E_r(x^\star) = \frac{\|x - x^\star\|_\infty}{\|x\|_\infty} \ . \tag{3.24}$$

The mean processing time for `funm` *was* 0.114550 *seconds and the mean processing time for the Hermite approximation was* 0.023535 *seconds. The first average time corresponds only to the computation of* $\cos(A)$ *using the function* `funm`. *The second value corresponds to the computation of* $\cos(A)$ *and* $\sin(A)$ *using Hermite expansion. Our proposed implementation was* 4.8672 *times faster. In the computation of* $\cos(A)$, *the Hermite method gave a smaller error than* `funm` *in* 70 % *of the test cases. In the computation of* $\sin(A)$, *the Hermite method gave a smaller error than* `funm` *in* 67 % *of the test cases.*

Example 3.2.3 *We consider 100 randomly matrices in the same conditions as in experiment 3.2.2. We have computed the approximation of the matrix cosine* $C_N(\lambda, A)$ *and sine* $S_N(\lambda, A)$ *with* $N = 25$. *We choose in this new experiment* $\lambda = 0.6175$.

Figure 3.4 shows the comparison between the relative errors of function `funm` *of MATLAB and series (3.11) with* $\lambda = 0.6175$ *using infinite norm (3.24).*

Now, the mean processing time for `funm` *was* 0.113997 *seconds and the mean processing time for the Hermite approximation was* 0.027875 *seconds. The first average time corresponds only to the computation of* $\cos(A)$ *using the function* `funm`. *The second value corresponds to the computation of* $\cos(A)$ *and* $\sin(A)$ *using Hermite expansion. Our proposed implementation was* 4.0896 *times faster. In the computation of* $\cos(A)$, *the Hermite method gave a smaller error than* `funm` *in* 74 % *of the test cases. In the computation of* $\sin(A)$, *the Hermite method gave a smaller error than* `funm` *in* 74 % *of the test cases.*

3.2.5. Conclusions

In this paper a modification of the method proposed in [DJ98] for computing matrix cosine and sine based on Hermite matrix polynomial expansion is presented. Numerical tests and an algorithm are given. The described method allows the simultaneous evaluation of the matrix sine and cosine and it has been compared with the function `funm` of MATLAB. The method depends on the parameter λ, whose impact on the numerical efficiency is currently studied. Furthermore, pending work focuses on the optimal scaling of the matrix and the study of the evaluation [PS73] of the approximations (3.13) and (3.14). To do parallel implementation of the algorithms presented in this work in a distributed memory platform, using the message passing paradigm, MPI and BLACS for communications, and PBLAS and ScaLAPACK [BCC+97] for computations.

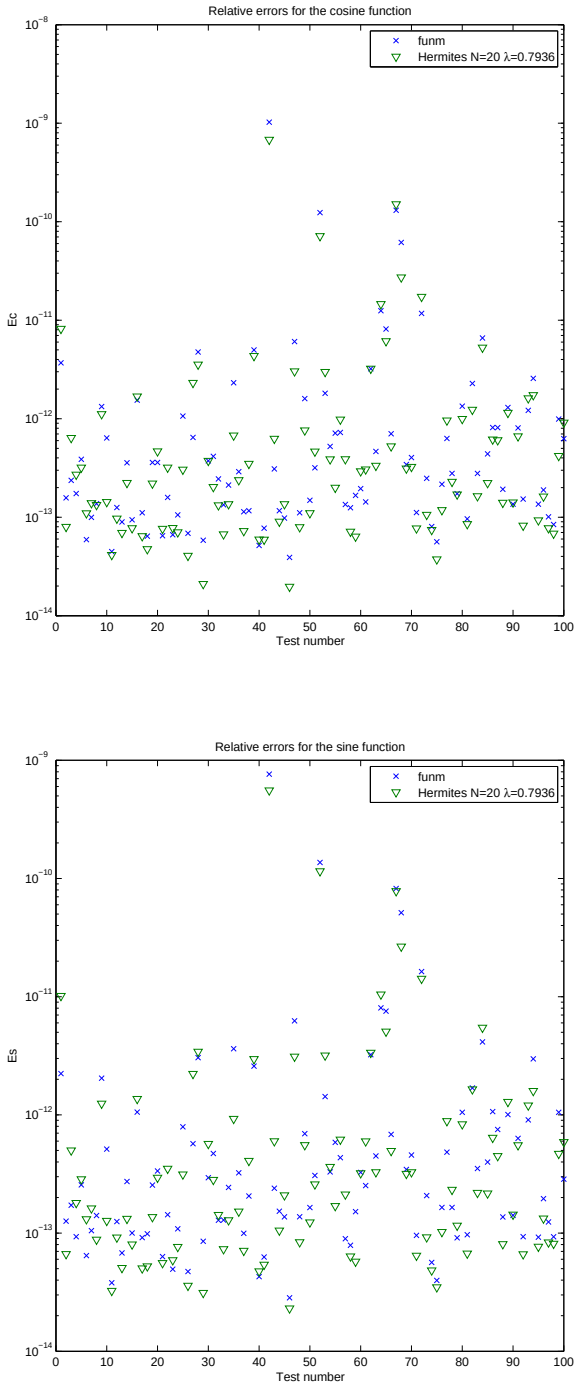


Figura 3.3: Comparison between the relative errors for cosine and sine computation with $N = 20$ and $\lambda = 0.7936$.

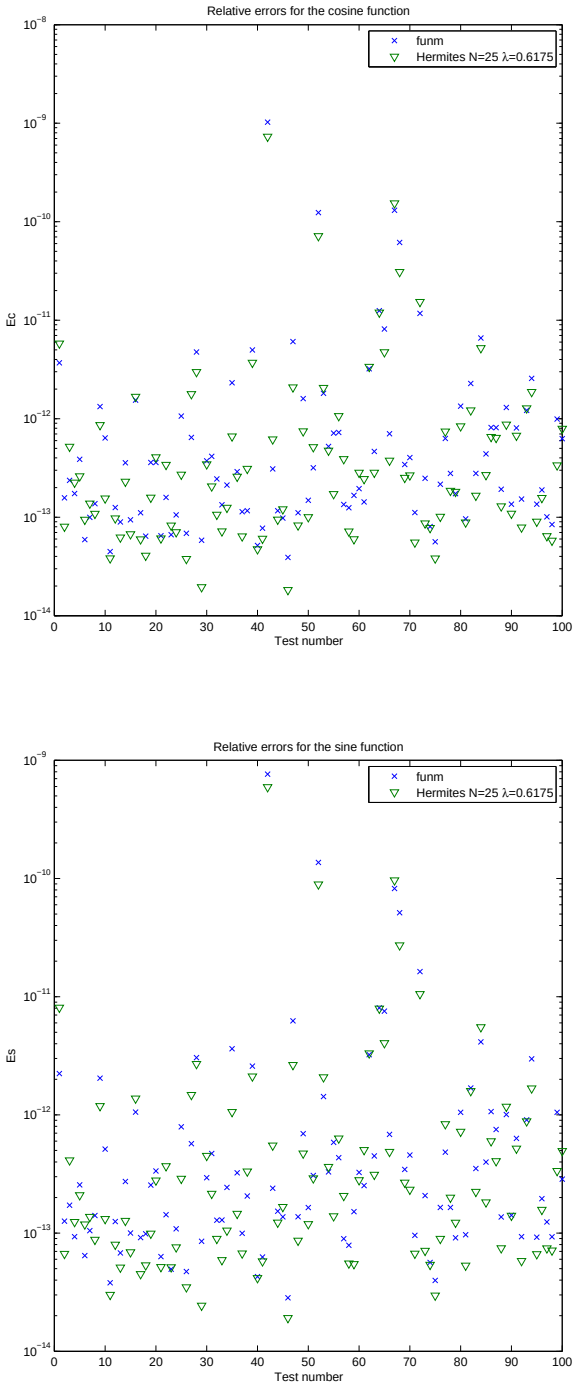


Figura 3.4: Comparison between the relative errors for cosine and sine computation with $N = 25$ and $\lambda = 0.6175$.

3.3. Computing matrix functions arising in engineering models with orthogonal matrix polynomials[2]

Referencia del artículo:
E. Defez, J. Sastre, J. Ibáñez and P. Ruiz
Computing matrix functions arising in engineering models with orthogonal matrix polynomials
Mathematical and Computer Modelling, Volume 57, Issues 7-8, April 2013, Pages 1738–1743, ISSN 0895-7177, http://dx.doi.org/10.1016/j.mcm.2011.11.022

Abstract

Trigonometric matrix functions play a fundamental role in the solution of second order differential equations. Hermite series truncation together with Paterson-Stockmeyer method and the double angle formula technique allow efficient computation of the matrix cosine. A careful error bound analysis of the Hermite approximation is given and a theoretical estimate for the optimal value of its parameters is obtained. Based on the ideas above, an efficient and highly-accurate Hermite algorithm is presented. A MATLAB implementation of this algorithm has also been developed and made available online. This implementation has been compared to other efficient state-of-the-art implementations on a large class of matrices for different dimensions, obtaining higher accuracy and lower computational costs in the majority of cases.

3.3.1. Introduction

Matrix functions play a relevant role in different areas of science and technology. They arise most frequently in connection with the solution of differential systems and control theory. For example, it is well known that the wave equation

$$v^2 \frac{\partial^2 \psi}{\partial x^2} = \frac{\partial^2 \psi}{\partial t^2} \ , \tag{3.25}$$

plays an important role in many areas of engineering and applied sciences. When we use the spatially semi-discretization method of the wave equation (3.25), we obtain the matrix differential problem

$$Y''(t) + AY(t) = 0 \ , \ Y(0) = Y_0 \ , \ Y'(0) = Y_1 \ , \tag{3.26}$$

where A is a square matrix and Y_0 and Y_1 are vectors, see [SB80] for details. Matrix problem (3.26) has the solution

$$Y(t) = \cos\left(\sqrt{A}t\right)Y_0 + \left(\sqrt{A}\right)^{-1} \sin\left(\sqrt{A}t\right)Y_1, \tag{3.27}$$

where $\sqrt{A}$ denotes any square root of a non-singular matrix A (see *e.g.* expression (1.2) of [HH05]). More general problems of type (3.26), with a forcing term $F(t)$ on the right-hand

[2]This work has been supported by the Spanish *Ministerio de Educación* grant MTM2009-08587.

side arise from mechanical systems without damping, and their solutions can be expressed in terms of integrals involving the matrix sine and cosine [Ser79]. Thus, trigonometric matrix functions play an important role in second order differential systems, similar to matrix exponential e^{At} in first order differential systems [ML03].

Moreover, the matrix cosine is used in the method of Yau-Lu for reducing the symmetric eigenvalue problem (finding all the eigenvalues and eigenvectors of a dense symmetric matrix) to a number of matrix multiplications [YL93].

Computing the matrix sine reduces to computing the matrix cosine through $\sin(A) = \cos(A - \frac{\pi}{2}I)$. Thus we concentrate on the matrix cosine. Serbin and Blalock proposed a general algorithm for computing the matrix cosine in [SB80], which uses rational approximations and the double angle formula

$$\cos(2A) = 2\cos^2(A) - I. \tag{3.28}$$

In [DJ98] new methods for computing matrix exponential, sine and cosine based on Hermite matrix polynomial series were presented. A new bound for Hermite matrix polynomials was provided and it was used to give expressions for obtaining the number of Hermite series terms depending on the desired approximation error in exact arithmetic. Later, Higham, Smith and Hargreaves developed two algorithms based on (3.28) and Padé approximation in [HS03, HH05], including truncation and rounding error analysis. Recently, in [DSIR09] the authors introduced a new parameter in the matrix cosine and sine Hermite series from [DJ98] and new error bounds, improving both accuracy and efficiency.

In this paper we use the new matrix cosine Hermite series from [DSIR09], providing sharper bounds for Hermite matrix polynomials and the approximation error, and computing the optimal values of the series parameter to develop a competitive Hermite algorithm for computing the matrix cosine in IEEE double precision arithmetic that uses: matrix scaling based on (3.28), Paterson-Stockmeyer's method for the evaluation of Hermite series [PS73, SIDR11b], and accuracy bound tests similar to those proposed in [SIDR11b, p. 6456-6457]. A MATLAB implementation of this algorithm is made available online and it is compared with the MATLAB function `funm` [DH03] and a MATLAB implementation based on the Padé algorithm given in [HH05], i.e. function `cosm`, providing higher accuracy and efficiency than both methods in the majority of test matrices.

This paper is organized as follows. Section 3.3.2 summarizes previous results of Hermite matrix polynomial series expansion of $\cos(A)$ and the development of a new error bound. In Section 3.3.3, an algorithm based on that error bound is described. Numerical experiments are presented in Section 3.3.4. Finally, conclusions are given in Section 3.3.5.

Throughout this paper, $[x]$ denotes the integer part of x. To obtain the above mentioned error bound, we will use any subordinate matrix norm $\|A\|$, $A \in \mathbb{C}^{r\times r}$, and in the subsequent error analysis, we will use the 1-norm $\|A\|_1$. If $\mathcal{A}(k,n)$ is a matrix in $\mathbb{C}^{r\times r}$ for $n \geq 0, k \geq 0$, the following identity holds [DJ98]:

$$\sum_{n\geq 0}\sum_{k\geq 0}\mathcal{A}(k,n) = \sum_{n\geq 0}\sum_{k=0}^{n}\mathcal{A}(k,n-k). \tag{3.29}$$

3.3.2. Hermite matrix polynomial series expansions of matrix cosine. Error bound

For the sake of clarity in the presentation of the following results we recall some properties of Hermite matrix polynomials which have been established in [JC96, DJ98, DSIR09]. From (3.4) of [JC96, p. 25] the nth Hermite matrix polynomial satisfies

$$H_n\left(x, \frac{1}{2}A^2\right) = n! \sum_{k=0}^{[\frac{n}{2}]} \frac{(-1)^k (xA)^{n-2k}}{k!(n-2k)!}, \tag{3.30}$$

for an arbitrary matrix A in $\mathbb{C}^{r\times r}$. From [DSIR09], we have the following Hermite matrix polynomial series expansion of the matrix cosine $\cos(Ay)$:

$$\cos(Ay) = e^{-\frac{1}{\lambda^2}} \sum_{n\geq 0} \frac{(-1)^n}{\lambda^{2n}(2n)!} H_{2n}\left(y\lambda, \frac{1}{2}A^2\right). \tag{3.31}$$

Denoting by $C_N(\lambda, A^2)$ the Nth partial sum of series (3.31) for $y = 1$, one gets the approximation

$$C_N(\lambda, A^2) = e^{-\frac{1}{\lambda^2}} \sum_{n=0}^{N} \frac{(-1)^n}{\lambda^{2n}(2n)!} H_{2n}\left(\lambda, \frac{1}{2}A^2\right) \approx \cos(A), \ \lambda \in \mathbb{C}. \tag{3.32}$$

Working similarly as in [DTS11, pp. 1913], we can obtain a bound for Hermite matrix polynomials $\left\|H_{2n}\left(x, \frac{1}{2}A^2\right)\right\|$ based on $\|A^2\|$, see [HH05], using the Taylor series for the hyperbolic cosine $\cosh(y) = \sum_{n\geq 0} y^{2n}/(2n)!$. Taking norms in (3.30), one gets

$$\left\|H_{2n}\left(x, \frac{1}{2}A^2\right)\right\| \leq (2n)! \sum_{k=0}^{n} \frac{\left(\left\|A^2\right\|^{\frac{1}{2}}\right)^{2(n-k)}}{k!(2(n-k))!} |x|^{2(n-k)}. \tag{3.33}$$

On the other hand, using (3.29), it follows that

$$\begin{aligned} e\cosh\left(|x|\left\|A^2\right\|^{\frac{1}{2}}\right) &= \sum_{n\geq 0} \frac{\left(|x|\left\|A^2\right\|^{\frac{1}{2}}\right)^{2n}}{(2n)!} \sum_{k\geq 0} \frac{1}{k!} = \sum_{n\geq 0}\sum_{k\geq 0} \frac{\left(|x|\left\|A^2\right\|^{\frac{1}{2}}\right)^{2n}}{(2n)!k!} \\ &= \sum_{n\geq 0}\sum_{k=0}^{n} \frac{\left(\left\|A^2\right\|^{\frac{1}{2}}\right)^{2(n-k)}}{k!(2(n-k))!} |x|^{2(n-k)}, \end{aligned} \tag{3.34}$$

and as a consequence

$$\sum_{k=0}^{n} \frac{\left(\left\|A^2\right\|^{\frac{1}{2}}\right)^{2(n-k)}}{k!(2(n-k))!} |x|^{2(n-k)} \leq e\cosh\left(|x|\left\|A^2\right\|^{\frac{1}{2}}\right). \tag{3.35}$$

Multiplying by $(2n)!$ in (3.35) and using (3.33), we have the result:

$$\left\|H_{2n}\left(x, \frac{1}{2}A^2\right)\right\| \leq (2n)!\, e\cosh\left(x\left\|A^2\right\|^{\frac{1}{2}}\right), \ \forall x \in \mathbb{R}, \ n \geq 0, \ \forall A \in \mathbb{C}^{r\times r}. \tag{3.36}$$

Taking into account approximation (3.32) and bound (3.36), it follows that

$$\begin{aligned}\left\|\cos(A) - C_N(\lambda, A^2)\right\| &\leq e^{-\frac{1}{\lambda^2}} \sum_{k\geq N+1} \frac{1}{\lambda^{2k}(2k)!} \left\|H_{2k}\left(\lambda, \frac{1}{2}A^2\right)\right\| \\ &\leq e^{1-\frac{1}{\lambda^2}} \cosh\left(\lambda \left\|A^2\right\|^{\frac{1}{2}}\right) \sum_{k\geq N+1} \frac{1}{\lambda^{2k}} \\ &= e^{1-\frac{1}{\lambda^2}} \cosh\left(\lambda \left\|A^2\right\|^{\frac{1}{2}}\right)\left[\sum_{k\geq 0} \frac{1}{\lambda^{2k}} - \sum_{k=0}^{N} \frac{1}{\lambda^{2k}}\right]. \end{aligned} \tag{3.37}$$

Simplifying the geometric series in (3.37), we have finally the bound:

$$\left\|\cos(A) - C_N(\lambda, A^2)\right\| \leq \frac{e^{1-\frac{1}{\lambda^2}} \cosh\left(\lambda \left\|A^2\right\|^{\frac{1}{2}}\right)}{(\lambda^2 - 1)\lambda^{2N}}. \tag{3.38}$$

3.3.3. Algorithm

In this section we describe Algorithm 3.2 (`cosher`), based on Hermite series, which uses bound (3.38) for choosing the scaling factor for computing the matrix cosine.

Algorithm 3.2 `(cosher)` Given a matrix $A \in \mathbb{C}^{r\times r}$ and the order N of Hermite matrix approximation of the cosine function, this algorithm computes $B \cong \cos(A)$.

1: Preprocessing of matrix A: $\check{A}$.
2: Compute optimal values of λ and s.
3: SCALING PHASE: $\tilde{A} = \check{A}/2^s$
4: Compute $\tilde{B} = C_N(\lambda, \tilde{A}^2)$, where C_N is the Hermite approximation of the cosine function.
5: **for** $i = 1 : s$ **do**
6: $\quad \tilde{B} = 2\tilde{B}^2 - I$
7: **end for**
8: Postprocessing of matrix $\tilde{B}$: B.

The preprocessing and postprocessing are based on applying transformations to reduce the norm of matrix A and recover the matrix $B \cong \cos(A)$ from the matrix $\tilde{B}$ obtained in the Loop 5-7. The available techniques to reduce the norm of a matrix are argument translation and balancing [Hig08, p. 299]. The argument translation is based on the formula

$$\cos(A - \pi j I) = (-1)^j \cos(A),\ k \in \mathbb{Z},$$

and on finding the integer q such that the norm of matrix $A - \pi q I$ is minimum. This value can be calculated by using Theorem 4.18 from [Hig08]. Balancing is a heuristic that attempts to equalize the norms of the kth row and kth column, for each k, by a diagonal similarity transformation defined by a non singular matrix D. Balancing tends to reduce the norm, though this is not guaranteed, so we will use it only for matrices where the norm is really reduced. For those matrices, if $\check{A} = D^{-1}(A - \pi q I)D$ is the obtained matrix in the preprocessing, the postprocessing consists of computing $B = (-1)^q D \cos(\tilde{B}) D^{-1}$.

For the evaluation of $C_N(\lambda, \tilde{A}^2)$ the Horner and Paterson-Stockmeyer's method can be applied [PS73, SIDR11b], obtaining previously with MATLAB the symbolic expression of $C_N(\lambda, \tilde{A}^2)$ as a polynomial of matrix $\tilde{A}^2$ with degree N and coefficients depending on λ, for each considered value of N. The double angle formula (3.28) is used to recover $\cos(\check{A})$ from the matrix $\tilde{B}$ obtained in Step 4, see [SB80] for details.

The optimal values of N with respect to cost of the evaluation of matrix polynomial $C_N(\lambda, \tilde{A}^2)$ with Paterson-Stockmeyer method are included in the set $S_N = \{1, 2, 4, 6, 9, 12, 16, 20, \cdots\}$, see [SIDR11b, p. 6454]. The scaling factor s and the parameter λ is chosen as follows. If N_k, where k is the position of N in S_N, is the chosen order of Hermite approximation, the number of matrix product evaluations in Algorithm 3.2 is equal to $k+s$. Hence, the number of matrix products depends on the scaling factor s. For the evaluation of $\cos(A)$ with IEEE double precision arithmetic, analogously to [HH05], we consider an absolute error-based algorithm. If we take the error in (3.38) to be lower than or equal to the unit roundoff in double precision floating-point $u = 2^{-53}$, i.e,

$$\frac{e^{1-\frac{1}{\lambda^2}} \cosh\left(\lambda \sqrt{\left\| \left(\check{A}/2^s\right)^2 \right\|_1}\right)}{(\lambda^2-1)\lambda^{2N}} \leq u, \tag{3.39}$$

then

$$s \geq \frac{1}{2}\log_2\left(\left\|\check{A}^2\right\|_1\right) + \log_2\left(\frac{\lambda}{\operatorname{arc\,cosh}\left(\frac{u(\lambda^2-1)\lambda^{2N}}{e^{1-1/\lambda^2}}\right)}\right). \tag{3.40}$$

We choose the parameter $\lambda = \lambda_{\text{mín}}$, where $\lambda_{\text{mín}}$ is the value of λ such that the right-hand side of (3.40) reaches the minimum value. Hence, if we define

$$\Theta_N = \frac{1}{\lambda_{\text{mín}}} \operatorname{arc\,cosh}\left(\frac{u(\lambda_{\text{mín}}^2-1)\lambda_{\text{mín}}^{2N}}{e^{1-1/\lambda_{\text{mín}}^2}}\right) \tag{3.41}$$

then, using (3.40), it follows that

$$s = \left\lceil \log_2\left(\frac{\sqrt{\|\check{A}^2\|_1}}{\Theta_N}\right)\right\rceil. \tag{3.42}$$

We discard the values of N except for $\{1, 2, 4, 6, 9, 12, 16, 20\}$, because for $N = 25, 30$ the corresponding values of $\lambda_{\text{mín}}$ are negative. The values of $\lambda_{\text{mín}}$ are listed in Table 3.5a. These values are independent of the norm of matrix $\check{A}^2$, see (3.40), and they have been computed by using the symbolic functions `diff` and `solve` from the Symbolic Math Toolbox 5 of MATLAB. The values Θ_N of (3.41) are listed in Table 3.5b.

	$\lambda_{\text{mín}}$
N=1	28614.3702451495925
N=2	1304.99637514915918
N=4	110.428178898694292
N=6	38.3201292093300207
N=9	17.3255806739152432
N=12	11.2995380153548675
N=16	8.08117035928883672
N=20	6.56678564572528643

(a) Optimal values of $\lambda_{\text{mín}}$.

	Θ_N
N=1	$1.3988322173046763 \cdot 10^{-4}$
N=2	$4.5977704110066707 \cdot 10^{-3}$
N=4	$9.0556596644120163 \cdot 10^{-2}$
N=6	$3.6534325997941364 \cdot 10^{-1}$
N=9	1.1543637495804793
N=12	2.3009899711770276
N=16	4.2073703112196084
N=20	6.3959908727565082

(b) Values of Θ_N from (3.41).

Figura 3.5: Tables of $\lambda_{\text{mín}}$ and Θ_N.

In the final MATLAB implementation of `cosher`, available online in `http://personales.upv.es/~jorsasma/cosher.m`, we have applied similar accuracy bound tests to those in [SIDR11b, p. 6456-6457] to the sum of the highest degree terms of the Hermite series $C_N(\lambda, \tilde{A}^2)$. The tests consist of verifying if the norm of the sum of several highest degree terms is lower than the unit roundoff. If that is the case, then the corresponding series terms can be neglected, permitting to save matrix products. This kind of tests were already performed with the matrix exponential Hermite series in [SIDR11b], saving matrix products and therefore reducing the cost for some matrices. For a complete description of algorithm `cosher` see MATLAB implementation `cosher.m` mentioned above.

The analysis of rounding errors of Algorithm 3.2 can be made analogously to the analysis of rounding errors given in [Hig08, p. 293] for polynomials $p_{2m}(A)$ or $q_{2m}(A)$ from the Padé approximation, see (12.25) from [Hig08, p. 293], and the analysis of the rounding error in the evaluation of A^2 given in the same reference. A complete analysis is given in [DSIR11].

3.3.4. Numerical examples.

In this section we compare MATLAB implementation `cosher` with functions `funm` and `cosm`. `funm` is a MATLAB function that computes matrix cosine and other matrix functions at square matrices using the Schur-Parlett algorithm from [DH03]. `cosm` is a MATLAB implementation of Algorithm 5.1 proposed in [HH05] which uses Padé approximants of cosine function (`http://www.maths.manchester.ac.uk/~higham/mftoolbox`). MATLAB 7.9 (R2009b) implementations were tested on an Intel Core 2 Duo processor at 2.52 GHz with 4 GB main memory. Algorithm accuracy was tested by computing the relative error

$$\mathrm{E} = \frac{\|\cos(A) - \tilde{Y}\|_1}{\|\cos(A)\|_1},$$

where $\tilde{Y}$ is the computed solution and $\cos(A)$ the exact solution. In the tests we did not use any preprocessing/postprocessing in the implemented algorithms. Analogously to the experiments in [Hig05], we found that turning on preprocessing in this algorithm provided similar results to those presented in this section without preprocessing. We used a set of 102 test matrices: forty-seven 10×10 matrices obtained from the function `matrix` of the Matrix Computation Toolbox [Hig93], twenty four 9×9 or 10×10 matrices from the Eigtool

MATLAB package (`http://web.comlab.ox.ac.uk/pseudospectra/eigtool`), twenty four matrices from the state-of-the-art of matrix functions [DH03, War77, ML03, NH95, Wes90, Lu03, DP00, SIDR11b] and seven from built-in MATLAB functions, sized from 2×2 to 20×20. For a complete description of the set of text matrices see [SIDR11b]. The "exact"matrix cosine was calculated analytically when possible, and otherwise using MATLAB's Symbolic Math Toolbox with high precision arithmetic.

Tables 3.6a and 3.6b show the comparatives `cosher-cosm` and `cosher-funm` for $N \in \{9, 12, 16, 20\}$. The first three rows show the percentages of times that the relative error of the first function is lower, equal or greater than the relative error of the second function. The fourth row shows the ratio of matrix products needed for computing the matrix cosine for over all the test matrices. In the same way as in [SIDR11b, p. 6459] we have considered that the asymptotic cost in terms of matrix products for solving the multiple right-hand side linear system that appears in Padé algorithm is 4/3. The computational cost of `funm` depends greatly on the eigenvalue distribution of the considered matrix. If $A \in \mathbb{C}^{r\times r}$, this cost is roughly between $28r^3$ flops and $r^4/3$ flops [Hig08, p. 228]. Since the cost of a square matrix product is $2r^3$ flops, we estimated by default that the cost of `funm` is 14 matrix products.

	N=9	N=12	N=16	N = 20
L	33.33	67.65	84.31	72.55
E	0	0	0	0
G	66.67	32.35	15.69	27.45
R	0.94	0.92	0.92	0.91

(a) Comparative `cosher-cosm`.

	N=9	N=12	N=16	N=20
L	64.71	74.51	77.45	73.53
E	0	0	0	0
G	35.29	25.49	22.55	26.47
R	0.60	0.58	0.58	0.58

(b) Comparative `cosher-funm`.

Figura 3.6: Comparatives `cosher-cosm` and `cosher-cosm`. The first three rows show the percentage of times that relative error of `cosher` is lower (L), equal (E) or greater (G) than relative error of `cosm` or `funm`. The last row shows the ratio (R) of costs in terms of matrix products between `cosher` and `cosm` (3.6a), and `cosher` and `funm` (3.6b).

Figure 3.7 shows the performances [DM02] of the functions compared, where α coordinate varies between 1 and 5 in steps equal to 0.1, and p coordinate is the probability that the considered algorithm has a relative error lower than or equal to α-times the smallest error over all the methods, where probabilities are defined over all matrices.

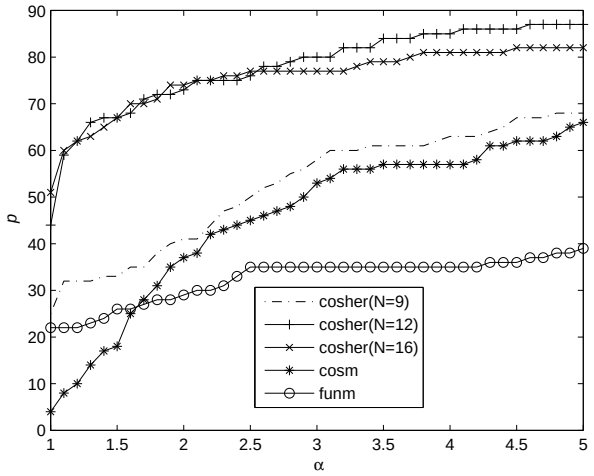


Figura 3.7: Performance profile of `cosher`, `cosm` and `funm` for the set of test matrices.

From tests, the following conclusions can be emphasized:

- Computational cost of `cosher` is lower than computational costs of `cosm` and `funm` for all considered orders.
- Function `cosher` is more accurate than `cosm` for $N = 12, 16, 20$, and it is more accurate than `funm` for all considered orders.
- In the performed numerical tests, the optimal order of Hermite algorithm is $N = 16$.

3.3.5. Conclusions.

In this work an efficient algorithm to compute the matrix cosine based on Hermite matrix polynomial expansions has been proposed, improving the algorithms proposed by the authors in [DJ98, DSIR09]. The new algorithm uses a scaling technique based on the double angle formula, the Horner and Paterson-Stockmeyer's method for computing the Hermite matrix polynomial approximation, a new bound of the absolute error in exact arithmetic, and accuracy bound tests for neglecting higher order series terms. A MATLAB implementation of this algorithm has been compared with the built-in MATLAB function `funm` and the MATLAB function `cosm` based on the Padé algorithm given in [HH05]. Numerical tests show that the new algorithm has lower computational cost and higher accuracy than both functions `funm` and `cosm` for several orders of the Hermite approximation, reaching its best performance when Hermite approximation of order $N = 16$ is used.

3.4. Efficient computation of the matrix cosine [3]

Referencia del artículo:
J. Sastre, J. Ibáñez, P. Ruiz and E. Defez
Efficient computation of the matrix cosine
Applied Mathematics and Computation, Volume 219, Issue 14, March 2013, Pages 7575–7585, ISSN 0096-3003, http://dx.doi.org/10.1016/j.amc.2013.01.043

Abstract

Trigonometric matrix functions play a fundamental role in second order differential equation systems. This work presents an algorithm for computing the cosine matrix function based on Taylor series and the cosine double angle formula. It uses a forward absolute error analysis providing sharper bounds than existing methods. The proposed algorithm had lower cost than state-of-the-art algorithms based on Hermite matrix polynomial series and Padé approximants with higher accuracy in the majority of test matrices.

3.4.1. Introduction

Many engineering processes are described by second order differential equations, whose exact solution is given in terms of trigonometric matrix functions sine and cosine. For example, the wave problem

$$v^2 \frac{\partial^2 \psi}{\partial x^2} = \frac{\partial^2 \psi}{\partial t^2} \ , \tag{3.43}$$

plays an important role in many areas of engineering and applied sciences. If the spatially semi-discretization method is used to solve (3.43), we obtain the matrix differential problem

$$X''(t) + AX(t) = 0 \ , \ X(0) = X_0 \ , \ X'(0) = Y_1 \ , \tag{3.44}$$

where A is a square matrix and Y_0 and Y_1 are vectors. The solution of (3.44) is

$$X(t) = \cos\left(\sqrt{A}t\right)X_0 + \left(\sqrt{A}\right)^{-1} \sin\left(\sqrt{A}t\right)X_1, \tag{3.45}$$

where $\sqrt{A}$ denotes any square root of a non-singular matrix A [Hig08, p. 36]. More general problems of type (3.44), with a forcing term $F(t)$ on the right-hand side arise from mechanical systems without damping, and their solutions can be expressed in terms of integrals involving the matrix sine and cosine [Ser79].

The most competitive algorithms for computing matrix cosine are based on Padé approximations [SB80, HS03, HH05], and recently on Hermite matrix polynomial series [DSIR13, DSIR11], using scaling of matrix A by a power of two, i.e. $A/2^s$ with a nonnegative integer parameter s, and the double angle formula

$$\cos 2A = 2\cos^2 A - I. \tag{3.46}$$

[3]This work has been supported by *Universitat Politècnica de València* grant PAID-06-011-2020.

Matrix sine can be computed using formula $\sin(A) = \cos\left(A - \frac{\pi}{2}I\right)$ and an algorithm to compute both cosine and sine with a lower cost than computing them separately has been proposed in [Hig08, Algorithm 12.8].

In this work we present a competitive scaling algorithm for the computation of matrix cosine based on Taylor series. It uses matrix scaling based on sharp absolute forward error bounds of the types given in [SIDR11a], and Paterson-Stockmeyer's method for the evaluation of Taylor matrix polynomial [PS73]. A MATLAB implementation of this algorithm is made available online and it is compared with MATLAB function `cosher` based on Hermite series [DSIR13, DSIR11], and MATLAB function `cosm` implementing the Padé algorithm from [HH05].

Throughout this paper $\mathbb{C}^{n\times n}$ denotes the set of complex matrices of size $n\times n$, I denotes the identity matrix for this set, $\rho(A)$ is the spectral radius of matrix A, and $\mathbb{N}$ denotes the set of positive integers. The matrix norm $\|\cdot\|$ denotes any subordinate matrix norm, in particular $\|\cdot\|_1$ is the 1-norm. This paper is organized as follows. Section 3.4.2 summarizes some existing results for efficient matrix polynomial evaluation based on Paterson-Stockmeyer's method [PS73]. Section 3.4.3 presents a general Taylor algorithm for computing matrix cosine. Section 3.4.4 deals with the error analysis in exact arithmetic. Section 3.4.5 describes the new scaling algorithm. Section 3.4.6 provides a rounding error analysis. Section 3.4.7 deals with numerical tests, and Section 3.4.8 gives the conclusions.

3.4.2. Matrix polynomial computation by Paterson-Stockmeyer's method

From [SIDR11b, p. 6454-6455] matrix polynomial Taylor approximation $P_m(B) = \sum_{i=0}^{m} p_i B^i$, $B \in \mathbb{C}^{n\times n}$ can be computed optimally for m in the set

$$\mathbb{M} = \{1, 2, 4, 6, 9, 12, 16, 20, 25, 30, \ldots\}, \tag{3.47}$$

where we denote the elements of $\mathbb{M}$ as m_0, m_1, $m_2, \ldots$, respectively, by using Paterson-Stockmeyer's method [PS73], see [Hig08, p. 72–74] for a complete description. First, matrix powers B^2, $B^3, \cdots, B^q$ are computed, where $q = \lceil\sqrt{m_k}\rceil$ or $\lfloor\sqrt{m_k}\rfloor$, both values dividing m_k and giving the same cost [Hig08, p. 74]. Then, evaluation formula (23) from [SIDR11b, p. 6455] is computed

$$\begin{aligned} P_{m_k}(B) = &\left(\left(\cdots\left(B^q p_{m_k} + B^{q-1}p_{m_k-1} + \cdots + Bp_{m_k-q+1} + Ip_{m_k-q}\right)\right.\right.\\ &\times B^q + B^{q-1}p_{m_k-q-1} + B^{q-2}p_{m_k-q-2} + \cdots + Bp_{m_k-2q+1} + Ip_{m_k-2q}\Big)\\ &\times B^q + B^{q-1}p_{m_k-2q-1} + B^{q-2}p_{m_k-2q-2} + \cdots + Bp_{m_k-3q+1} + Ip_{m_k-3q}\Big)\\ &\cdots\\ &\times B^q + B^{q-1}p_{q-1} + B^{q-2}p_{q-2} + \cdots + Bp_1 + Ip_0. \end{aligned} \tag{3.48}$$

Taking into account Table 4.1 from [Hig08, p. 74], the cost of evaluating $P_{m_k}(B)$ in terms of matrix products, denoted by Π_{m_k}, for $k = 0, 1, \ldots$ is

$$\Pi_{m_k} = k. \tag{3.49}$$

3.4.3. General Algorithm

Taylor approximation of order $2m$ of cosine of matrix $A \in \mathbb{C}^{n\times n}$ can be expressed as the polynomial of order m

$$P_m(B) = \sum_{i=0}^{m} p_i B^i, \tag{3.50}$$

where $p_i = \frac{(-1)^i}{(2i)!}$ and $B = A^2$. Since Taylor series is accurate only near the origin, in algorithms that use this approximation the norm of matrix A is reduced using techniques based on the double angle formula (3.46), similar to those employed in the scaling an squaring method for computing the matrix exponential [ML03]. Algorithm 3.3 `costay` computes the matrix cosine based on these ideas, considering the values of $m_k \in \mathbb{M}$ for the truncated Taylor series (3.50), with a maximum allowed value of m_k equal to m_M.

Algorithm 3.3 `costay`: Given a matrix $A \in \mathbb{C}^{n\times n}$ and a maximum order $m_M \in \mathbb{M}$, this algorithm computes $C = \cos(A)$ by a Taylor approximation of order $2m_k \leqslant 2m_M$ and the double angle formula (3.46).

1: Preprocessing of matrix A.
2: $B = A^2$ ▷ The memory for A is reused for B
3: SCALING PHASE: Choose $m_k \leqslant m_M$, $m_k \in \mathbb{M}$, and an adequate scaling parameter $s \in \mathbb{N} \cup \{0\}$ for the Taylor approximation with scaling.
4: Compute $C = P_{m_k}(B/4^s)$ using (3.48)
5: **for** $i = 1 : s$ **do**
6: $\quad C = 2C^2 - I$
7: **end for**
8: Postprocessing of matrix C.

The preprocessing and postprocessing are based on applying transformations to reduce the norm of matrix A and recover the matrix $C \cong \cos(A)$ from the result of Loop 5-7. The available techniques to reduce the norm of a matrix are argument translation and balancing [Hig08, p. 299]. The argument translation is based on the formula

$$\cos(A - \pi j I) = (-1)^j \cos(A),\ k \in \mathbb{Z},$$

and on finding the integer j such that the norm of matrix $A - \pi j I$ is minimum. This value can be calculated by using Theorem 4.18 from [Hig08]. Balancing is a heuristic that attempts to equalize the norms of the kth row and kth column, for each k, by a diagonal similarity transformation defined by a non singular matrix D. Balancing tends to reduce the norm, though this is not guaranteed, so it should be used only for matrices where the norm is really reduced. For those matrices, if $A = D^{-1}(A - \pi j I)D$ is the obtained matrix in the preprocessing step, the postprocessing consists of computing $(-1)^j DCD^{-1}$, where C is the matrix obtained after Loop $5 - 7$.

We consider as input argument the maximum Taylor order m_M that can be used for computing the matrix cosine. In the SCALING PHASE the optimal order of Taylor approximation $m_k \leqslant m_M$ and the scaling parameter s are chosen. Analogously to [SIDR11a], in the

proposed scaling algorithm it will be necessary that the same powers of B are used in (3.48) for the two last orders m_{M-1} and m_M, i.e. $B^i, i = 2, 3, \ldots, q$. For each value of m_M Table 3.1 shows the selected optimal values of q for orders m_k, $k = 0, 1, 2, \ldots, M$, denoted by q_k. For example, if $m_M = 20$ and $m_4 = 9$ is the optimal order obtained in the SCALING PHASE, then $q_4 = 3$.

For the evaluation of P_{m_k} in Step 4 the Paterson-Stockmeyer's method described in Section 3.4.2 is applied. Then, the double angle formula is used to obtain $\cos(A)$ in Steps $5 - 7$ from matrix C of Step 4. Thus, using (3.49) it follows that computational cost of Algorithm `costay` in terms of matrix products is

$$\text{Cost}(m_k, s) = 1 + k + s. \tag{3.51}$$

Tabla 3.1: Values of q_k depending on the selection of m_M.

k	0	1	2	3	4	5	6	7
$m_M \setminus m_k$	1	2	4	6	9	12	16	20
12	1	2	2	3	3	3		
16	1	2	2	3	3	4	4	
20	1	2	2	3	3	4	4	4

3.4.4. Error analysis in exact arithmetic and practical considerations

Using (3.50), it follows that

$$E_{m_k,s} = \|\cos(A/2^s) - P_{m_k}(B/4^s)\| = \left\| \sum_{i=m_k+1}^{\infty} \frac{(-1)^i B^i}{(2i)!4^{si}} \right\| \tag{3.52}$$

represents the forward absolute error in exact arithmetic from the approximation of cosine matrix function of the scaled matrix $A/2^s$ by Taylor series truncation. Analogously to [HH05], for the evaluation of $\cos(A)$ in IEEE double precision arithmetic we consider an absolute error-based algorithm, by selecting the appropriate values of m_k and s such that

$$E_{m_k,s} \leqslant u, \tag{3.53}$$

where $u = 2^{-53}$ is the unit roundoff in IEEE double precision arithmetic, providing high accuracy with minimal computational cost. The application of the proposed scaling algorithm to IEEE single precision arithmetic is straightforward and it will not be included in this paper.

In order to bound the norm of the matrix power series in (3.52), we will use the following improved version of Theorem 1 from [SIDR11a, p. 1835]:

Theorem 9 *Let $h_l(x) = \sum_{i=l}^{\infty} p_i x^i$ be a power series with radius of convergence w, $\tilde{h}_l(x) = \sum_{i=l}^{\infty} |p_i| x^i$, $B \in \mathbb{C}^{n\times n}$ with $\rho(B) < w$, $l \in \mathbb{N}$ and $t \in \mathbb{N}$ with $1 \leqslant t \leqslant l$. If t_0 is the multiple of t*

such that $l \leqslant t_0 \leqslant l + t - 1$ *and*

$$\beta_t = \max\{b_j^{1/j} : j = t, l, l+1, \ldots, t_0 - 1, t_0 + 1, t_0 + 2, \ldots, l + t - 1\}, \tag{3.54}$$

where b_j *is an upper bound for* $\|B^j\|$, $\|B^j\| \leqslant b_j$, *then*

$$\|h_l(B)\| \leqslant \tilde{h}_l(\beta_t). \tag{3.55}$$

Proof. Note that $\|B^{t_0}\|^{1/t_0} \leqslant (\|B^t\|^{t_0/t})^{1/t_0} = \|B^t\|^{1/t}$, and then, if we denote

$$\alpha_t = \max\left\{\left\|B^j\right\|^{\frac{1}{j}} : j = t, l, l+1, \cdots, l+t-1\right\}, \tag{3.56}$$

it follows that

$$\begin{aligned} \alpha_t &= \max\left\{\left\|B^j\right\|^{\frac{1}{j}} : j = t, l, l+1, \ldots, t_0 - 1, t_0 + 1, t_0 + 2, \ldots, l + t - 1\right\} \\ &\leqslant \beta_t. \end{aligned} \tag{3.57}$$

Hence

$$\begin{aligned} \|h_l(B)\| &\leqslant \sum_{j\geqslant 0}\sum_{i=l}^{l+t-1} |p_{i+jt}|\,\|B^t\|^j\|B^i\| \leqslant \sum_{j\geqslant 0}\sum_{i=l}^{l+t-1} |p_{i+jt}|\alpha_t^{i+jt} \\ &\leqslant \sum_{i\geqslant l} |p_i|\beta_t^i = \tilde{h}_l(\beta_t). \qquad \square \end{aligned} \tag{3.58}$$

Theorem 9 simplifies Theorem 1 from [SIDR11a, p. 1835] avoiding the need for the bound b_{t_0} for $\|B^{t_0}\|$ to obtain β_t, see (3.54).

Similarly to [SIDR11a] we use three types of bounds for the absolute error. Using (3.52) and Theorem 9 it follows that

$$E_{m_k,s} \leqslant \sum_{i=m_k+1}^{\infty} \frac{(\beta_t^{(m_k)}/4^s)^i}{(2i)!}, \tag{3.59}$$

$$E_{m_k,s} \leqslant \sum_{i=m_k+1}^{\infty} \frac{\left\|(B/4^s)^i\right\|}{(2i)!}, \tag{3.60}$$

$$E_{m_k,s} \leqslant \left\|(B/4^s)^{m_k+1}\right\| \left\|\sum_{i=0}^{q_k} \frac{(-1)^{i+m_k+1}(B/4^s)^i}{(2(i+m_k+1))!}\right\| + \sum_{i=m_k+q_k+2}^{\infty} \frac{\left\|(B/4^s)^i\right\|}{(2i)!}, \tag{3.61}$$

where $m_k \in \mathbb{M}$ and $\beta_t^{(m_k)}$, $l = m_k + 1$, $1 \leqslant t \leqslant l$, are the values given in (3.54) from Theorem 9. The superscript on $\beta_t^{(m_k)}$ remarks the dependency on the order m_k through the value of l. In order to compute (3.54), bounds b_j for the norms of matrix powers $\|B^j\|$ are needed. Analogously to [SIDR11a], first, $\left\|B^{m_k+1}\right\|_1$ will be estimated using the block 1-norm estimation algorithm of [HT00], taking $b_{m_k+1} = \left\|B^{m_k+1}\right\|_1$. For a $n \times n$ matrix, this algorithm carries out a 1-norm power iteration whose iterates are $n \times r$ matrices, where r is a parameter that has been taken to be 2, see [AMH09, p. 983]. Hence, the estimation algorithm has $O(n^2)$ computational cost, negligible compared with a matrix product, whose cost is $O(n^3)$. Then, we compute bounds b_j for the rest of needed matrix power 1-norms involved in (3.54), (3.60) and (3.61) using products of the estimated 1-norm of matrix powers, and the

norms of the matrix powers needed for the computation of P_{m_k}, taking for them $b_j = \|B^j\|_1$, $j = 1, 2, \ldots, q_k$, see Section 3.4.2. Thus, if $b_{e_k} = \|B^{e_k}\|_1$, $k = 1, 2, \ldots, L$, are all the known norms of matrix powers we obtain the remaining bounds of norms of matrix powers using

$$\left\|B^j\right\|_1 \leqslant b_j = \min\left\{b_{e_1}^{i_1} \cdot b_{e_2}^{i_2} \cdots b_{e_L}^{i_L} \,:\, e_1 i_1 + e_2 i_2 + \cdots + e_L i_L = j\right\}. \tag{3.62}$$

Note that the minimum in (3.62) is desirable but not necessary. A simple Matlab function has been provided to obtain b_j, see nested function `powerbound` from `costay.m` available at [Cos], analogous to nested function `powerbound` in `exptayns.m` from [SIDR11a].

Then, the values $\beta_t^{(m_k)}$ from (3.59) can be obtained using bounds b_j in (3.54) with $l = m_k + 1$. Taking into account (3.59), let

$$\Theta_{m_k} = \max\left\{\theta : \sum_{i=m_k+1}^{\infty} \frac{\theta^i}{(2i)!} \leqslant u\right\}. \tag{3.63}$$

To compute Θ_{m_k}, we follow Higham in [Hig05] using the MATLAB Symbolic Math Toolbox to evaluate $\sum_{i=m_k+1}^{\infty} \frac{\theta^i}{(2i)!}$ in 250-digit decimal arithmetic for each m_k, summing the first 200 terms with the coefficients obtained symbolically. Then, a numerical zero-finder is invoked to determine the highest value of Θ_{m_k} such that $\sum_{i=m_k+1}^{\infty} \frac{\theta^i}{(2i)!} \leqslant u$ holds. Table 3.2 shows the values obtained for the first ten values of $m_k \in \mathbb{M}$. Using (3.59) and (3.63), if $\beta_t^{(m_k)} \leqslant \Theta_{m_k}$ for two given values of m_k and t, then $E_{m_k,0} \leqslant u$, and $s = 0$ can be used with order m_k. Otherwise, for using order m_k the appropriate minimum scaling parameter $s > 0$ such that $\beta_t^{(m_k)}/4^s \leqslant \Theta_{m_k}$ and $E_{m_k,s} \leqslant u$ should be taken.

Taking into account (3.49) it follows that $\Pi_{m_{k+1}} = \Pi_{m_k} + 1$, but this is offset by the larger allowed value of $\theta = \beta_t^{(m_{k+1})}/4^s$ if $\Theta_{m_{k+1}} > 4\Theta_{m_k}$, since decreasing s by 1 saves one matrix multiplication in the application of the double angle formula in Steps 5 – 7 of `costay`. Table 3.2 shows that $\Theta_{m_{k+1}} > 4\Theta_{m_k}$ for $k \leqslant 3$. Therefore, taking into account (3.59) and (3.63) it follows that selecting $m_M < m_4 = 9$ as maximum order is not an efficient choice.

On the other hand, Table 3.2 shows that $\Theta_{m_{k+1}}/4 < \Theta_{m_k}$ for $k \geqslant 4$. Therefore, for $m_M \geqslant m_5 = 12$, if the following expression holds for certain values of $s \geqslant 0$ and t_1, $1 \leqslant t_1 \leqslant m_M + 1$

$$\Theta_{m_M}/4 \lesssim \beta_{t_1}^{(m_M)}/4^s \leqslant \Theta_{m_M}, \tag{3.64}$$

then, for those matrices where next expression also holds

$$\Theta_{m_M}/4 \leqslant \beta_{t_2}^{(m_{M-1})}/4^s \leqslant \Theta_{m_{M-1}}, \tag{3.65}$$

for certain value of t_2, $1 \leqslant t_2 \leqslant m_{M-1} + 1$, one can select s with m_{M-1} instead of m_M saving one matrix product, see (3.51). Therefore, if $m_M \geqslant 12$ the proposed scaling algorithm will consider both orders m_{M-1} and m_M, selecting the one that provides the lowest cost.

Tabla 3.2: Highest values Θ_{m_k} such that $\sum_{i=m_k+1}^{\infty} \frac{\theta^i}{(2i)!} \leqslant u$.

k	m_k	Θ_{m_k}	k	m_k	Θ_{m_k}
0	1	5.161913651490293e−8	5	12	6.592007689102032
1	2	4.307719974921524e−5	6	16	2.108701860627005e1
2	4	1.321374609245925e−2	7	20	4.735200196725911e1
3	6	1.921492462995386e−1	8	25	9.944132963297543e1
4	9	1.749801512963547	9	30	1.748690782129054e2

Bounds (3.60) and (3.61) are used to refine the results obtained with (3.59). In [SIDR11a] it is shown that using the 1-norm a bound of type (3.60) can be lower or higher than a bound of type (3.61) for normal and also for nonnormal matrices, depending on the specific matrix, see (20) [SIDR11a, p. 1839]. Therefore, both bounds are considered.

To approximate bounds (3.60) and (3.61) we use the matrix power 1-norm $b_{m_k+1} = \left\|B^{m_k+1}\right\|_1$ estimated previously, and the bounds b_j for matrix powers obtained from (3.62). Taking into account that B^i takes the values $I, B, \ldots, B^{q_k}$ in the first summation of (3.61), this summation can be evaluated with a cost $O(n^2)$ reusing the matrix powers needed to compute P_{m_k} from (3.50), see Section 3.4.2. Following [SIDR11a], we will truncate adequately the infinite series of (3.60) and (3.61) determining in Section 3.4.5 the minimum number of terms needed to introduce negligible errors.

3.4.5. Scaling algorithm

Scaling Phase of Algorithm 3.3 first tests if any of the orders $m_k < m_{M-1} \in \mathbb{M}$, $m_M \geqslant 12$, verifies (3.53) with $s = 0$, using the error bounds described in Section 3.4.4. If no order $m_k < m_{M-1}$ verifies (3.53) with $s = 0$, the algorithm will use Theorem 9, (3.59), (3.63) and values Θ_{m_k} from Table 3.2, to obtain initial values of scaling parameter for m_{M-1} and m_M, denoted by $s_0^{(M-1)}$ and $s_0^{(M)}$, respectively. If $s_0^{(M-1)} > 0$ or $s_0^{(M)} > 0$, the algorithm will verify if $s_0^{(M-1)}$ or $s_0^{(M)}$ can be reduced using bounds (3.60) and (3.61), selecting finally the combination of order and scaling parameter that gives the lowest cost in Steps 4 – 7 from `costay`, see (3.51). Next we describe the proposed algorithm.

First we test if $m_0 = 1$ can be used with $s = 0$. Note that order m_0 is not very likely to be used, given (3.59), (3.63) and the value of Θ_1 in Table 3.2. Then we do not waste work estimating $\|B^2\|_1$, and, using Theorem 9 with $l = m_0 + 1$, (3.59) and (3.63), order m_0 will be selected only if

$$\beta_1^{(1)} = \max\{\|B\|, \|B^2\|^{1/2}\} = \|B\| \leqslant \Theta_1, \tag{3.66}$$

taking in practice

$$\beta_1^{(1)} = \min\{\|B\|_1, \|B\|_\infty\}. \tag{3.67}$$

For order $m_1 = 2$, taking into account Table 3.1, it follows that $q_1 = 2$ and then B^2 is computed. Analogously this order is not very likely to be used. Then, taking $b_2 = \|B^2\|$ and

$b_3 = \|B^2\|\,\|B\|$ and using Theorem 9 with $l = m_1 + 1$, from (3.59) and (3.63) we will only select m_1 if

$$\beta_2^{(2)} = \max\{b_2^{1/2}, b_3^{1/3}\} = b_3^{1/3} = (\|B^2\|\,\|B\|)^{1/3} \leqslant \Theta_2, \tag{3.68}$$

taking in practice

$$\beta_2^{(2)} = (\min\{\|B^2\|_1\,\|B\|_1, \|B^2\|_\infty \|B\|_\infty\})^{1/3}. \tag{3.69}$$

Orders $m_k \geqslant m_2 = 4$ are more likely to be used given that $\Theta_{m_k} \geqslant \Theta_4 \approx 0.013$, see Table 3.2. Then, we will test successively each value $m_k \in \mathbb{M}$, from $m_2 = 4$ to m_M in increasing order. Whenever the corresponding value of q_k for the current value of m_k increases, the corresponding matrix power B^{q_k} will be computed and used for evaluating the different error bounds. In Subsection 3.4.5.1 we use Theorem 9 and (3.59) to obtain an initial value of the scaling parameter for m_k, denoted by $s_0^{(k)}$. In Subsection 3.4.5.2 we use (3.60) and (3.61) to verify if $s_0^{(k)}$ can be reduced when $s_0^{(k)} > 0$. The 1-norm will be used for all norms in both Subsections. The complete algorithm is given finally as Algorithm 3.4.

3.4.5.1. Initial value of the scaling parameter

For each value of $m_k \in \mathbb{M}$, $4 \leqslant m_k \leqslant m_M$, we search for the minimum value of $\beta_t^{(m_k)}$ values of (3.54) from Theorem 9 with $l = m_k + 1$. For that task we estimate $b_{m_k+1} = \|B^{m_k+1}\|_1$ as explained in Section 3.4.4, and use the estimated 1-norms of powers of B that have been computed to test orders $m_2, m_3, \ldots, m_{k-1}$, i.e. $\|B^{m_2+1}\|_1, \|B^{m_3+1}\|_1, \cdots, \|B^{m_{k-1}+1}\|_1$, the 1-norms of the powers of B used for the computation of P_{m_k}, i.e. B, $B^2, \ldots, B^{q_k}$, and bounds b_j obtained from (3.62). Thus, for $t = 2, 3, \ldots, q_k, m_2 + 1, m_3 + 1, \cdots, m_k + 1$ we will obtain successively

$$\beta_t^{(m_k)} = \max\{b_j^{1/j} : j = t, m_k + 1, m_k + 2, \ldots, t_0 - 1, t_0 + 1, t_0 + 2, \ldots, m_k + t\}, \tag{3.70}$$

where t_0 is the multiple of t in $[m_k + 1, m_k + t]$, stopping the process for the lowest value $t = r$ such that, see (3.54),

$$b_r{}^{1/r} \leq \max\left\{b_j^{1/j} : j = m_k + 1, m_k + 2, \ldots, r_0 - 1, r_0 + 1, \ldots, m_k + r\right\}, \tag{3.71}$$

where r_0 is the multiple of $r \in \mathbb{N}$ such that $m_k + 1 \leqslant r_0 \leqslant m_k + r$. Next we show that if (3.71) is verified then it follows that $\beta_i^{(m_k)} \geqslant \beta_r^{(m_k)}$ for $i \geqslant r$. Note that by (3.62) one gets $b_{r_0} \leqslant b_r^{r_0/r}$, and then it follows that

$$b_{r_0}^{1/r_0} \leqslant b_r^{1/r} \leqslant \max\{b_j^{1/j} : j = m_k + 1, m_k + 2, \ldots, r_0 - 1, r_0 + 1, \ldots, m_k + r\}. \tag{3.72}$$

Thus, substituting t by r and t_0 by r_0 in (3.70), and using (3.72) it follows that

$$\begin{aligned} \beta_r^{(m_k)} &= \max\{b_j^{1/j} : j = m_k + 1, m_k + 2, \ldots, r_0 - 1, r_0 + 1, \ldots, m_k + r\} \\ &= \max\{b_j^{1/j} : j = m_k + 1, m_k + 2, \ldots, m_k + r\}. \end{aligned} \tag{3.73}$$

Hence, for $i > r$, if

$$b_i^{1/i} \leqslant \max\{b_j^{1/j} : j = m_k + 1, m_k + 2, \ldots, i_0 - 1, i_0 + 1, \ldots, m_k + i\}, \tag{3.74}$$

where i_0 is the multiple of i in $[m_k+1, m_k+i]$, then in a similar way and using (3.73) it follows that

$$\begin{aligned}\beta_i^{(m_k)} &= \max\{b_j^{1/j} : j = m_k+1, m_k+2, \ldots, m_k+i\} \\ &\geqslant \max\{b_j^{1/j} : j = m_k+1, m_k+2, \ldots, m_k+r\} = \beta_r^{(m_k)}. \end{aligned} \tag{3.75}$$

Otherwise, if (3.74) is not verified, since $i > r$ and by (3.62) one gets $b_i^{1/i} \geqslant b_{i_0}^{1/i_0}$, then it follows that

$$\begin{aligned}\beta_i^{(m_k)} &= b_i^{1/i} > \max\{b_j^{1/j} : j = m_k+1, m_k+2, \ldots, m_k+i\} \\ &\geqslant \max\{b_j^{1/j} : j = m_k+1, m_k+2, \ldots, m_k+r\} = \beta_r^{(m_k)}, \end{aligned} \tag{3.76}$$

and then, using (3.75) and (3.76), it follows that $\beta_i^{(m_k)} \geqslant \beta_r^{(m_k)}$ for $i > r$.

Let $\beta_{min}^{(m_k)}$ be the minimum value of all computed values $\beta_t^{(m_k)}$, $1 \leqslant t \leqslant r$,

$$\beta_{min}^{(m_k)} = \min\{\beta_t^{(m_k)},\ t = 2, 3, \ldots, q_k, m_2+1, m_3+1, \cdots, r\}. \tag{3.77}$$

Then the appropriate initial minimum scaling parameter $s_0^{(k)} \geqslant 0$ is obtained such that $4^{-s_0^{(k)}} \beta_{min}^{(m_k)} \leqslant \Theta_{m_k}$, i.e.

$$s_0^{(k)} = \max\left\{0, \lceil 1/2 \log_2(\beta_{min}^{(m_M)}/\Theta_{m_k})\rceil\right\}. \tag{3.78}$$

Table 3.3 shows the order and initial scaling selection depending on the values of $\beta_1^{(1)}$, $\beta_2^{(2)}$ and $\beta_{min}^{(m)}$, $m = 4, 6, \ldots, m_M$, taking into account (3.64) and (3.65) to use the most efficient choice between m_M and m_{M-1} in each case represented in the last four rows of the table, where i is a nonnegative integer parameter. If $s_0^{(k)} = 0$ then the proposed scaling algorithm selects m_k and $s = 0$ for Steps 4-7 of Algorithm `costay`. Otherwise the refinement proposed in next subsection is carried out.

3.4.5.2. Refinement of the scaling parameter

If $s_0^{(k)} > 0$ for the current value of m_k, taking into account (3.61), bounds from (3.62) are used to verify if the following inequality holds:

$$\sum_{i=m_k+1}^{m_k+N} \frac{b_i}{c_i 4^{si}} \leqslant u, \tag{3.79}$$

where $c_i = (2i)!$, and $s = 0$ if $m_k < m_{M-1}$, and $s = s^{(k)} = s_0^{(k)} - 1 \geqslant 0$ if $m_k \geqslant m_{M-1}$. For testing (3.79), we have truncated the series in (3.61) by choosing $N \geqslant q_k + 2$ terms as this number of terms will be also used when computing (3.61), see (3.80). At the end of this subsection we provide justification for the number of terms N to select for a negligible truncation error. We stop the series summation in (3.79) if after summing one term the sum is greater than u. If the sum of one or more terms is lower than u but the complete truncated series sum is not, we can estimate $b_{m_k+2} = \|B^{m_k+2}\|_1$ to verify if (3.79) holds. If (3.79) holds, using (3.60) it follows that the forward absolute error $E_{m_k,s}$ is approximately lower than or equal to u.

Tabla 3.3: Selection of initial scaling $s_0^{(k)}$ and order $m \in \mathbb{M}$, $m \leqslant m_M$ using only the first part of the proposed scaling algorithm, described in Subsection 3.4.5.1, depending on the values of $\beta_{min}^{(m)}$, for $m_M = 9, 12, 16, 20$. Total cost, denoted by C_m^T, is also presented. The cost with $m_M = 12$ and 9 is the same and it is presented in one column. $\beta_{min}^{(1)}$ and $\beta_{min}^{(2)}$ are not calculated for first orders $m = 1$ and 2, using $\beta_1^{(1)}$ and $\beta_2^{(2)}$ instead. The tests are done from top to bottom: If the condition for current row is not verified then we test the condition for the next row. In last four rows i can take the values $i = 0, 1, \ldots$

m_M	9	12	9,12	16	20
interval	s_0, m	s_0, m	C_m^T	s_0, m, C_m^T	s_0, m, C_m^T
$\beta_1^{(1)} \leqslant \Theta_1$	0,1	0,1	0	0,1,0	0,1,0
$\beta_2^{(2)} \leqslant \Theta_2$	0,2	0,2	1	0,2,1	0,2,1
$\beta_{min}^{(m)} \leqslant \Theta_4$	0,4	0,4	2	0,4,2	0,4,2
$\beta_{min}^{(m)} \leqslant \Theta_6$	0,6	0,6	3	0,6,3	0,6,3
$\beta_{min}^{(m)} \leqslant \Theta_9$	0,9	0,9	4	0,9,4	0,9,4
$\beta_{min}^{(m)} \leqslant \Theta_{12}$	1,9	0,12	5	0,12,5	0,12,5
$\beta_{min}^{(m)} \leqslant 4\Theta_9$	1,9	1,9	5	0,16,6	0,16,6
$\beta_{min}^{(m)} \leqslant \Theta_{16}$	2,9	1,12	6	0,16,6	0,16,6
$\beta_{min}^{(m)} \leqslant 4^{i+1}\Theta_{12}$	2+i,9	1+i,12	6+i	1+i,12,6+i	i,20,7+i
$\beta_{min}^{(m)} \leqslant 4^{i+2}\Theta_9$	2+i,9	2+i,9	6+i	1+i,16,7+i	i,20,7+i
$\beta_{min}^{(m)} \leqslant 4^{i}\Theta_{20}$	3+i,9	2+i,12	7+i	1+i,16,7+i	i,20,7+i
$\beta_{min}^{(m)} \leqslant 4^{i+1}\Theta_{16}$	3+i,9	2+i,12	7+i	1+i,16,7+i	1+i,16,7+i

If (3.79) does not hold, then, from (3.61) we verify if next bound holds

$$\frac{\left\|B^{m_k+1}\right\|_1}{4^{s(m_k+2)}} \left\| \sum_{i=0}^{q_k} \frac{c_{m_k+2}}{c_{i+m_k+1}} \frac{(-1)^i B^i}{4^{s(i-1)}} \right\|_1 + \sum_{i=m_k+q_k+2}^{m_k+N} \frac{c_{m_k+2}}{c_i} \frac{b_i}{4^{si}} \leqslant u c_{m_k+2}, \tag{3.80}$$

where $s = 0$ if $m_k < m_{M-1}$, and $s = s^{(k)} = s_0^{(k)} - 1 \geqslant 0$ if $m_k \geqslant m_{M-1}$, and we truncate the series in (3.80) taking $N \geqslant q_k + 2$. We multiply both sides of (3.61) by c_{m_k+2} to save the product of matrix B by a scalar. If the first term of the left-hand side of (3.80) is lower than uc_{m_k+2} but the sum of the two terms is not, we can estimate $b_{m_k+q_k+2} = \|B^{m_k+q_k+2}\|_1$ to verify if (3.80) holds then.

Next, we obtain lower bounds for expression (3.80) to avoid its computation in some cases, see (16)-(18) from [SIDR11a, p. 1838]. Let

$$T_i^{(m_k)} = \frac{c_{m_k+2} \left\|B^i\right\|}{c_{i+m_k+1} 4^{s(i-1)}}, i = 0 : q_k,$$

and

$$T_j^{(m_k)} = \max\left\{T_i^{(m_k)}, i = 0 : q_k\right\},$$

be. Since

$$\left\| \sum_{i=0}^{q_k} \frac{c_{m_k+2}}{c_{i+m_k+1}} \frac{(-1)^i B^i}{4^{s(i-1)}} \right\| \geqslant T_s^{(m_k)},$$

where

$$T_s^{(m_k)} = \max\left\{0, T_j^{(m_k)} - \sum_{i\neq j} T_i^{(m_k)}\right\},$$

then if

$$\frac{\left\|B^{m_k+1}\right\|}{4^{s(m_k+2)}} T_s^{(m_k)} + \sum_{i=m_k+q_k+2}^{i=m_k+N} \frac{c_{m_k+2} b_i}{c_i 4^{si}} > u c_{m_k+2}, \tag{3.81}$$

then there is no need to test (3.80).

For the case where $m_k < m_{M-1}$ if (3.79) or (3.80) hold with $s = 0$, the scaling algorithm selects m_k and $s = 0$ for Steps $4-7$ of `costay`. Otherwise, the two stages of the scaling algorithm from previous and this subsections are repeated with the next value of $m_k \in \mathbb{M}$ until $m_k = m_{M-1}$.

For $m_k \geqslant m_{M-1}$, firstly, an initial value $s_0^{(k)}$ of the scaling parameter is obtained as explained in Subsection 3.4.5.1, see (3.78). If $s_0^{(k)} = 0$, then m_k and $s^{(k)} = 0$ are selected. Otherwise, (3.79) and (3.80) are tested with $s^{(k)} = s_0^{(k)} - 1$. If none of both bounds hold, then we take $s^{(k)} = s_0^{(k)}$, and finally the order m_{M-1} or m_M that provides the lowest cost is selected for Steps $4-7$ of `costay`. It is possible to evaluate $P_m(4^{-s}B)$ with optimal cost for both orders because we set in its evaluation that both use the same powers of B, see Section 3.4.3.

Note that the cost of evaluating (3.79) and (3.80) is $O(n^2)$ and if any of them is verified with $s^{(k)} < s_0^{(k)}$ matrix products are saved, whose cost is $O(n^3)$.

Finally, we provide justification for the truncation of the series in (3.79) and (3.80). From (3.59), (3.77) and (3.78) it follows that the remainder of the infinite series without the first N terms, denoted by $R_{m_k+N+1,s}(B)$, verifies

$$\|R_{m_k+N+1,s}(B)\| \leqslant \sum_{i=m_k+N+1}^{\infty} \frac{\left\|B^i\right\|}{(2i)!4^{si}} \leqslant \sum_{i=m_k+N+1}^{\infty} \frac{(\beta_{min}^{(m_k)}/4^s)^i}{(2i)!}. \tag{3.82}$$

If the minimum value of s to be tested is $s = s^{(k)} = s_0^{(k)} - 1$ then by (3.78) it follows that $\beta_{min}^{(m_k)}/4^s \leqslant 4\Theta_{m_k}$. Hence, using (3.82) it follows that

$$\|R_{m_k+N+1,s}(B)\| \leqslant \sum_{i=m_k+N+1}^{\infty} \frac{(4\Theta_{m_k})^i}{(2i)!}. \tag{3.83}$$

Using Matlab Symbolic Math Toolbox and computing the first 150 terms of the series in (3.83) with high precision arithmetic, Table 3.4 presents the values of bound (3.83) for m_k, $k = 2, 3, \ldots, 7$, with the corresponding values of q_k proposed in Section 3.4.3 for each value of $m_k \in \mathbb{M}$, and $N = q_k + 2, q_k + 3, \ldots, q_k + 8$. Since error $E_{m,s}$ must verify $E_{m,s} \leqslant u \approx 1.11 \cdot 10^{-16}$ and rounding errors of values at least nu are expected in the evaluation of P_{m_k} where n is the matrix dimension, see [Hig08], the values of bound (3.83) from Table 3.4 are satisfactory taking $N = q_k + 2$ for $m_k \leqslant 12$, $N = q_k + 4$ for $m_k = 16$, and $N = q_k + 7$ for $m_k = 20$.

However, in numerical tests we have observed that if (3.79) or (3.80) hold with $N = q_k + 2$, usually the value of $\beta_{min}^{(m_k)}$ in (3.82) is nearer to Θ_{m_M} than to $4\Theta_{m_M}$, and bound (3.82) is

usually much lower than bound (3.83), therefore $N = q_k + 2$ being a good selection for all $m \leqslant 20$, see Section 3.4.7.

On the other hand, note that the values obtained in Table 3.4 for (3.83) with $m_k = 4$ and 6 are much lower than the unit roundoff u. Thus, in order to test if we can permit values of the final scaling parameter up to $s = s_0^{(k)} - 2$ for those orders, we have taken $4^2\Theta_{m_k}$ instead of $4\Theta_{m_k}$ in (3.83) with $N = q_k + 2$, resulting $\|R_{4+q_2+2+1,s}(B)\| \leqslant 1.3\text{e}{-22}$ and $\|R_{6+q_3+2+1,s}(B)\| \leqslant 1.2\text{e}{-18}$. Thus, for orders $m_k = 4$ and 6 we can take $s = s^{(k)} \geqslant s_0^{(k)} - 2$ with $N = q_k + 2$. Taking this into account Algorithm 3.4 describes the proposed scaling algorithm. The complete algorithm `costay` has been implemented in MATLAB and made available online in [Cos]. This version of `costay` permits the selection of maximum order m_M from 6 to 30 for testing purposes.

Tabla 3.4: Values of bound (3.83).

	N						
m_k, q_k	q_k+2	q_k+3	q_k+4	q_k+5	q_k+6	q_k+7	q_k+8
4,2	5.0e−28	7.0e−32	8.0e−36	7.7e−40	6.2e−44	4.4e−48	2.6e−52
6,3	6.9e−26	8.1e−29	8.2e−32	7.3e−35	5.6e−38	3.9e−41	2.4e−44
9,3	1.8e−20	1.3e−22	7.9e−25	4.4e−27	2.2e−29	9.8e−32	4.0e−34
12,3	1.0e−16	2.0e−18	3.3e−20	5.0e−22	7.0e−24	8.9e−26	1.0e−27
12,4	2.0e−18	3.3e−20	5.0e−22	7.0e−24	8.9e−26	1.0e−27	1.1e−29
16,4	3.8e−14	1.4e−15	4.8e−17	1.5e−18	4.5e−20	1.2e−21	3.1e−23
20,4	1.4e−10	8.7e−12	5.0e−13	2.7e−14	1.3e−15	6.2e−17	2.7e−18

Algorithm 3.4 SCALING PHASE: Given matrix B from Step 2 of `costay` and maximum order m_M with $12 \leqslant m_M \leqslant 20$, it computes $m \leqslant m_M$, $m \in \mathbb{M}$, q from Table 3.1, and the scaling parameter s to be used in Steps 4 – 7. This algorithm uses the values m_k and q_k from Table 3.1, and Θ_{m_k} from Table 3.2.

1: $b_1 = \|B\|_1$, $d_1 = \|B\|_\infty$
2: **if** $\min\{b_1, d_1\} \leqslant \Theta_1$ **then** ▷ Test $m_0 = 1$
3: **return** $s = 0$, $m = m_0$, $q = 1$
4: **end if**
5: $B_2 = B^2$, $b_2 = \|B_2\|_1$, $d_2 = \|B_2\|_\infty$
6: **if** $\min\{b_2 \cdot b_1, d_2 \cdot d_1\}^{1/3} \leqslant \Theta_2$ **then** ▷ Test $m_1 = 2$
7: **return** $s = 0$, $m = m_1$, $q = 2$
8: **end if**
9: $q = q_2$, $k = 1$
10: **while** $m < m_M$ **do**
11: $k = k + 1$, $m = m_k$
12: **if** $q < q_k$ **then**
13: $q = q_k$
14: **if** $q = 3$ **then**
15: $B_3 = B_2 B$, $b_3 = \|B_3\|_1$
16: **else if** $q = 4$ **then**
17: $B_4 = B_2^2$, $b_4 = \|B_4\|_1$
18: **end if**
19: **end if**

Estimate $b_{m+1} = \|B^{m+1}\|_1$.
Obtain $\beta_t^{(m)}$ from (3.54) with $l = m+1$, $t = 2, 3, \ldots, q, m_2+1, m_3+1, \cdots, r$ where r is the lowest value such that condition (3.71) is verified, computing the needed bounds b_j for $\|B^j\|_1$ using (3.62).
$\beta_{min}^{(m)} = \min\{\beta_t^{(m)}, t = 2, 3, \ldots, q, m_2+1, m_3+1, \cdots, r\}$.
$s^{(k)} = s_0^{(k)} = \max\left\{0, \lceil 1/2 \log_2(\beta_{min}^{(m)}/\Theta_m)\rceil\right\}$.
if $s_0^{(k)} > 0$ **then**
  if ($m \leqslant 6$ AND $s_0^{(k)} \leqslant 2$) OR ($m < m_{M-1}$ AND $s_0^{(k)} = 1$) **then**
    if (3.79) is verified with $s = 0$ **then**
      $s^{(k)} = 0$
    else if (3.81) is not verified with $s = 0$ **then**
      if (3.80) is verified with $s = 0$ **then**
        $s^{(k)} = 0$
      end if
    end if
  else if $m \geqslant m_{M-1}$ **then**
    if (3.79) is verified with $s_0^{(k)} - 1$ **then**
      $s^{(k)} = s_0^{(k)} - 1$
    else if (3.81) is not verified with $s_0^{(k)} - 1$ **then**
      if (3.80) is verified with $s_0^{(k)} - 1$ **then**
        $s^{(k)} = s_0^{(k)} - 1$
      end if
    end if
  end if
end if
if $s^{(k)} = 0$ **then** ▷ If $s^{(k)} = 0$ order m is selected directly
  return $s^{(k)}$, m, q
end if
end while
if $s^{(M-1)} \geqslant s^{(M)} + 1$ **then** ▷ Select the combination with the lowest cost
  return $s = s^{(M)}$, $m = m_M$, $q = q_M$
else
  return $s = s^{(M-1)}$, $m = m_{M-1}$, $q = q_{M-1}$ ▷ $q_{M-1} = q_M$
end if

3.4.6. Rounding error analysis

The analysis of rounding errors in Algorithm 3.3 is based on the results given in [DSIR11] and [Hig08, p. 293-294]. In [DSIR11] it was justified that rounding errors in the evaluation of $P_m(B)$ are balanced with rounding errors in the double angle phase. If we define $C_i = \cos\left(2^{i-s}A\right)$ and $\hat{C}_i = fl\left(2\hat{C}_{i-1}^2 - I\right)$, where fl is the floating operator [GL96, p. 61], and we

assume that $\|E_i\|_1 \leqslant 0.05\|\hat{C}_i\|_1$, then rounding error in Step 6 of Algorithm 3.3 verifies

$$\|E_i\|_1 = \|C_i - \hat{C}_i\|_1 \leqslant (4.1)^i\|E_0\|_1\|C_0\|_1\|C_1\|_1\cdots\|C_{i-1}\|_1 +$$
$$\tilde{\gamma}_{n+1}\sum_{j=0}^{i-1}(4.1)^{i-j-1}\left(2.21\|C_j\|_1^2+1\right)\|C_{j+1}\|_1\cdots\|C_{i-1}\|_1,$$

where $\tilde{\gamma}_{n+1}$ is defined by $\tilde{\gamma}_{n+1} = \frac{c(n+1)u}{1-c(n+1)u}$ [Hig08, p. 332]. Hence the error $\|E_i\|_1$ fundamentally depends on the norms of the matrices $\|C_0\|_1$ and $\|E_0\|_1$. From (3.53), values of m_k and s are chosen such that $\|E_0\|_1 \leqslant u$. Since $\|4^{-s}A^2\|$ is not bounded with the new proposed scaling algorithm, it follows that $\|C_0\|$ is not bounded. Taking into account that the values of Θ_{m_k} increase with m_k, it follows that the values of the scaling parameter s with high values of m_k will be typically lower, giving higher values of $\|4^{-s}A^2\|$, and then $\|C_0\|$ will usually be higher when permitting the use of higher orders. Hence, despite the error balancing between the evaluation of $P_m(B)$ and the double angle phase, orders not much higher than the approximately optimal $m_M = 12$ should be used in the proposed scaling algorithm.

3.4.7. Numerical experiments

In this section we compare MATLAB implementation `costay` with functions `cosm` and `cosher`. `cosm` is a MATLAB implementation of Algorithm 5.1 proposed in [HH05] which uses Padé approximants of cosine function and it is available online at `http://www.maths.manchester.ac.uk/~higham/mftoolbox`. `cosher` is a MATLAB function based on Hermite series proposed in [DSIR13] and available at `http://personales.upv.es/~jorsasma/cosher.m`. Similarly to `costay`, `cosher` also allows different maximum orders m_M for the Hermite approximation, recommending $m_M = 16$ for best performance in numerical tests, see [DSIR13]. All MATLAB implementations (R2011a) were tested on an Intel Core 2 Duo processor at 2.52 GHz with 4 GB main memory. Algorithm accuracy was tested by computing the relative error

$$\mathrm{E} = \frac{\|\cos(A) - \tilde{Y}\|_1}{\|\cos(A)\|_1},$$

where $\tilde{Y}$ is the computed solution and $\cos(A)$ the exact solution. We used 101 of the 102 test matrices from [DSIR11]. Test matrix 61 was removed due to very large rounding errors produced when computing the powers of that matrix, making all the tested algorithms fail. The "exact"matrix cosine was calculated analytically when possible, and otherwise using MATLAB's Symbolic Math Toolbox with high precision arithmetic.

In the tests we did not use any preprocessing/postprocessing in the implemented algorithms. Analogously to the experiments in [Hig05], we found that turning on preprocessing provided similar results to those presented in this section without preprocessing.

Figures 3.8a and 3.8b show the comparisons `costay-cosm` and `costay-cosher` for maximum orders $m_M \in \{9, 12, 16, 20\}$ in both `costay` and `cosher`. The first three rows show the percentages of times that the relative error of the first function is lower, equal or greater than the relative error of the second function. The fourth row shows the ratio of matrix products needed for computing the matrix cosine for over all the test matrices with `costay` divided by the number of matrix products for the compared function. The asymptotic cost

in terms of matrix products for solving the multiple right-hand side linear system appearing in Padé algorithm has been taken 4/3, see [SIDR11a].

As shown in Figures 3.8a and 3.8b, function `costay` presented more accurate results than `cosm` and `cosher` in the significant majority of tests, especially for $m_M = 16$ (in 91.09 % of cases with respect to `cosm` function and 44.55 % with respect to `cosher`). Moreover, `costay` has lower computational costs than the functions `cosm` and `cosher`. Note that the cost gains of using $m_M = 12$ and $m_M = 16$ are the same. Table 3.3 shows that both orders provide the same cost in almost all cases in the first stage of the scaling algorithm providing justification for that. However, $\Theta_{16} > \Theta_{12} > \Theta_9$ and then the values of s with $m_M = 16$ are lower in many cases than those with $m_M = 12$ and $m_M = 9$, see Table 3.3, reducing the number of double angle steps in Algorithm `costay`. Numerical results show that $m_M = 16$ provided the highest accuracy with similar cost to $m_M = 12$, being the best choice for m_M in tests.

We have observed that in a 94.50 % of cases the final value of the scaling parameter is directly $s_0^{(k)}$ given by (3.78), and that bounds (3.79) and (3.80) reduce the scaling parameter in the majority of cases where $\beta_{min}^{(m_k)} 4^{-(s_0^{(k)}-1)}/\Theta_{m_k}$ is slightly greater than 1. There were only two matrices where $\beta_{min}^{(m_k)} 4^{-(s_0^{(k)}-1)}/\Theta_{m_k}$ was not approximately 1, taking values 2.72 with order $m_k = 12$, and 5.86 with $m_k = 4$. With respect to the selection of the number of terms N to be considered in bounds (3.79) and (3.80) we have observed that taking $N = q_k + 2$, the greatest value of the remainder (3.82) using the values of $\beta_{min}^{(m_k)}$ obtained in numerical tests with $m_M = 12, 16, 20$, was $1.3 \cdot 10^{-21}$, thus confirming that the selection $N = q_k + 2$ is enough in practice for all orders $m_k \leqslant 20$.

m_M :	9	12	16	20
L	59.41	83.17	91.09	77.23
E	0	0	0	0
G	40.59	16.83	8.91	22.77
R	0.84	0.83	0.83	0.84

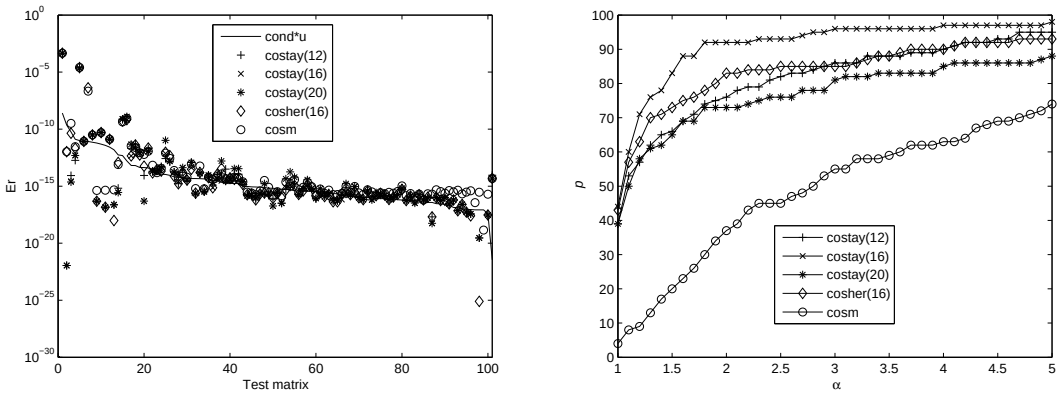
(a) Comparative `costay-cosm`.

9	12	16	20
55.45	46.53	44.55	44.55
30.69	28.71	20.59	19.80
13.86	24.76	34.66	35.65
0.88	0.90	0.90	0.92

(b) Comparative `costay-cosher`.

Figura 3.8: Comparatives `costay-cosm` and `costay-cosher`. The first three rows show the percentage of times that relative error of `costay` is lower (L), equal (E) or greater (G) than relative error of `cosm` or `cosher`. The last row shows the ratio (R) of cost in terms of matrix products between `costay` divided by the cost of `cosm` in 3.8a, and `cosher` in 3.8b.

To test the numerical stability of functions we plotted the normwise relative errors of functions `cosm`, `cosher` and `costay` for $m_M = 12, 16, 20$. Figure 3.9a shows the relative errors of all implementations, and a solid line that represents the unit roundoff multiplied by the relative condition number of the cosine function at X [Hig08, p. 55]. Relative condition number was computed using the MATLAB function `funm_condest1` from the Matrix Function Toolbox [Hig08, Appendix D] (`http://www.ma.man.ac.uk/~higham/mftoolbox`). For a method to perform in a backward and forward stable manner, its error should lie not far above this line on the graph [Hig05, p. 1188]. Figure 3.9a shows that in general the functions performed in a numerically stable way apart from some exceptions.



(a) Normwise relative error.

(b) Performance profile.

Figura 3.9: Normwise relative errors and perfomance profile of `cosm`, `cosher(16)` and `costay` for $m_M = 12, 16, 20$.

Figure 3.9b shows the performances [DM02] of the functions compared, where α coordinate varies between 1 and 5 in steps equal to 0.1, and p coordinate is the probability that the considered algorithm has a relative error lower than or equal to α-times the smallest error over all the methods, where probabilities are defined over all matrices, showing that the most accurate function is `costay` with $m_M = 16$ followed by `cosher` with $m_M = 16$.

3.4.8. Conclusions

In this work an accurate Taylor algorithm to compute matrix cosine is proposed. The new algorithm uses a scaling technique based on the double angle formula and sharp bounds for the forward absolute error, and the Horner and Paterson-Stockmeyer's method for computing the Taylor approximation. A MATLAB implementation of this algorithm has been compared with MATLAB function `cosher`, based on Hermites series [DSIR13], and the MATLAB function `cosm`, based on the Padé algorithm given in [HH05]. Numerical experiments show that the new algorithm has lower computational costs and higher accuracy than both functions `cosher` and `cosm` in the majority of test matrices. The new proposed Taylor algorithm provided the highest accuracy and lowest cost when maximum order $m_M = 16$ was used in tests, and this maximum order is therefore recommended.

Bibliografía

[AMH09] A. H. Al-Mohy and N. J. Higham. A new scaling and squaring algorithm for the matrix exponential. *SIAM J. Matrix Anal. Appl.*, 31(3):970–989, 2009.

[BCC+97] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, and I. Dhillon. *ScaLAPACK Users' Guide*. SIAM, 1997.

[Cos] http://personales.upv.es/jorsasma/costay.m.

[DH03] P. I. Davies and N. J. Higham. A Schur–Parlett algorithm for computing matrix functions. *SIAM J. Matrix Anal. Appl.*, 25(2):464–485, 2003.

[DJ98] E. Defez and L. Jódar. Some applications of Hermite matrix polynomials series expansions. *J. Comput. Appl. Math.*, 99:105–117, 1998.

[DM02] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Programming*, 91:201–213, 2002.

[DS57] N. Dunford and J. Schwartz. *Linear Operators, Part I: General Theory*. Interscience, New York, 1957.

[DSIR09] E. Defez, J. Sastre, J. Ibáñez, and P. A. Ruiz. Computing matrix functions solving coupled differential models. *Math. Comput. Model.*, 50(5-6):831–839, 2009.

[DSIR11] E. Defez, J. Sastre, J. Ibáñez, and P. A. Ruiz. Solving engineering models using matrix functions. In *Proc. Int. Conf. on Math. Model. in Engineering & Human Behaviour 2011*, pages 1–17, 2011.

[DSIR13] E. Defez, J. Sastre, J. Ibáñez, and P. Ruiz. Computing matrix functions arising in engineering models with orthogonal matrix polynomials. *Math. Comput. Model.*, 57(7–8):1738–1743, 2013.

[DTS11] E. Defez, Michael M. Tung, and J. Sastre. Improvement on the bound of hermite matrix polynomials. *Linear Algebra and its Applications*, 434(8):1910–1919, 2011.

[GL96] G. H. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins Studies in Mathematical Sciences. The Johns Hopkins University Press, third edition, 1996.

[HH05] G. I. Hargreaves and N. J. Higham. Efficient algorithms for the matrix cosine and sine. *Numer. Algorithms*, 40:383–400, 2005.

[Hig93] N. J. Higham. The Test Matrix Toolbox for MATLAB. Numerical Analysis Report No. 237, Manchester Centre for Computational Mathematics, Manchester Centre for Computational Mathematics, Manchester, England, December 1993.

[Hig05] N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl.*, 26(4):1179–1193, 2005.

[Hig08] N. J. Higham. *Functions of Matrices: Theory and Computation*. SIAM, Philadelphia, PA, USA, 2008.

[HS03] N. J. Higham and M. I. Smith. Computing the matrix cosine. *Numer. Algorithms*, 34:13–26, 2003.

[HT00] N. J. Higham and F. Tisseur. A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra. *SIAM J. Matrix Anal. Appl.*, 21:1185–1201, 2000.

[JC96] L. Jódar and R. Company. Hermite matrix polynomials and second order matrix differential equations. *Journal Approximation Theory Application*, 12(2):20–30, 1996.

[Leb72] N. N. Lebedev. *Special Functions and Their Applications*. Dover Publications, New York, 1972.

[ML03] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*. *SIAM Review*, 45:3–49, 2003.

[PS73] M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, 2(1):60–66, 1973.

[SB80] S. M. Serbin and S. A. Blalock. An algorithm for computing the matrix cosine. *SIAM J. Sci. Statist. Comput.*, 1(2):198–204, 1980.

[Ser79] S. M. Serbin. Rational approximations of trigonometric matrices with application to second-order systems of differential equations. *Appl. Math. Comput.*, 5(1):75–92, 1979.

[SIDR11a] J. Sastre, J. Ibáñez, E. Defez, and P. Ruiz. Accurate matrix exponential computation to solve coupled differential models in engineering. *Math. Comput. Model.*, 54:1835–1840, 2011.

[SIDR11b] J. Sastre, J. Ibáñez, E. Defez, and P. Ruiz. Efficient orthogonal matrix polynomial based method for computing matrix exponential. *Appl. Math. Comput.*, 217(14):6451–6463, 2011.

[SIRD13] Jorge Sastre, Javier Ibáñez, Pedro Ruiz, and Emilio Defez. Efficient computation of the matrix cosine. *Applied Mathematics and Computation*, 219(14):7575 – 7585, 2013.

[YL93] S.-T. Yau and Y. Y. Lu. Reducing the symmetric matrix eigenvalue problem to matrix multiplications. *SIAM J. Sci. Comput.*, 14(1):121–136, 1993.

Capítulo 4

Aplicaciones

4.1. Introducción

En este capítulo se incluyen dos artículos relacionados con la aplicación de la función exponencial matricial en el área de la resolución de ecuaciones diferenciales ordinarias. A continuación se presenta se presenta un breve resumen de los mismos:

- *"Solving Initial Value Problems for Ordinary Differential Equations by two approaches: BDF and Piecewise-linearized Method"* (sección 4.2, [IHAR09]): en este trabajo se desarrollan y comparan cinco algoritmos para la resolución de problemas de valor inicial dentro del área de las ecuaciones diferenciales ordinarias. Uno de los métodos analizados requiere del cálculo de la exponencial de una matriz en cada paso de tiempo, para lo cual se utilizan dos versiones basadas en los aproximantes diagonales de Padé: una utilizando la técnica de escalado y potenciación y otra sin utilizarla. Para poder comparar los cinco algoritmos entre sí, se han implementado todos ellos tanto en MATLAB como en FORTRAN.
- *"A Piecewise-linearized Algorithm based on Krylov Subspace for solving stiff ODEs"* (sección 4.3, [IHRA11]): en este segundo artículo se implementan dos nuevos algoritmos en MATLAB para la resolución de ecuaciones diferenciales ordinarias de valor inicial utilizando un método basado en subespacios de Krylov y se comparan con los dos mejores algoritmos de los implementados en el artículo anterior. En este caso, de nuevo se hace necesario el cálculo de la exponencial de una matriz en cada paso de tiempo, para lo cual se desarrolló una nueva función basada en aproximantes diagonales de Padé y en la técnica de escalado y potenciación. Los resultados experimentales demuestran que los nuevos algoritmos resultan más competitivos, especialmente conforme se va incrementando el tamaño del problema.

En las siguientes secciones de este capítulo se pueden consultar los artículos mencionados al completo.

4.2. Solving Initial Value Problems for Ordinary Differential Equations by two approaches: BDF and Piecewise-linearized Methods

Referencia del artículo:
J. Ibáñez, V. Hernández, E. Arias and P.A. Ruiz
Solving Initial Value Problems for Ordinary Differential Equations by two approaches BDF and Piecewise–linearized Methods
Computer Physics Communications, Volume 180, Issue 5, May 2009, Pages 712–723, ISSN 0010-4655,
http://dx.doi.org/10.1016/j.cpc.2008.11.013

Abstract

Many scientific and engineering problems are described using Ordinary Differential Equations (ODEs), where the analytic solution is unknown. Much research has been done by the scientific community on developing numerical methods which can provide an approximate solution of the original ODE. In this work, two approaches have been considered based on BDF and Piecewise-linearized Methods. The approach based on BDF methods uses a Chord-Shamanskii iteration for computing the nonlinear system which is obtained when the BDF schema is used. Two approaches based on piecewise-linearized methods have also been considered. These approaches are based on a theorem proved in this paper which allows to compute the approximate solution at each time step by means of a block-oriented method based on diagonal Padé approximations. The difference between these implementations is in using or not using the scale and squaring technique.

Five algorithms based on these approaches have been developed. MATLAB and Fortran versions of the above algorithms have been developed, comparing both precision and computational costs. BLAS and LAPACK libraries have been used in Fortran implementations. In order to compare in equality of conditions all implementations, algorithms with fixed step have been considered. Four of the five case studies analyzed come from biology and chemical kinetics stiff problems. Experimental results show the advantages of the proposed algorithms, especially when they are integrating stiff problems.

4.2.1. Introduction

Much research has been done by the scientific community on developing numerical methods which permit an approximate solution to ODEs. In recent years many review articles and books have appeared on numerical methods for integrating ODEs, in particular in stiff cases [HW96]. Stiff problems are very common problems in many fields of the applied sciences: control theory, biology, chemical kinetics, electronic circuit theory, fluids, etc. One of the most popular multistep method families for solving IVPs for stiff ODEs is formed by the BDF methods [CH52, Hin80, Gea71, BBH89]. Another approach used in this paper is based on piecewise-linearized methods. These methods solve an IVP by approximating the right-hand side of the ODE by means of a Taylor polynomial of degree one. The resulting approximation can be integrated analytically to obtain the solution in each subinterval and

yields the exact solution for linear problems. In [RGL97, GL98] an exhaustive study of this approach is introduced. The developed piecewise-linearized algorithms use Theorem 10 and Corollary 1 (Section 4.2.3) which allow to compute the approximate solution at each time step by means of a block-oriented method based on diagonal Padé approximations.

The paper is structured as follows. In Section 4.2.2 a BDF algorithm is presented. The proposed approaches for solving IVPs by a piecewise-linearized method based on diagonal Padé approximations are presented in Section 4.2.3. The experimental results are shown in Section 4.2.4. Finally, some conclusions and future work are outlined in Section 4.2.5.

4.2.2. A BDF algorithm

Let the IVP

$$\dot{x}(t) = f(t, x(t)),\ t \in [t_0, t_f], x(t_0) = x_0, \tag{4.1}$$

where $f(t, x(t)) \in \mathbb{R}^n$, $t \in [t_0, t_f]$, satisfies the necessary conditions under which the problem has a unique solution.

In this section an algorithm is presented which solves IVPs for ODEs by means of a BDF approach [AP98, Chapter 5] which uses a Chord-Shamanskii method [Kel95, Chapter 5] to solve the implicit equations that appear in BDF methods. In a BDF scheme, the integration interval $[t_0, t_f]$ is divided so that the approximate solution at t_i, x_i, is obtained by solving an implicit nonlinear system obtained by differentiating the polynomial which interpolates past values of x_i, and setting the derivative at t_i to $f(t_i, x_i)$. Several methods have been implemented for solving that implicit nonlinear system; however, in the context of stiff ODEs, one of the better choices is to apply implicit schemes based on Newton's or quasi-Newton methods.

If $\{t_0, t_1, \cdots t_f\}$ is a partition of interval $[t_0, t_f]$ and a BDF scheme is applied, the approximate solution x_i is obtained by solving the following equation:

$$x_i - \sum_{j=1}^{r} \alpha_j x_{i-j} - \Delta t_{i-1} \beta f(t_i, x_i) = 0, \tag{4.2}$$

where $\Delta t_{i-1} = t_i - t_{i-1}$, and α_j $(j = 1, 2, \cdots, r)$, β are parameters which appear in Table 4.1, where r is the order of BDF method

Usually a Newton method is used to solve (4.2) at each time step. In this way, x_i is obtained from the Newton iteration

$$\begin{gathered} x_i^0 = x_{i-1}, \\ \left(I_n - h\beta J_i^{l-1}\right) \Delta x = -F(x_i^l), x_i^l = x_i^{l-1} + \Delta x, l \geq 1, \end{gathered} \tag{4.3}$$

where J_i^{l-1} is the Jacobian matrix evaluated at (x_i^{l-1}, t_i).

In this paper an inexact Newton technique for solving (4.3) has been used which significantly speeds up the implicit BDF integrator without loss of accuracy. This approach [Kel95, p. 86] uses a combination of the Chord and Shamanskii methods, based on the reduction

in the nonlinear residual. This latter option decides if the Jacobian should be recomputed based on the ratio of successive residuals.

If two consecutive approaches x_i^{l-1} and x_i^l verify that $\|F(x_i^l)\| / \|F(x_i^{l-1})\|$ is below a given threshold, the Jacobian is not recomputed. If the ratio is too large, the Jacobian matrix at x_i^l is computed for use in the subsequent chord steps. In addition, a threshold for the ratio of successive residuals is also input. The Jacobian matrix is recomputed and factored if either the ratio of successive residuals exceeds the threshold $0 < \rho < 1$ or the number of iterations without an update exceeds a parameter $m \in \mathbb{N}$. Algorithm 4.1 (*iodbcs*) solves the IVP for ODEs (4.1) by means of the above BDF Chord-Shamanskii method.

Algorithm 4.1 $[\{x_i\}, e] = iodbcs(ff, Jf, x_0, t_0, t_f, \Delta t, r, tol, m, \rho)$

Inputs: functions ff and Jf compute $f(\tau, x) \in \mathbb{R}^n$ and the Jacobian matrix $J(\tau, x) \in \mathbb{R}^n$ ($\tau \in \mathbb{R}$, $x \in \mathbb{R}^n$); vector of initial conditions $x_0 \in \mathbb{R}^n$; initial time $t_0 \in \mathbb{R}$; final time $t_f \in \mathbb{R}$; step size Δt; order $r \in \mathbb{N}$ of the BDF method; tolerance vector $tol \in \mathbb{R}^2$ that contains the relative error tolerance (tol_1) and the absolute error tolerance (tol_2); maximum number $m \in \mathbb{N}$ of iterations without computing the Jacobian matrix; threshold ρ

Outputs: Solutions $\{x_i\}$ ($x_i \in \mathbb{R}^n$) at $t_0, t_0+\Delta t, t_0+2\Delta t, \ldots$; $e \in \mathbb{Z}$ indicates the convergence of the method (if $e = -1$ the method does not converge)

1 Initialize α and β according to the values given in Table 4.1

2 $i = 0$

3 $m = \lceil (t_f - t_0)/\Delta t \rceil$;

4 For $i = 1 : m$

4.1 $t_i = t_{i-1} + \Delta t$

4.2 $p = \text{mín}(r, i)$

4.3 $f_0 = \sum_{j=1}^{r} \alpha_j x_{i-j}$

4.4 $x_i = x_{i-1}$

4.5 $f = ff(t_i, x_i)$

4.6 $r_c = \|f\|_\infty$; $r_c = r_0$

4.7 While $r_c \geq t_1 r_0 + tol_2$

4.7.1 $J = Jf(t_i, x_i)$

4.7.2 $[L, U] = lu(I - \Delta t \beta_p J)$ (LU decomposition)

4.7.3 $i_s = 0$; $\rho = 0$

4.7.4 While $i_s < m$ and $\rho \leq m$

4.7.4.1 $i_s = i_s + 1$

4.7.4.2 Solve the lower linear system $Ly = f_0 - x_i + f$ for y

4.7.4.3 Solve the upper linear system $U\Delta x = y$ for Δx

4.7.4.4 $x_i = x_i + \Delta x$

4.7.4.5 $f = ff(t_i, x_i)$

4.7.4.6 $r_+ = \|f\|_\infty$

4.7.4.7 $\sigma = r_+ / r_m$; $r_c = r_+$

4.7.4.8 If $\|\Delta x\| < tol_1 r_0 + r_1$ Leave While loop 4.7.4

4.8 If $\sigma > 1$

4.8.1 $e = -1$ (The method does not converge)

4.8.2 Return

	β	α_1	α_2	α_3	α_4	α_5
$r=1$	1	1				
$r=2$	2/3	4/3	-1/3			
$r=3$	6/11	18/11	-9/11	2/11		
$r=4$	12/25	48/25	-36/25	16/25	-3/25	
$r=5$	60/137	300/137	-300/137	200/137	-75/137	12/137

Tabla 4.1: BDF method parameters (order r=1, 2, 3, 4 and 5).

4.2.3. A piecewise-linearized approach for solving IVPs for ODEs

Given a partition $t_0 < t_1 < \cdots < t_{l-1} < t_l = t_f$ of interval $[t_0, t_f]$, IVP (4.1) can be approximated by means of a set of IVPs obtained as a result of the linear approximation of $f(t, x(t))$ at each subinterval [RGL97, C. 03]

$$\dot{y}(t) = f_i + J_i(y(t) - y_i) + g_i(t - t_i),\ t \in [t_i, t_{i+1}],$$
$$y(t_i) = y_i,\ i = 0, 1, \cdots, l-1,$$

where

$$f_i = f(t_i, y_i) \in \mathbb{R}^n,$$
$$J_i = \frac{\partial f}{\partial x}(t_i, y_i) \in \mathbb{R}^{n\times n} (\text{Jacobian matrix}),$$
$$g_i = \frac{\partial f}{\partial t}(t_i, y_i) \in \mathbb{R}^n (\text{gradient vector}).$$

The IVP associated to the first subinterval is

$$\dot{y}(t) = f_0 + J_0(y(t) - y_0) + g_0(t - t_0), t \in [t_0, t_1],$$
$$y(t_0) = y_0 = x_0.$$

Its analytic solution is given by

$$y(t) = y_0 + \int_{t_0}^{t} e^{J_0(t-\tau)}[f_0 + g_0(\tau - t_0)]d\tau \ \ , t \in [t_0, t_1],$$

therefore it is possible to compute $y_1 = y(t_1)$.

By proceeding in the same way, the analytic solution of IVP associated to subinterval i, $i = 1, \cdots, l-1$,

$$\dot{y}(t) = f_i + J_i(y(t) - y_i) + g_i(t - t_i),\ t \in [t_i, t_{i+1}],\ \ y(t_i) = y_i$$

is given by

$$y(t) = y_i + \int_{t_i}^{t} e^{J_i(t-\tau)}[f_i + g_i(\tau - t_i)]d\tau,\ \ t \in [t_i, t_{i+1}]\ . \tag{4.4}$$

If second order partial derivatives of $f(t, x)$ are bounded on $[t_0, t_f] \times \mathbb{R}^n$, then the above piecewise-linearized method converges [RGL97]. If a (1,1) Padé approximation is used for computing $e^{J_i(t-t_i)}$, the above method is consistent of order 1 for autonomous ODEs and 2 for non-autonomous ODEs and linearly stable [GL98, p. 26].

The piecewise-linearized approaches for solving IVPs presented in this paper are based on the following theorem and corollary.

Theorem 10 *The solution of IVP*

$$\begin{gathered}\dot{y}(t) = f_i + J_i(y(t) - y_i) + g_i(t - t_i),\ t \in [t_i, t_{i+1}], \\ y(t_i) = y_i \in \mathbb{R}^n, \\ f_i \in \mathbb{R}^n, J_i \in \mathbb{R}^{n\times n}, g_i \in \mathbb{R}^n,\end{gathered} \tag{4.5}$$

is

$$y(t) = y_i + F_{12}^{(i)}(t - t_i)f_i + F_{13}^{(i)}(t - t_i)g_i, \tag{4.6}$$

where $F_{12}^{(i)}(t-t_i)$ *and* $F_{13}^{(i)}(t-t_i)$ *are the blocks* $(1,2)$ *and* $(1,3)$ *of* $e^{C_i(t-t_i)}$, *and*

$$C_i = \begin{bmatrix} J_i & I_n & 0_n \\ 0_n & 0_n & I_n \\ 0_n & 0_n & 0_n \end{bmatrix}.$$

Proof. Defining $s = \tau - t_i$ and $\theta = t - t_i$, the integral which appears in (4.4) can be expressed as

$$\int_{t_i}^{t} e^{J_i(t-\tau)}[f_i + g_i(\tau - t_i)]d\tau = \left[\int_0^\theta e^{J_i(\theta-s)}ds\right]f_i + \left[\int_0^\theta e^{J_i(\theta-s)}sds\right]g_i. \tag{4.7}$$

Because C_i is an upper-triangular block matrix, the exponential of $C_i\theta$ has the same structure,

$$e^{C_i\theta} = \begin{bmatrix} F_{11}^{(i)}(\theta) & F_{12}^{(i)}(\theta) & F_{13}^{(i)}(\theta) \\ 0_n & F_{22}^{(i)}(\theta) & F_{23}^{(i)}(\theta) \\ 0_n & 0_n & F_{33}^{(i)}(\theta) \end{bmatrix},$$

where $F_{jk}^{(i)}(\theta)$, $1 \le j \le k \le 3$, are square matrices of order n. Since

$$\tfrac{de^{C_i\theta}}{d\theta} = C_i e^{C_i\theta}\ \ , e^{C_i\theta}|_{\theta=0} = I_{3n},$$

the following IVPs are obtained

$$\frac{dF_{11}^{(i)}(\theta)}{d\theta} = J_i F_{11}^{(i)}(\theta),\ F_{11}^{(i)}(0) = I_n, \tag{4.8}$$

$$\frac{dF_{22}^{(i)}(\theta)}{d\theta} = 0,\ F_{22}^{(i)}(0) = I_n, \tag{4.9}$$

$$\frac{dF_{33}^{(i)}(\theta)}{d\theta} = 0,\ F_{33}^{(i)}(0) = I_n, \tag{4.10}$$

$$\frac{dF_{23}^{(i)}(\theta)}{d\theta} = F_{33}^{(i)}(\theta),\ F_{23}^{(i)}(0) = 0_n, \tag{4.11}$$

$$\frac{dF_{12}^{(i)}(\theta)}{d\theta} = J_i F_{12}^{(i)}(\theta) + F_{22}^{(i)}(\theta), F_{12}^{(i)}(0) = 0_n, \tag{4.12}$$

$$\frac{dF_{13}^{(i)}(\theta)}{d\theta} = J_i F_{13}^{(i)}(\theta) + F_{23}^{(i)}(\theta),\ F_{13}^{(i)}(0) = 0_n. \tag{4.13}$$

The solutions of (4.8), (4.9) and (4.10) are given by

$$F_{11}^{(i)}(\theta) = e^{J_i\theta},$$
$$F_{22}^{(i)}(\theta) = I_n,$$
$$F_{33}^{(i)}(\theta) = I_n.$$

Therefore the solutions of (4.11), (4.12) and (4.13) are

$$F_{23}^{(i)}(\theta) = \theta I_n, \tag{4.14}$$
$$F_{12}^{(i)}(\theta) = \int_0^\theta e^{J_i(\theta-s)}ds, \tag{4.15}$$
$$F_{13}^{(i)}(\theta) = \int_0^\theta e^{J_i(\theta-s)}sds. \tag{4.16}$$

Note that the integrals involved in (4.7) can be computed by means of (4.15) and (4.16). In conclusion, once the Jacobian matrix J_i and the vectors g_i and f_i have been computed, the solution of IVP (4.5) is given by

$$y(t) = y_i + F_{12}^{(i)}(\theta)f_i + F_{13}^{(i)}(\theta)g_i.$$

As $\theta = t - t_i$, then

$$y(t) = y_i + F_{12}^{(i)}(t-t_i)f_i + F_{13}^{(i)}(t-t_i)g_i, \tag{4.17}$$

so the theorem is proved. □

According to Theorem 10, the approximate solution of IVP (4.5) at t_{i+1} is obtained from the approximate solution at t_i by the following expression

$$y_{i+1} = y_i + F_{12}^{(i)}(\Delta t_i)f_i + F_{13}^{(i)}(\Delta t_i)g_i, \tag{4.18}$$

where $\Delta t_i = t_{i+1} - t_i$.

For autonomous ODEs the following result is obtained.

Corollary 1 *The solution of IVP*

$$\dot{y}(t) = f_i + J_i(y(t) - y_i),\ y(t_i) = y_i, \tag{4.19}$$

is

$$y(t) = y_i + F_{12}^{(i)}(t-t_i)f_i,$$

where $F_{12}^{(i)}(t-t_i)$ *is the block (1,2) of matrix* $e^{C_i(t-t_i)}$, *and*

$$C_i = \begin{bmatrix} J_i & I_n \\ 0_n & 0_n \end{bmatrix}.$$

Proof. It is enough to apply Theorem 10 for $g_i = 0_{n\times 1} \in \mathbb{R}^n$. □

According to Corollary 1, the approximate solution of IVP (4.19) at t_{i+1} is obtained from the approximate solution at t_i by the following expression

$$y_{i+1} = y_i + F_{12}^{(i)}(\Delta t_i) f_i. \tag{4.20}$$

Algorithm 4.2 is consequence of Theorem 1. This algorithm (*inolex*) computes the approximate solution of the IVP for non-autonomous ODEs (4.1) by means of a piecewise-linearized method based on the exponential of matrix $C_i \Delta t_i$.

Algorithm 4.2 $\{x_i\} = inolex(data, x_0, t_0, t_f, \Delta t)$

Inputs: *data* is a function that computes Jacobian matrix $J(\tau, x) \in \mathbb{R}^{n\times n}$ and function vector $f(\tau, x) \in \mathbb{R}^n$ ($\tau \in \mathbb{R}$, $x \in \mathbb{R}^n$); vector $x_0 \in \mathbb{R}^n$ of initial conditions ; initial time $t_0 \in \mathbb{R}$; final time $t_f \in \mathbb{R}$; step size $\Delta t \in \mathbb{R}$

Outputs: Vector of solutions $\{x_i\}$ ($x_i \in \mathbb{R}^n$) at $t_0, t_0 + \Delta t, t_0 + 2\Delta t, \ldots$

1 $m = \lceil (t_f - t_0)/\Delta t \rceil$

2 For $i = 0 : m - 1$

2.1 $[J, f, g] = data(t_i, x_i)$

2.2 $C = \begin{bmatrix} J & I_n & 0_n \\ 0_n & 0_n & I_n \\ 0_n & 0_n & 0_n \end{bmatrix}$

2.3 $F = e^{C\Delta t}$

2.4 $t_{i+1} = t_i + \Delta t$

2.5 $x_{i+1} = x_i + F_{12} f + F_{13} g$

4.2.3.1. Algorithms based on the scaling and squaring technique

The (p, q) Padé approximation to e^A is defined by

$$R_{pq} = [D_{pq}(A)]^{-1} N_{pq}(A),$$

where

$$N_{pq}(A) = \sum_{k=0}^{p} \frac{(p+q-k)!p!}{(p+k)!k!(p-k)!} A^k$$

and

$$D_{pq}(A) = \sum_{k=0}^{p} \frac{(p+q-k)!p!}{(p+k)!k!(p-k)!} (-A)^k.$$

Non-singularity of $D_{pq}(A)$ is assured if p and q are large enough or if the eigenvalues of A are negative.

The problem with this method is that it only provides good approaches near the origin [GL96, p.573]. This problem can be avoided by using the widely used scaling and squaring method for computing the matrix exponential [ML03, Hig04] by exploiting the equality

$$e^A = \left(e^{A/m}\right)^m.$$

The idea is to choose m to be a power of two ($m = 2^j$) for which $e^{A/m}$ can be reliably and efficiently computed, and then to form the matrix $\left(e^{A/m}\right)^m$ by repeated squaring. One commonly used criterion for choosing m is to make it the smallest power of two for which $\|A\|/m \leq 1$.

Diagonal Padé approximants ($p = q$) are preferred, since R_{pq} ($p \neq q$) is less accurate than R_{ll}, where $l = max(p, q)$, but R_{ll} can be computed at same cost.

Algorithm 4.3 (*exmdpa*) computes the exponential of a matrix by means of a scaling-squaring diagonal Padé approximation method with variable order q.

Algorithm 4.3 $F = exmdpa(A, q)$

Inputs: Matrix $A \in \mathbb{R}^{n\times n}$; order $q \in \mathbb{N}$ of diagonal Padé approximation of the exponential function

Output: Matrix $F = e^A \in \mathbb{R}^{n\times n}$

1 $nor = \|A\|_\infty$

2 $j_A = \max(0, 1 + int(\log_2(nor)))$

3 $s = \frac{1}{2^{j_A}}$

4 $A = sA$

5 $X = A$

6 $N = I_n + c_1(1)$

7 $D = I_n + c_2(1)$

8 For $k = 2 : q$

  8.1 $X = XA$

  8.2 $N = N + c_1(k)X$

  8.3 $D = D + c_2(k)X$

9 Solve $DF = N$ for F using Gaussian elimination

10 For $k = 1 : j_A$

  10.1 $F = F^2$

In order to reduce the high computational and storage costs of Algorithm 4.2, a block oriented version of Algorithm 4.3 has been developed. In this way, y_{i+1} can be computed without explicitly computing the exponential of matrix $C\Delta t$ (step 2.3 of Algorithm 4.2). This algorithm only computes blocks (1,2) and (1,3) of the exponential of matrix

$$A = \begin{bmatrix} J_i & I_n & 0_n \\ 0_n & 0_n & I_n \\ 0_n & 0_n & 0_n \end{bmatrix}. \tag{4.21}$$

With this goal, some steps of Algorithm 4.3 are adapted to matrices

$$X = \begin{bmatrix} X_{11} & X_{12} & X_{13} \\ 0_n & X_{22} & X_{23} \\ 0_n & 0_n & X_{33} \end{bmatrix},$$

$$N = \begin{bmatrix} N_{11} & N_{12} & N_{13} \\ 0_n & N_{22} & N_{23} \\ 0_n & 0_n & N_{33} \end{bmatrix},$$

$$D = \begin{bmatrix} D_{11} & D_{12} & D_{13} \\ 0_n & D_{22} & D_{23} \\ 0_n & 0_n & D_{33} \end{bmatrix},$$

$$F = \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ 0_n & F_{22} & F_{23} \\ 0_n & 0_n & F_{33} \end{bmatrix}.$$

Step 4: $A = sA$.

$$A = sA = \begin{bmatrix} sJ_i & sI_n & 0_n \\ 0_n & 0_n & sI_n \\ 0_n & 0_n & 0_n \end{bmatrix}.$$

Let $J_i = sJ_i$, then A can be expressed as

$$A = \begin{bmatrix} J_i & sI_n & 0_n \\ 0_n & 0_n & sI_n \\ 0_n & 0_n & 0_n \end{bmatrix}.$$

Step 5: $X = A$.

$$X = \begin{bmatrix} J_i & sI_n & 0_n \\ 0_n & 0_n & sI_n \\ 0_n & 0_n & 0_n \end{bmatrix}.$$

Step 6: $N = I_n + c_1(1)A$.

$$\begin{bmatrix} N_{11} & N_{12} & N_{13} \\ 0_n & N_{22} & N_{23} \\ 0_n & 0_n & N_{33} \end{bmatrix} = \begin{bmatrix} I_n + c_1(1)J_i & c_1(1)sI_n & 0_n \\ 0_n & I_n & c_1(1)sI_n \\ 0_n & 0_n & I_n \end{bmatrix},$$

therefore

$$\begin{aligned} N_{11} &= I_n + c_1(1)J_i, \\ N_{12} &= c_1(1)sI_n, \\ N_{13} &= 0_n, \\ N_{22} &= I_n, \\ N_{23} &= c_1(1)sI_n, \\ N_{33} &= I_n. \end{aligned}$$

Step 7: $D = I_n + c_2(1)A$.

$$\begin{bmatrix} D_{11} & D_{12} & D_{13} \\ 0_n & D_{22} & D_{23} \\ 0_n & 0_n & D_{33} \end{bmatrix} = \begin{bmatrix} I_n + c_2(1)J_i & c_2(1)sI_n & 0_n \\ 0_n & I_n & c_2(1)sI_n \\ 0_n & 0_n & I_n \end{bmatrix},$$

therefore

$$\begin{aligned} D_{11} &= I_n + c_2(1)J_i, \\ D_{12} &= c_2(1)sI_n, \\ D_{13} &= 0_n, \\ D_{22} &= I_n, \\ D_{23} &= c_2(1)sI_n, \\ D_{33} &= I_n. \end{aligned}$$

Substep 8.1: $X = XA$.

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ 0_n & 0_n & X_{23} \\ 0_n & 0_n & 0_n \end{bmatrix} = \begin{bmatrix} X_{11} & X_{12} & X_{13} \\ 0_n & 0_n & X_{23} \\ 0_n & 0_n & 0_n \end{bmatrix} \begin{bmatrix} J_i & sI_n & 0_n \\ 0_n & 0_n & sI_n \\ 0_n & 0_n & 0_n \end{bmatrix} = \begin{bmatrix} X_{11}J_i & sX_{11} & sX_{12} \\ 0_n & 0_n & 0_n \\ 0_n & 0_n & 0_n \end{bmatrix}.$$

Bearing in mind data dependencies, X_{ij}, $1 \le i \le j \le 3$, can be computed as follows

$$\begin{aligned} X_{13} &= sX_{12}, \\ X_{12} &= sX_{11}, \\ X_{11} &= X_{11}J_i, \\ X_{22} &= 0_n, \\ X_{23} &= 0_n, \\ X_{33} &= 0_n. \end{aligned}$$

Substep 8.2: $N = N + c_1(k)X$.

$$\begin{bmatrix} N_{11} & N_{12} & N_{13} \\ 0_n & N_{22} & N_{23} \\ 0_n & 0_n & N_{33} \end{bmatrix} = \begin{bmatrix} N_{11} + c_1(k)X_{11} & N_{12} + c_1(k)X_{12} & N_{13} + c_1(k)X_{13} \\ 0_n & N_{22} & N_{23} \\ 0_n & 0_n & N_{33} \end{bmatrix}.$$

As matrices N_{22}, N_{23} and N_{33} do not vary inside loop 8, then

$$N_{22} = I_n,$$
$$N_{23} = c_1(1)sI_n,$$
$$N_{33} = I_n.$$

Substep 8.3: $D = D + c_2(k)X$.

$$\begin{bmatrix} D_{11} & D_{12} & D_{13} \\ 0_n & D_{22} & D_{23} \\ 0_n & 0_n & D_{33} \end{bmatrix} = \begin{bmatrix} D_{11} + c_2(k)X_{11} & D_{12} + c_2(k)X_{12} & D_{13} + c_2(k)X_{13} \\ 0_n & D_{22} & D_{23} \\ 0_n & 0_n & D_{33} \end{bmatrix}.$$

As matrices D_{22}, D_{23} and D_{33} do not vary inside loop 8, then

$$D_{22} = I_n,$$
$$D_{23} = c_2(1)sI_n,$$
$$D_{33} = I_n.$$

Step 9: To compute F by solving $DF = N$.

$$\begin{bmatrix} D_{11} & D_{12} & D_{13} \\ 0_n & I_n & c_2(1)sI_n \\ 0_n & 0_n & I_n \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ 0_n & F_{22} & F_{23} \\ 0_n & 0_n & F_{33} \end{bmatrix} = \begin{bmatrix} N_{11} & N_{12} & N_{13} \\ 0_n & I_n & c_1(1)sI_n \\ 0_n & 0_n & I_n \end{bmatrix},$$

therefore

$$D_{11}F_{11} = N_{11},$$
$$D_{11}F_{12} + D_{12}F_{22} = N_{12},$$
$$D_{11}F_{13} + D_{12}F_{23} + D_{13}F_{33} = N_{13},$$
$$F_{22} = I_n,$$
$$F_{23} + c_2(1)sF_{33} = c_1(1)sI_n,$$
$$F_{33} = I_n.$$

Since $c_1(1) = 0.5$ and $c_2(1) = -0.5$, then $F_{23} = sI_n$ and matrices F_{11}, F_{12} and F_{13} can be computed solving the equations

$$D_{11}F_{11} = N_{11},$$
$$D_{11}F_{12} = N_{12} - D_{12},$$
$$D_{11}F_{13} = N_{13} - sD_{12} - D_{13}.$$

It is only necessary to know N_{11}, N_{12}, N_{13}, D_{11}, D_{12} and D_{13} to compute F_{11}, F_{12} and F_{13} in step 9.

Substep 10.1: $F = F^2$. Making the product

$$\begin{bmatrix} F_{11} & F_{12} & F_{13} \\ 0_n & F_{22} & F_{23} \\ 0_n & 0_n & F_{33} \end{bmatrix} = \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ 0_n & F_{22} & F_{23} \\ 0_n & 0_n & F_{33} \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ 0_n & F_{22} & F_{23} \\ 0_n & 0_n & F_{33} \end{bmatrix},$$

and equaling the corresponding blocks, then

$$\begin{aligned} F_{11} &= F_{11}^2, \\ F_{12} &= F_{11}F_{12} + F_{12}F_{22}, \\ F_{13} &= F_{11}F_{13} + F_{12}F_{23} + F_{13}F_{33}, \\ F_{22} &= F_{22}^2, \\ F_{23} &= F_{22}F_{23} + F_{23}F_{33}, \\ F_{33} &= F_{33}^2. \end{aligned}$$

Bearing in mind that before entering in loop 10 $F_{22} = I_n$ and $F_{33} = I_n$, then inside the above loop $F_{22} = I_n$ and $F_{33} = I_n$, therefore

$$F_{23} = F_{23} + F_{23} = 2F_{23}.$$

Because before entering loop 10 $F_{23} = sI_n$, then

$$F_{23} = 2^k sI_n$$

in the k iteration. According to data dependence, F_{11}, F_{12} and F_{13} can be computed as

$$\begin{aligned} F_{13} &= F_{11}F_{13} + F_{12}F_{23} + F_{13}, \\ F_{12} &= F_{11}F_{12} + F_{12}, \\ F_{11} &= F_{11}^2. \end{aligned}$$

The complete algorithm that solves IVP (4.1) corresponds to Algorithm 4.4 (*inolsp*). This algorithm solves IVPs for non-autonomous ODEs by a piecewise-linearized approach with scaling-squaring of the diagonal Padé approximants. This algorithm uses the following auxiliary algorithms:

- Algorithm 4.5 (*coedpa*) computes the coefficients of the polynomials of degree greater than zero in the diagonal Padé approximation of the exponential function.

- Algorithm 4.6 (*inlbsp*) computes the approximate solution at t_{i+1} of IVP for non-autonomous ODEs (4.1), obtained after the piecewise-linearized process, by a block-oriented version of the scaling-squaring diagonal Padé method. This is the block oriented version of Algorithm 4.2. The approximate computational cost of Algorithm 4.6 is $2\left(q + 3j_{J\Delta t} + \frac{7}{3}\right)n^3$ flops, where $j_{J\Delta t} = \max(0, 1 + int(\log_2(\|J\Delta t\|_\infty)))$.

Algorithm 4.4 $\{x_i\} = inolsp(data, x_0, t_0, t_f, \Delta t, q)$

Inputs: *Data* is a function that computes function vector $f(\tau, x) \in \mathbb{R}^n$, Jacobian matrix $J(\tau, x) \in \mathbb{R}^{n\times n}$ and gradient vector $g(\tau, x) \in \mathbb{R}^n$ ($\tau \in \mathbb{R}$, $x \in \mathbb{R}^n$); vector $x_0 \in \mathbb{R}^n$ of initial conditions; initial time $t_0 \in \mathbb{R}$; final time $t_f \in \mathbb{R}$; step size $\Delta t \in \mathbb{R}$; order $q \in \mathbb{N}$ of the diagonal Padé approximation of the exponential function

Outputs: Vectors of solutions $\{x_i\}$ ($x_i \in \mathbb{R}^n$) at $t_0, t_0 + \Delta t, t_0 + 2\Delta t, \ldots$

1 $[c_1, c_2] = coedpa(q)$ (Algorithm 4.5)

2 $m = \lceil (t_f - t_0)/\Delta t \rceil$

3 For $i = 0 : m - 1$

3.1 $[f, J, g] = data(t_i, x_i)$

3.2 $x_{i+1} = inlbsp(J, f, g, x_i, \Delta t, c_1, c_2)$ (Algorithm 4.6)

3.3 $t_{i+1} = t_i + \Delta t$

Algorithm 4.5 $[c_1, c_2] = coedpa(q)$

Inputs: Order $q \in \mathbb{N}$ of the diagonal Padé approximation of the exponential function

Outputs: Vectors $c_1, c_2 \in \mathbb{R}^q$ with the coefficients of terms greater than 0 in the (q,q) diagonal Padé approximation of the exponential function

1 $c_1(1) = 0.5$

2 $c_2(1) = -0.5$

3 For $k = 2 : q$

3.1 $c_1(k) = \frac{q-k+1}{(2q-k+1)k} c_1(k-1)$

3.2 $c_2(k) = (-1)^k c_1(k)$

Algorithm 4.6 $y_{i+1} = inlbsp(J, f, g, y_i, \Delta t, c_1, c_2)$

Inputs: Jacobian matrix $J \in \mathbb{R}^{n\times n}$; function vector $f \in \mathbb{R}^n$; gradient vector $g \in \mathbb{R}^n$; vector $y_i \in \mathbb{R}^n$; step size $\Delta t \in \mathbb{R}$; vectors $c_1, c_2 \in \mathbb{R}^q$ with the coefficients of terms of degree greater than 0 in the (q,q) diagonal Padé approximation of the exponential function

Output: Vector $y_{i+1} \in \mathbb{R}^n$ given by expression (4.18)

1 $nor = \|J\|_\infty \Delta t$

2 $j_{J\Delta t} = \text{máx}(0, 1 + int(\log_2(nor)))$; $s = \frac{\Delta t}{2^{j_{J\Delta t}}}$; $J = sJ$

3 $X_{11} = J$; $X_{12} = sI_n$; $X_{13} = 0_n$

4 $N_{11} = I_n + c_1(1)J$;$N_{12} = c_1(1)sI_n$; $N_{13} = 0_n$

5 $D_{11} = I_n + c_2(1)J$;$D_{12} = c_2(2)sI_n$; $D_{13} = 0_n$

6 For $k = 2 : q$

6.1 $X_{13} = sX_{12}$; $X_{12} = sX_{11}$; $X_{11} = X_{11}J$

6.2 $N_{11} = N_{11} + c_1(k)X_{11}$; $N_{12} = N_{12} + c_1(k)X_{12}$; $N_{13} = N_{13} + c_1(k)X_{13}$

6.3 $D_{11} = D_{11} + c_2(k)X_{11}$; $D_{12} = D_{12} + c_2(k)X_{12}$; $D_{13} = D_{13} + c_2(k)X_{13}$

7 Solve $D_{11}F_{11} = N_{11}$ for F_{11}

8 Solve $D_{11}F_{12} = N_{12} - D_{12}$ for F_{12}

9 Solve $D_{11}F_{13} = N_{13} - sD_{12} - D_{13}$ for F_{13}

10 For $k = 1 : j_{J\Delta t}$

10.1 $F_{13} = F_{11}F_{13} + sF_{12} + F_{13}$; $F_{12} = F_{11}F_{12} + F_{12}$; $F_{11} = F_{11}^2$

10.2 $s = 2s$

11 $y_{i+1} = y_i + F_{12}f + F_{13}g$

For autonomous ODEs the computational and storage costs can be reduced if Corollary 1 is applied. Hence, another algorithm (*iaolsp*) can be developed to solve IVPs for autonomous ODEs by a piecewise-linearized approach with scaling-squaring of the diagonal Padé approximants.

This algorithm uses the auxiliary Algorithms 4.5 (*coedpa*) and 4.8. Algorithm 4.8 (*ialbsp*) computes the approximate solution at t_{i+1} of IVP for autonomous ODEs (4.1), obtained after the piecewise-linearized process, by a block-oriented version of the scaling-squaring diagonal Padé method. The approximate computational cost of Algorithm 4.8 is $2\left(q + 2j_{J\Delta t} + \frac{4}{3}\right)n^3$ flops, where $j_{J\Delta t} = \text{máx}(0, 1 + int(\log_2(\|J\Delta t\|_\infty)))$.

Algorithm 4.7 $\{x_i\} = iaolsp(data, x_0, t_0, t_f, \Delta t, q)$

Inputs: *Data* is a function that computes function vector $f(\tau, x) \in \mathbb{R}^n$ and the Jacobian matrix $J(\tau, x) \in \mathbb{R}^{n\times n}$; vector $x_0 \in \mathbb{R}^n$ of initial conditions; initial time $t_0 \in \mathbb{R}$; final time $t_f \in \mathbb{R}$; step size $\Delta t \in \mathbb{R}$; order $q \in \mathbb{N}$ of the diagonal Padé approximation of the exponential function

Outputs: Vectors of solutions $\{x_i\}$ $(x_i \in \mathbb{R}^n)$ at $t_0, t_0 + \Delta t, t_0 + 2\Delta t, \ldots$

1 $[c_1, c_2] = coedpa(q)$ (Algorithm 4.5)

2 $m = \lceil (t_f - t_0)/\Delta t \rceil$

3 For $i = 0 : m - 1$

  3.1 $[f, J, g] = data(t_i, x_i)$

  3.2 $x_{i+1} = ialbsp(J, f, x_i, \Delta t, c_1, c_2)$ (Algorithm 4.8)

  3.3 $t_{i+1} = t_i + \Delta t$

Algorithm 4.8 $y_{i+1} = ialbsp(J, f, y_i, \Delta t, c_1, c_2)$

Inputs: Jacobian matrix $J \in \mathbb{R}^{n\times n}$; function vector $f \in \mathbb{R}^n$; vector $y_i \in \mathbb{R}^n$; step size $\Delta t \in \mathbb{R}$; vectors $c_1, c_2 \in \mathbb{R}^q$ with the coefficients of terms of degree greater than 0 in the (q,q) diagonal Padé approximation of the exponential function

Output: Vector $y_{i+1} \in \mathbb{R}^n$ given by expression (4.20)

1 $nor = \|J\|_\infty \Delta t$

2 $j_{J\Delta t} = \text{máx}(0, 1 + int(\log_2(nor)))$; $s = \frac{\Delta t}{2^{j_{J\Delta t}}}$; $J = sJ$

3 $X_{11} = J$; $X_{12} = sI_n$

4 $N_{11} = I_n + c_1(1)J$; $N_{12} = c_1(1)sI_n$

5 $D_{11} = I_n + c_2(1)J$; $D_{12} = c_2(1)sI_n$

6 For $k = 2 : q$

  6.1 $X_{12} = sX_{11}$; $X_{11} = X_{11}J$

  6.2 $N_{11} = N_{11} + c_1(k)X_{11}$; $N_{12} = N_{12} + c_1(k)X_{12}$

  6.3 $D_{11} = D_{11} + c_2(k)X_{11}$; $D_{12} = D_{12} + c_2(k)X_{12}$

7 Solve $D_{11}F_{11} = N_{11}$ for F_{11}

8 Solve $D_{11}F_{12} = N_{12} - D_{12}$ for F_{12}

9 For $k = 1 : j_{J\Delta t}$

  9.1 $F_{12} = F_{11}F_{12} + F_{12}$

  9.2 $F_{11} = F_{11}^2$

  9.3 $s = 2s$

10 $y_{i+1} = y_i + F_{12}f$

4.2.3.2. Algorithms not based on scale-squaring technique

Since matrix C of Algorithm 4.2 is multiplied by Δt, it is possible compute the approximate solution x_{i+1} without using the scaling-squaring technique. Therefore, the computational costs are reduced without loss of accuracy (see Section 4.2.4). Note that in this case it not is necessary to compute block N_{11}. The following algorithms solve IVPs for ODEs by that method:

- Algorithm 4.9 (*inolwp*) solves the IVP for non-autonomous ODEs (4.1) by a piecewise-linearized approach without scaling-square of the diagonal Padé approximants.
- Algorithm 4.10 (*iaolwp*) solves the IVP for autonomous ODEs (4.1) by a piecewise-linearized approach without scaling-square of the diagonal Padé approximants.

These algorithms use the following auxiliary algorithms:

- Algorithm 4.11 (*inlbwp*) computes the approximate solution at t_{i+1} of IVP for nonautonomous ODEs (4.1), obtained after the piecewise-linearized process, by a block-oriented implementation without scaling-squaring of diagonal Padé method. The approximate computational cost of Algorithm 4.11 is $2\left(q+\frac{4}{3}\right)n^3$ flops.

- Algorithm 4.12 (*ialbwp*) computes the approximate solution at t_{i+1} of IVP for autonomous ODEs (4.1), obtained after the piecewise-linearized process, by a block-oriented version without scaling-squaring implementation of the diagonal Padé method. The approximate computational cost of this algorithm is $2\left(q+\frac{1}{3}\right)n^3$ flops.

Figure 4.1 shows a scheme with the developed piecewise-linearized algorithms.

Algorithm 4.9 $\{x_i\} = inolwp(data, x_0, t_0, t_f, \Delta t, q)$

Inputs: *data* is a function that computes $f(\tau, x) \in \mathbb{R}^n$, $J(\tau, x) \in \mathbb{R}^{n\times n}$ and $g(\tau, x) \in \mathbb{R}^n$ ($\tau \in \mathbb{R}$, $x \in \mathbb{R}^n$); vector $x_0 \in \mathbb{R}^n$ of initial conditions; initial time $t_0 \in \mathbb{R}$; final time $t_f \in \mathbb{R}$; step size $\Delta t \in \mathbb{R}$; order $q \in \mathbb{N}$ of the diagonal Padé approximation of the exponential function

Outputs: Vector of solutions $\{x_i\}$ ($x_i \in \mathbb{R}^n$) at $t_0, t_0 + \Delta t, t_0 + 2\Delta t, \ldots$

1 $[c_1, c_2] = coedpa(q)$ (Algorithm 4.5)

2 $m = \lceil (t_f - t_0)/\Delta t \rceil$

3 For $i = 0 : m - 1$

  3.1 $[f, J, g] = data(t_i, x_i)$

  3.2 $x_{i+1} = inlbwp(J, f, g, x_i, \Delta t, c_1, c_2)$ (Algorithm 4.11)

  3.3 $t_{i+1} = t_i + \Delta t$

Algorithm 4.10 $\{x_i\} = iaolwp(data, x_0, t_0, t_f, \Delta t, q)$

Inputs: *Data* is a function that computes $f(\tau, x) \in \mathbb{R}^n$ and $J(\tau, x) \in \mathbb{R}^{n\times n}$ ($\tau \in \mathbb{R}$, $x \in \mathbb{R}^n$); vector $x_0 \in \mathbb{R}^n$ of initial conditions; initial time $t_0 \in \mathbb{R}$; final time $t_f \in \mathbb{R}$; step size $\Delta t \in \mathbb{R}$; order $q \in \mathbb{N}$ of the diagonal Padé approximation of the exponential function

Outputs: Vector of solutions $\{x_i\}$ ($x_i \in \mathbb{R}^n$) at $t_0, t_0 + \Delta t, t_0 + 2\Delta t, \ldots$

1 $[c_1, c_2] = coedpa(q)$ (Algorithm 4.5)

2 $m = \lceil (t_f - t_0)/\Delta t \rceil$

3 For $i = 0 : m - 1$

3.1 $[f, J, g] = data(t_i, x_i)$

3.2 $x_{i+1} = ialbwp(J, f, x_i, \Delta t, c_1, c_2)$ (Algorithm 4.12)

3.3 $t_{i+1} = t_i + \Delta t$

Algorithm 4.11 $y_{i+1} = inlbwp(J, f, g, y_i, \Delta t, c_1, c_2)$

Inputs: Jacobian matrix $J \in \mathbb{R}^{n\times n}$; function vector $f \in \mathbb{R}^n$; gradient vector $g \in \mathbb{R}^n$; vector $y_i \in \mathbb{R}^n$; step size $\Delta t \in \mathbb{R}$; vectors $c_1, c_2 \in \mathbb{R}^q$ with the coefficients of terms of degree greater than 0 in the (q,q) diagonal Padé approximation of the exponential function

Output: Vector $y_{i+1} \in \mathbb{R}^n$ given by expression (4.18)

1 $J = \Delta t J$

2 $X_{11} = J$; $X_{12} = \Delta t I_n$; $X_{13} = 0_n$

3 $N_{12} = c_1(1) I_n$; $N_{13} = 0_n$

4 $D_{11} = I_n + c_2(1) J$; $D_{12} = c_2(2) s I_n$; $D_{13} = 0_n$

5 For $k = 2 : q$

5.1 $X_{13} = sX_{12}$; $X_{12} = sX_{11}$; $X_{11} = X_{11} J$

5.2 $N_{12} = N_{12} + c_1(k) X_{12}$; $N_{13} = N_{13} + c_1(k) X_{13}$

5.3 $D_{11} = D_{11} + c_2(k) X_{11}$; $D_{12} = D_{12} + c_2(k) X_{12}$; $D_{13} = D_{13} + c_2(k) X_{13}$

6 Solve $D_{11} F_{12} = N_{12} - D_{12}$ for F_{12}

7 Solve $D_{11} F_{13} = N_{13} - sD_{12} - D_{13}$ for F_{13}

8 $y_{i+1} = y_i + F_{12} f + F_{13} g$

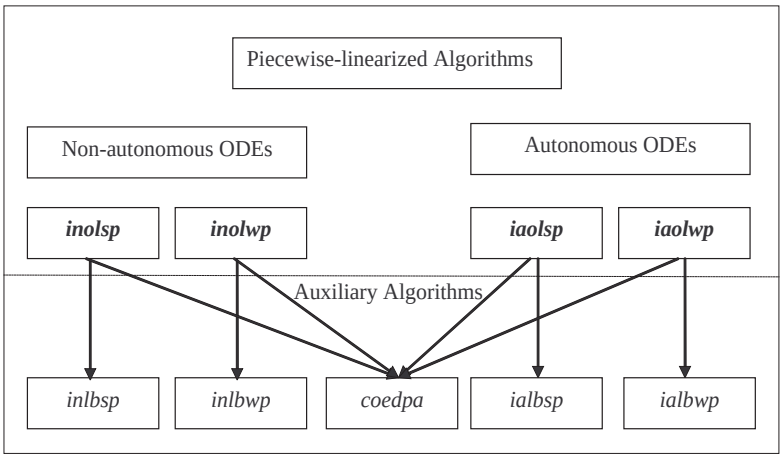


Fig. 4.1: Schema of piecewise-linearized Algorithms 4.4($inolsp$), 4.9($inolwp$), 4.7($iaolsp$) and 4.10($iaolwp$)

Algorithm 4.12 $y_{i+1} = ialbwp(J, f, y_i, \Delta t, c_1, c_2)$

Inputs: Jacobian matrix $J \in \mathbb{R}^{n\times n}$; function vector $f \in \mathbb{R}^n$; vector $y_i \in \mathbb{R}^n$; step size $\Delta t \in \mathbb{R}$; vectors $c_1, c_2 \in \mathbb{R}^q$ with the coefficients of terms of degree greater than 0 in the (q,q) diagonal Padé approximation of the exponential function

Output: Vector $y_{i+1} \in \mathbb{R}^n$ given by expression (4.20)

1 $J = \Delta t J$

2 $X_{11} = J$; $X_{12} = \Delta t I_n$

3 $D_{11} = I_n + c_2(1)J$

4 $D_{12} = c_2(1)sI_n$

5 For $k = 2 : q$

  5.1 $X_{12} = sX_{11}$; $X_{11} = X_{11}J$

  5.2 $N_{12} = N_{12} + c_1(k)X_{12}$

  5.3 $D_{11} = D_{11} + c_2(k)X_{11}$; $D_{12} = D_{12} + c_2(k)X_{12}$

6 Solve $D_{11}F_{12} = N_{12} - D_{12}$ for F_{12}

7 $y_{i+1} = y_i + F_{12}f$

4.2.4. Experimental results

The main objective of this section is to compare the algorithms developed in Sections 4.2.2 and 4.2.3. What follows is a short description of the characteristic parameters for the implemented algorithms:

- *iaolsp* and *inolsp* solve IVPs for ODEs by means of a piecewise-linearized approach and a block-oriented version with scaling-squaring implementation of the diagonal Padé approximation method:
  - Order (q) of the diagonal Padé approximation of the exponential function.
- *iaolwp* and *inolwp* solve IVPs for ODEs by means of a piecewise-linearized approach and a block-oriented version without scaling-squaring implementation of the diagonal Padé approximation method:
  - Order (q) of the diagonal Padé approximation of the exponential function.
- *iodbcs* solves IVPs for ODEs by means of a BDF method based on a Chord-Shamanskii iteration:
  - Order (r) of BDF method.
  - Relative error tolerance (tol_1) and absolute error tolerance (tol_2).
  - Maximum number of iterations without computing the Jacobian matrix (m).
  - Threshold (ρ).

As test battery five case studies were considered. The criteria to select these cases studies were:

- To solve physics and physical chemistry problems by the developed algorithms (case studies 1, 2, 3 and 5). Case study 4 has been selected because it has a known analytic solution.
- To prove the implementations on autonomous and non-autonomous ODEs: three IVPs for autonomous ODEs and two IVPs for non-autonomous ODEs have been selected.
- To compare the implementations when the ODE is stiff (cases studies 1, 2, 3 and 5) or non-stiff (case study 4).

Numerous tests were made (for each case study the characteristic parameters were varied, although only the parameters which offered better accuracy and lower computational cost for each algorithm are presented). Three kinds of tests are shown:

- Variable step size.
- Variable final time.

For each test, the following results are shown:

- Tables which contain the relative error

$$E_r = \frac{\|x - x^*\|_\infty}{\|x\|_\infty},$$

where x^* is the computed solution and x is the analytic solution (case study 4) or the solution computed by the MATLAB function ode15s with a vector of relative error tolerances $rtol = 10^{-13}$ and a vector of absolute error tolerances $atol = 10^{-13}$ [SGT03] (case studies 1, 2, 3 and 5).

- Tables with the execution time.

All algorithms were implemented in MATLAB and Fortran. The MATLAB implementations were tested on an Intel Core 2 Duo processor at 1.83 GHz with 2 Gb main memory, using MATLAB version 7.5. The tests for the larger dimension problem (case study 5) were carried out on a SGI Altix 3700 node [Gra03], with 1.3 GHz Intel Itanium II with 3 MB cache. In this case the algorithms were implemented in FORTRAN using the mathematical libraries BLAS [DCHH88] and LAPACK [ABB+92]. The implementations were compiled with Intel FORTRAN compiler (release 8.1) and SGI SCSL (Scientific Computing Software Library) mathematical library (release 1.5.1) was used. The SCSL is an optimized version of BLAS and LAPACK for SGI systems. The implemented algorithms are available online at [App]. The case studies considered in this work are presented below.

4.2.4.1. Case study 1 (Chemical Akzo Nobel problem)

This case study corresponds to the stiff autonomous ODE [LdS98] defined by

$$\dot{x} = f(x), x = x(t) \in \mathbb{R}^6, 0 \le t \le 180,$$
$$x(0) = [0.444,\ 0.00123,\ 0,\ 0,\ 0.007,\ 0.35999964]^T,$$

where

$$f(x) = \begin{bmatrix} -2r_1 + r_2 - r_3 - r_4 \\ -0.5r_1 - r_4 - 0.5r_5 + F_{in} \\ r_1 - r_2 + r_3 \\ -r_2 - r_3 - 2r_4 \\ r_2 - r_3 + 2r_5 \\ -r_5 \end{bmatrix},$$

and r_i, $i = 1, 2, 3, 4, 5$, and F_{in} are defined as

$$r_1 = k_1 x_1^4 x_2^{0.5},$$
$$r_2 = k_2 x_3 x_4,$$
$$r_3 = \frac{k_2}{K} x_1 x_5,$$
$$r_4 = k_3 x_1 x_4^2,$$
$$r_5 = k_4 x_6^2 x_2^{0.5},$$
$$F_{in} = klA\left(\frac{p(\mathrm{CO}_2)}{H} - x_2\right).$$

Er	Δt=0.1	Δt=0.05	Δt=0.01	Δt=0.005	Δt=0.001
iodbcs	2.576e-5	1.223e-5	8.658e-7	2.323e-7	9.823e-9
iaolsp	2.080e-5	5.238e-6	1.485e-7	3.851e-8	1.588e-9
iaolwp	8.100e-6	2.824e-6	1.485e-7	3.851e-8	1.588e-9

Tabla 4.2: Relative errors considering t_f=60 and varying Δt (case study 1)

The tests were carried out considering:

$$\begin{array}{ll} k_1 = 18.7, & K = 34.4, \\ k_2 = 0.58, & klA = 3.3, \\ k_3 = 0.09, & p(\mathrm{CO}_2) = 0.9 \\ k_4 = 0.42, & H = 737. \end{array}$$

This problem originates from Akzo Nobel Central Research in Arnhem. It describes a chemical process in which two species, FLB and ZHU, are mixed, while carbon dioxide is continuously added. The variables x_i correspond to the following concentrations: x_1 = [FLB], x_2 = [CO_2], x_3 = [FLBT], x_4 = [ZHU], x_5 = [ZLA] and x_6 = [ZLA.ZHU], where FLBT, ZLA and ZLA.ZHU are other species that appear in the chemical process.

The optimal values of characteristic parameters were:

- *iodbcs*: r=3, tol_1=tol_2=10^{-14}, m=2, ρ = 0.5.
- *iaolsp*: q=1.
- *iaolwp*: q=1.

The following tests were done:

- First test (Tables 4.2 and 4.3): t_f=60 and Δt variable.
- Second test (Table 4.4 and Figure 4.2): Δt=0.01 and t_f variable.

Conclusions for this case study:

- Considering the same step size, the implementations based on the piecewise-linearized approach have lower relative error and lower execution time than the implementation based on BDF approach.
- The implementation based on the piecewise-linearized method and on the diagonal Padé approach without scaling-squaring (*iaolwp*) has the shorter execution time.

Execution time (seconds)	Δt=0.1	Δt=0.05	Δt=0.01	Δt=0.005	Δt=0.001
iodbcs	0.196	0.253	1.275	2.545	11.43
iaolsp	0.087	0.153	0.745	1.493	7.462
iaolwp	0.058	0.114	0.570	1.138	5.692

Tabla 4.3: Execution time of the MATLAB implementations considering t_f=60 and varying Δt (case study 1)

Er	t_f=60	t_f=90	t_f=120	t_f=150	t_f=180
iodbcs	8.658e-7	5.388e-7	4.183e-7	3.607e-7	3.303e-7
iaolsp	1.485e-7	9.687e-8	7.838e-8	6.980e-8	6.546e-8
iaolwp	1.485e-7	9.687e-8	7.838e-8	6.980e-8	6.546e-8

Tabla 4.4: Relative errors considering Δt=0.01 and varying t_f (case study 1)

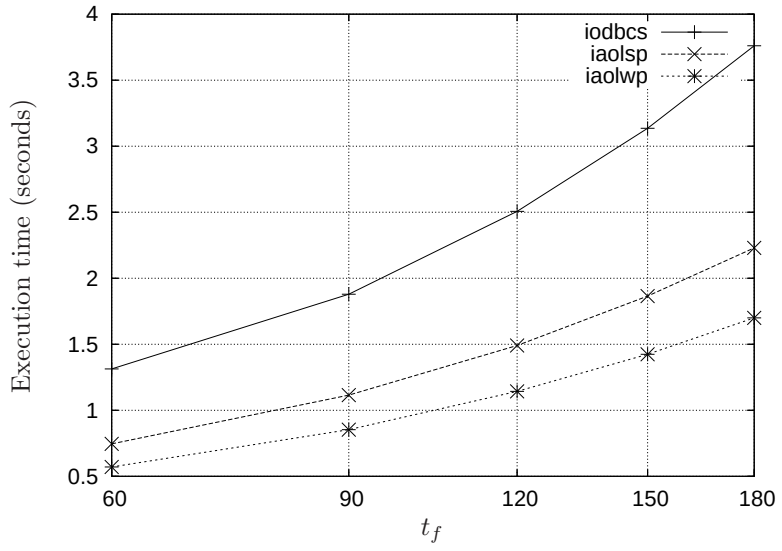


Fig. 4.2: Execution time of the MATLAB implementations considering Δt = 0.01 and varying t_f (case study 1)

Er	Δt=0.1	Δt=0.05	Δt=0.01	Δt=0.005	Δt=0.001
iodbcs	2.136e-4	5.279e-5	1.933e-6	4.767e-7	1.885e-8
iaolsp	4.185e-5	1.147e-5	4.8495e-7	1.219e-7	4.899e-9
iaolwp	4.183e-5	1.147e-5	4.8495e-7	1.219e-7	4.899e-9

Tabla 4.5: Relative errors considering t_f=50 and varying Δt (case study 2)

4.2.4.2. Case study 2 (HIRES problem)

This case study is presented in [LdS98] and it corresponds to the stiff IVP for ODEs defined by

$$\dot{x} = f(x), x = x(t) \in \mathbb{R}^8, 0 \leq t \leq 321.8122,$$
$$x(0) = [1, 0, 0, 0, 0, 0, 0, 0.0057]^T,$$

where

$$f(x) = \begin{pmatrix} -1.71x_1 + 0.43x_2 + 8.32x_3 + 0.0007 \\ 1.71x_1 - 8.75x_2 \\ -10.03x_3 + 0.43x_4 + 0.035x_5 \\ 8.32x_2 + 1.71x_3 - 1.12x_4 \\ -1.745x_5 + 0.43x_6 + 0.43x_7 \\ -280x_6x_8 + 0.69x_4 + 1.71x_5 - 0.43x_6 + 0.69x_7 \\ 280x_6x_8 - 1.81x_7 \\ -280x_6x_8 + 1.81x_7 \end{pmatrix}.$$

The name HIRES was given by Hairer and Wanner [HW96]. The HIRES problem explains the "High Irradiance Responses"(HIRES) of phytochrome-mediated photomorphogenesis by means of a chemical reaction involving eight reactants. The variables x_i are the concentrations of the eight reactants.

The optimal values of characteristic parameters were:

- *iodbcs*: r=3, tol_1=tol_2=10^{-14}, m=2, $\rho = 0.5$.
- *iaolsp*: $q = 2$.
- *iaolwp*: q=2.

The following tests were done:

- First test (Tables 4.5 and 4.6): t_f=50 and Δt variable.
- Second test (Table 4.7 and Figure 4.3): Δt=0.01 and t_f variable.

The conclusions obtained in this case are the same as for case study 1.

Execution time (seconds)	Δt=0.1	Δt=0.05	Δt=0.01	Δt=0.005	Δt=0.001
iodbcs	0.199	0.300	1.521	2.471	10.258
iaolsp	0.137	0.250	0.991	1.819	9.098
iaolwp	0.069	0.139	0.709	1.316	6.578

Tabla 4.6: Execution time of the MATLAB implementations considering t_f=50 and varying Δt (case study 2)

Er	t_f=100	t_f=150	t_f=200	t_f=250	t_f=300
iodbcs	2.294e-6	2.989e-6	4.276e-6	7.425e-6	2.406e-5
iaolsp	5.753e-7	7.496e-7	1.072e-6	1.862e-6	6.041e-6
iaolwp	5.753e-7	7.496e-7	1.072e-6	1.862e-6	6.041e-6

Tabla 4.7: Relative errors considering Δt=0.01 and varying t_f (case study 2)

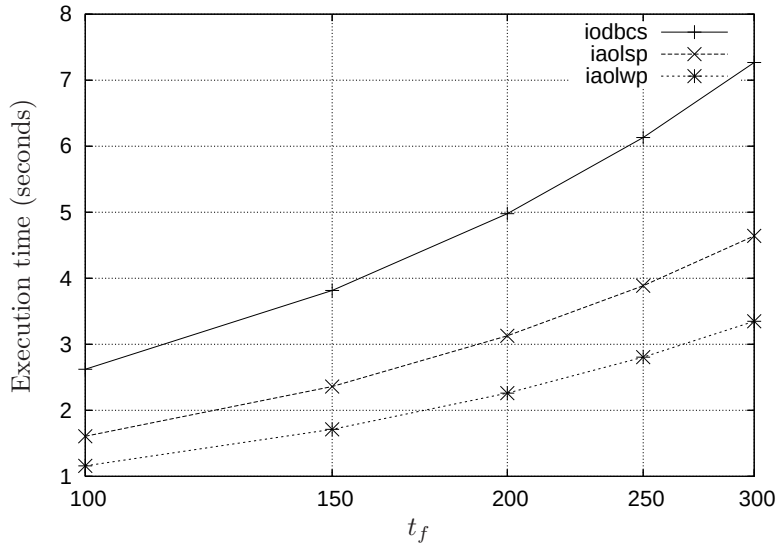


Fig. 4.3: Execution time of the MATLAB implementations considering Δt = 0.01 and varying t_f (case study 2)

4.2.4.3. Case study 3

This case study corresponds to the stiff autonomous ODE [SGT03, p. 29] defined by

$$\dot{x} = f(x),\ x = x(t) \in \mathbb{R}^3,\ 0 \le t \le 8 \cdot 10^5,$$
$$x(0) = [0, 1, 0]^T,$$

where

$$f([x_1, x_2, x_3]^T) = \begin{pmatrix} -k_1 x_1 + k_2 x_3 \\ -k_4 x_2 + k_3 x_3 \\ k_1 x_1 + k_4 x_2 - (k_1 + k_3) x_3 \end{pmatrix},$$

and

$$k_1 = 8.4303270 \cdot 10^{-10},\ k_2 = 2.9002673 \cdot 10^{11},$$
$$k_3 = 2.4603642 \cdot 10^{10},\ k_4 = 8.7600580 \cdot 10^{-6}.$$

This case study corresponds to the proton transfer hydrogen-hydrogen bond problem, where $x_1(t)$ and $x_2(t)$ are the solution components of the proton transfer and $x_3(t)$ is the quickly reacting intermediate component.

The optimal values of characteristic parameters were:

- *iodbcs*: r=2, tol_1=tol_2=10^{-14}, m=2, $\rho = 0.5$.
- *iaolsp*: $q = 1$.
- *iaolsp*: q=1.

The following tests were done:

- First test (Table 4.8 and Fig. 4.9): t_f=100 and Δt variable. Since with small values of Δt very high precisions have been reached, only $\Delta t = 0.1$ and $\Delta t = 0.05$ have been considered.
- Second test (Table 4.10 and Fig. 4.4): Δt=0.01 and t_f variable.

Conclusions for this case study:

- For the same step size, the relative errors committed by the implementation based on the piecewise-linearized method and on the diagonal Padé approach without scaling-squaring (*iaolwp*) have been minor in comparison to those committed by the other two implementations; also this implementation has the shortest execution time.
- The implementation based on the BDF method (*iodbcs*) has a lower relative error than the implementation based on the piecewise-linearized method and on the diagonal Padé approach with scaling-squaring (*iaolsp*), but execution time is longer.

Er	Δt=0.1	Δt=0.05
iodbcs	4.104e-14	8.202e-14
iaolsp	4.706e-12	2.358e-12
iaolwp	6.274e-15	6.065e-15

Tabla 4.8: Relative errors considering t_f=100 and varying Δt (case study 3)

Execution time (seconds)	Δt=0.1	Δt=0.05
iodbcs	0.199	0.300
iaolsp	0.137	0.250
iaolwp	0.069	0.139

Tabla 4.9: Execution time of the MATLAB implementations considering t_f=100 and varying Δt (case study 3)

Er	t_f=500	t_f=1000	t_f=1500	t_f=2000	t_f=2500
iodbcs	4.359e-12	8.787e-12	1.420e-12	1.995e-11	2.6616e-11
iaolsp	4.636e-12	9.071e-12	1.538e-12	2.130e-11	2.666e-11
iaolwp	3.838e-14	1.303e-13	1.927e-12	3.960e-12	5.814e-12

Tabla 4.10: Relative errors considering Δt=0.01 and varying t_f (case study 3)

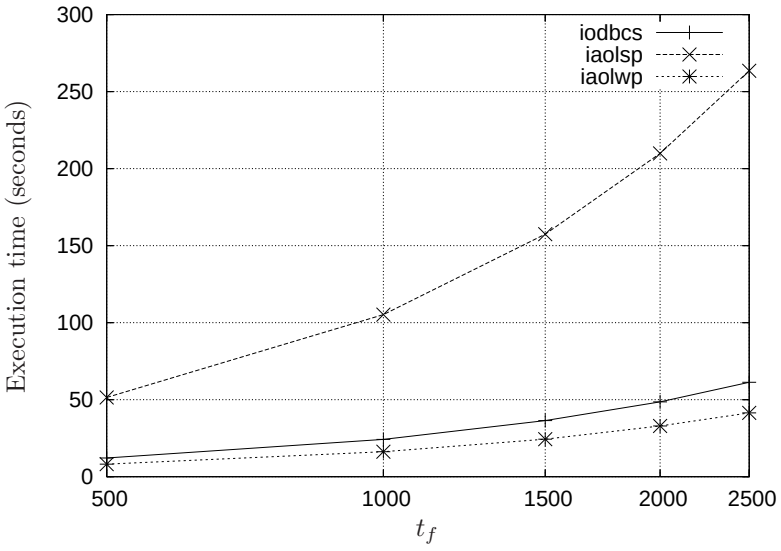


Fig. 4.4: Execution time of the MATLAB implementations considering Δt = 0.01 and varying t_f (case study 3)

Er	Δt=0.1	Δt=0.05	Δt=0.01	Δt=0.005	Δt=0.001
iodbcs	5.167e-06	1.171e-06	4.103e-08	1.009e-08	3.971e-10
inolsp	1.079e-15	1.447e-15	6.296e-15	2.268e-14	4.965e-14
inolwp	1.079e-15	1.447e-15	6.296e-15	2.268e-14	4.965e-14

Tabla 4.11: Relative errors considering t_f=10 and varying Δt (case study 4)

4.2.4.4. Case study 4

This case study corresponds to the non-stiff and non-autonomous ODE [RGL97] defined as

$$\dot{x}(t) = (t - x(t))^2 + 1,\ t \geq 3,$$
$$x(3) = 2.$$

The analytic solution is

$$x(t) = t + \frac{1}{2-t}.$$

The values chosen for the characteristic parameters were:

- *iodbcs*: r=2, tol_1=tol_2=10^{-12}, m=2, $\rho = 0.5$.
- *inolsp*: q=1.
- *inolwp*: q=1.

The following tests were done:

- First test (Tables 4.11 and 4.12): t_f=10 and Δt variable. Since the piecewise-linearized method presents a very small error for a step size equal to 0.1, we will show only the results obtained for that increase.
- Second test (Table 4.13 and Fig. 4.5): Δt=0.1 and t_f is variable.

Conclusions for this case study:

- For the same step size, the implementation based on the BDF method *iodbcs* shows more significant error than the implementations based on the piecewise-linearized method.
- The implementation based on the piecewise-linearized method and on the diagonal Padé approach without scaling-squaring (*inolwp*) has the shortest execution time.

Execution time (seconds)	Δt=0.1	Δt=0.05	Δt=0.01	Δt=0.005	Δt=0.001
iodbcs	0.015	0.027	0.071	0.121	0.609
inolsp	0.010	0.017	0.046	0.092	0.456
inolwp	0.008	0.015	0.040	0.080	0.400

Tabla 4.12: Execution time the MATLAB implementations for t_f=10 and Δt (case study 4)

Er	t_f=100	t_f=200	t_f=300	t_f=400	t_f=500
iodbcs	1.192e-08	1.460e-09	4.295e-10	1.807e-10	9.228e-11
inolsp	1.236e-14	1.904e-14	1.762e-14	5.032e-14	6.209e-14
inolwp	1.236e-14	1.904e-14	1.762e-14	5.032e-14	6.209e-14

Tabla 4.13: Relative errors considering Δt=0.1 and varying t_f (case study 4)

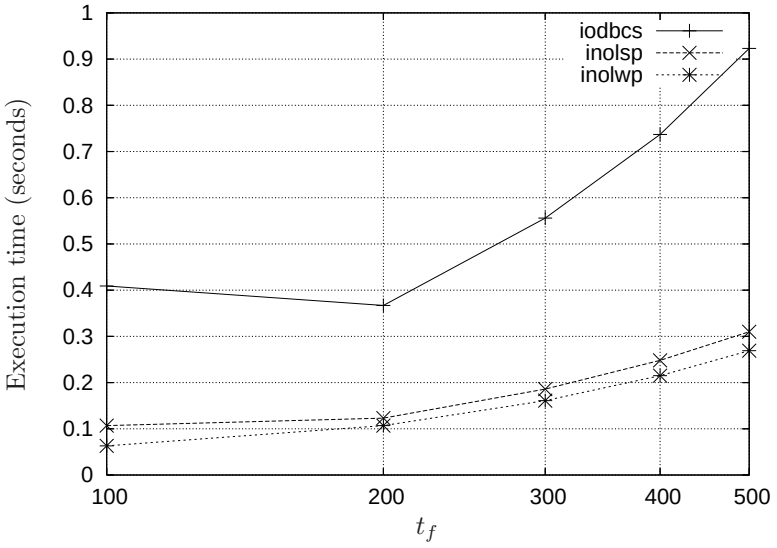


Fig. 4.5: Execution time of the MATLAB implementations considering Δt = 0.1 and varying t_f (case study 4)

4.2.4.5. Case study 5 (Medical Akzo Nobel problem)

This case study corresponds to a stiff non-autonomous ODE [LdS98] defined as

$$\dot{x} = f(t,x) \ , t \geq 0, x = x(t) \in \mathbb{R}^{2N}, \ 0 \leq t \leq 20,$$
$$x(0) = [0, v_0, 0, v_0, \cdots, 0, v_0] \in \mathbb{R}^{2N}, \ N = 200.$$

Function f is defined as follows

$$f_{2j-1} = \alpha_j \frac{x_{2j+1} - x_{2j-3}}{2\Delta\zeta} + \beta_j \frac{x_{2j-3} - 2x_{2j-1} + x_{2j+1}}{(\Delta\zeta)^2} - k x_{2j-1} x_{2j},$$
$$f_{2j} = -k x_{2j} x_{2j-1},$$

where

$$\alpha_j = \frac{2(j\Delta\zeta - 1)^3}{c^2},$$
$$\beta_j = \frac{(j\Delta\zeta - 1)^4}{c^2}.$$

The values of j are from 1 to$N$, $\Delta\zeta = \frac{1}{N}$,$x_{-1}(t) = \phi(t)$, $x_{2N+1} = x_{2N-1}$ and ϕ function is given by

$$\phi(t) = \begin{cases} 2, \ t \in (0,5] \\ 0, \ t \in (5,20] \end{cases}.$$

The Akzo Nobel research laboratories formulated this problem in their study of the penetration of radio-labeled antibodies into a tissue infected by a tumor. This study was carried out for diagnostic and therapeutic purposes. The desired results are the evolutions of the concentrations u and v of both elements (the antibodies and the tissue, respectively) along the discretized space in function of time. This problem can be formulated as the above ODE by using the lines method. In this formulation, the data vector is noted as x, where elements x_{2j-1}, $j = 1, \ldots, N$, represent concentrations u along the spatial dimension and elements x_{2j}, $j = 1, \ldots, N$, represent concentrations v. The chosen values of k, v_0 and c were 100, 1 and 4 respectively.

The optimal values of characteristic parameters were:

- *iodbcs*: r=2, tol_1=tol_2=10^{-14}, m=2, $\rho = 0.5$.
- *inolsp*: q=1.
- *inolwp*: q=1.

Fig. 4.6 summarizes the execution time of the Fortran implementation of the algorithms considered in this work, considering $\Delta t = 10^{-6}$ and varying t. The relative error in all algorithms was approximately equal to 10^{-6}. In this case, the implementation based on the piecewise-linearized method and on the diagonal Padé approach without scaling-squaring (*inolwp*) has the shortest execution time.

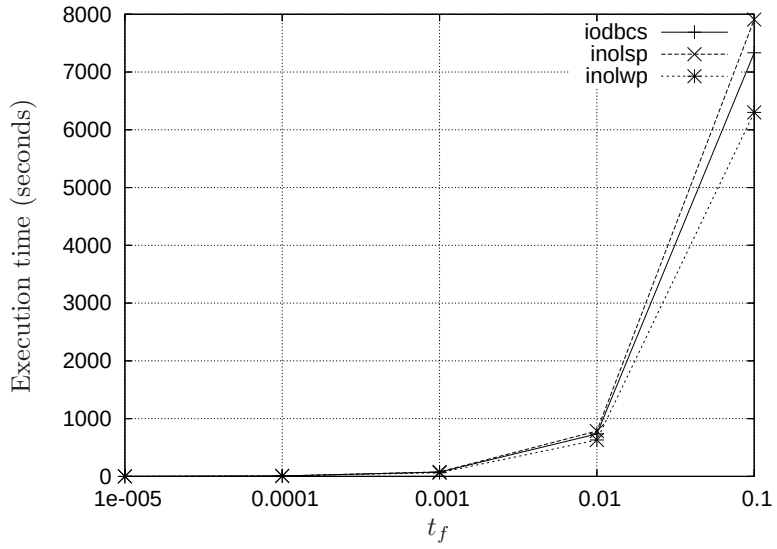


Fig. 4.6: Execution time of the Fortran implementations considering $\Delta t = 10^{-6}$ and varying t_f (case study 5)

4.2.5. Conclusions and future work

In this paper three methods for solving IVPs for ODEs have been developed and implemented. The first is a BDF method based on a Chord and Shamanskii iteration. The other two methods are based on the piecewise-linearized method and the diagonal Padé approximants. These methods are based on Theorem 10, which makes it possible to solve IVPs for ODEs. In addition, four algorithms, two for non-autonomous ODEs (*inolsp – inolwp*) and another two for autonomous ODEs (*iaolsp – iaolwp*), have been implemented. These algorithms have been compared to the BDF algorithm based on Chord-Shamanskii iteration (*idobcs*). According to experimental results, the algorithms based on the piecewise-linearized method and on the diagonal Padé approach without scaling-squaring (*iaolwp–inolwp*) behave better, both in terms of precision and computational costs, than the BDF algorithm (see Tables 4.14 and 4.15). Below is a summary of the five case studies:

1. The optimal order of the BDF algorithm varies between 1 and 3, depending on the case study.

2. The optimal order of algorithms based on diagonal Padé approximants varies between 1 (case studies 1, 3, 4 and 5) and 2 (case study 2).

3. Considering the same step size, the BDF algorithm (*idobcs*) is generally less accurate than the piecewise-linearized algorithms. The more accurate algorithms are based on the diagonal Padé approximants without scaling-squaring (*iaolwp* and *inolwp* algorithms). Also these algorithms have the lowest computational cost.

4. All implemented algorithms show good behavior in stiff ODEs.

Accuracy	1-A-S	2-A-S	3-A-S	4 NA-NS	5 NA-S
iodbcs	−	−		−	≅
iaolsp-inolsp		+	−	+	≅
iaolwp-inolwp	+	+	+	+	≅

Tabla 4.14: Comparative precision of implemented algorithms for the five case studies: The symbols +, − and ≅ indicate respectively greater, less and similar precision. A/NA denotes Autonomous/Non-Autonomous ODEs, and S/NS denotes whether the problem is Stiff or Non-Stiff.

Execution times	1-A-S	2-A-S	3-A-S	4 NA-NS	5 NA-S
iodbcs	+	+		+	
iaolsp-inolsp			+		+
iaolwp-inolwp	−	−	−	−	−

Tabla 4.15: Comparative execution times of the implemented algorithms for the five case studies: The symbols +, − indicate respectively greater and less execution time. A/NA denotes Autonomous/Non-Autonomous ODEs, and S/NS denotes whether the problem is Stiff or Non-Stiff.

As future work, new improvements will be developed, such as:

1. To consider adapting the methodology described here in order to obtain efficient resolution of ODEs with sparse Jacobian matrices.
2. To implement algorithms with error control in order to vary step size dynamically. The tests reported here considered constant step size. It is possible to improve the developed algorithms using a step size variable by estimating the error committed in each iteration [C. 03].
3. To do parallel implementation of the algorithms presented in this work in a distributed memory platform, using the message passing paradigm, MPI [GLS94] and BLACS [DG91] for communications, and PBLAS [CDO+95] and ScaLAPACK [BCC+97] for computations.

4.3. A Piecewise-linearized Algorithm based on Krylov Subspace for solving stiff ODEs [1]

Referencia del artículo:
J. J. Ibáñez, V. Hernández, P. Ruiz and E. Arias
A piecewise–linearized algorithm based on the Krylov subspace for solving stiff ODEs
Journal of Computational and Applied Mathematics, Volume 235, Issue 7, February 2011, Pages 1798–1804, ISSN 0377-0427, http://dx.doi.org/10.1016/j.cam.2010.07.012

Abstract

Numerical methods for solving Ordinary Differential Equations (ODEs) have received considerable attention in recent years. In this paper a piecewise-linearized algorithm based on Krylov subspaces for solving Initial Value Problems (IVPs) is proposed. MATLAB versions for autonomous and non-autonomous ODEs of this algorithm have been implemented. These implementations have been compared with other piecewise-linearized algorithms based on Padé approximants, recently developed by the authors of this paper, comparing both precisions and computational costs in equal conditions. Four case studies have been used in the tests that come from stiff biology and chemical kinetics problems. Experimental results show the advantages of the proposed algorithms, especially when the dimension is increased in stiff problems.

4.3.1. Introduction

Many scientific and engineering problems are described by ODEs where the analytic solution is unknown. In recent years many review articles and books have appeared on numerical methods for integrating stiff ODEs. Stiff problems are very common problems in many fields of the applied sciences: control theory, biology, chemical kinetics, electronic circuit theory, fluids, etc. There exist numerous one-step algorithms for solving stiff ODEs based on the implicit Runge-Kutta methods [HW96, But08, VAR06]. Another popular family of algorithms for solving these problems are the multistep algorithms based on the BDF method [CH52, Hin80, Gea71, BBH89]. In this paper we have developed a one-step method based on a piecewise-linearized method [RGL97]. These methods solve an IVP by approximating the right hand side of the corresponding ODE by a Taylor polynomial of degree 1. The resulting approximation can be integrated analytically to obtain the solution in each subinterval and yields the exact solution for linear problems. In [RGL97, GL98] an exhaustive study of this method is introduced. The proposed method requires a non-singular Jacobian matrix on each subinterval.

In [IHAR09] the authors presented a piecewise-linearized method for solving ODEs. This method uses a theorem proved in that article, which enables the approximate solution to be computed at each time step by a block-oriented approach based on diagonal Padé approximations. In this work another approach based on the piecewise-linearized method is

[1] This work has been supported by the Spanish CICYT project CGL2007-66440-C04-03.

introduced. In this case, the matrix-vector product $e^A v$, which appears in these methods, is computed by a Krylov subspace approach. The computational costs and precisions of the algorithms are compared in equal conditions. The paper is structured as follows. The new approach for solving ODEs based on the Krylov subspace approach is presented in Section 4.3.2. The experimental results are shown in Section 4.3.3. Finally, conclusions and future expectations are given in Section 4.3.4.

4.3.2. A piecewise-linearized algorithm for solving ODEs based on the Krylov subspace approach

In [IHAR09] the authors presented a piecewise-linearized method for solving ODEs, based on the following theorem which enables the approximate solution to be computed at each time step by a block-oriented approach based on diagonal Padé approximations.

Theorem 11 ([IHAR09]) *Let*

$$\dot{x}(t) = f(t, x(t)), t \in [t_0, t_f], \tag{4.22}$$

be an ODE with initial value

$$x(t_0) = x_0 \in \mathbb{R}^n,$$

such that the first-order partial derivatives of $f(t,x)$ are continuous on $[t_0, t_f]\times\mathbb{R}^n$. Given a mesh $t_0 < t_1 < \cdots < t_{l-1} < t_l = t_f$, ODE (4.22) can be approximated by a set of LDEs obtained as a result of a linear approximation of $f(t,x(t))$ on each subinterval ([GL98, C. 03]),

$$\begin{aligned} \dot{y}(t) &= f_i + J_i(y(t) - y_i) + g_i(t - t_i), t \in [t_i, t_{i+1}], \\ y(t_i) &= y_i, \ i = 0, 1, \ldots, l-1. \end{aligned} \tag{4.23}$$

The solution of (4.23) is

$$y(t) = y_i + E_{12}^{(i)}(t - t_i) f_i + E_{13}^{(i)}(t - t_i) g_i, \tag{4.24}$$

where $E_{12}^{(i)}(t-t_i)$ and $E_{13}^{(i)}(t-t_i)$ are blocks $(1,2)$ and $(1,3)$ of $E = e^{C_i(t-t_i)}$, where

$$C_i = \begin{bmatrix} J_i & I_n & 0_n \\ 0_n & 0_n & I_n \\ 0_n & 0_n & 0_n \end{bmatrix}. \qquad \square$$

If t is replaced by t_{i+1} in (4.24), the approximate solution of ODE (4.22) at t_{i+1}, $i = 0, 1, \ldots, l-1$, is given by

$$y_{i+1} = y_i + E_{12}^{(i)}(\Delta t_i) f_i + E_{13}^{(i)}(\Delta t_i) g_i, \ \Delta t_i = t_{i+1} - t_i. \tag{4.25}$$

In this work, another approach based on the piecewise-linearized method is introduced as follows.

From [IHAR09, p. 716], $e^{C_i \Delta t_i}$ can be expressed as

$$\begin{bmatrix} e^{J_i \Delta t_i} & E_{12}^{(i)}(\Delta t_i) & E_{13}^{(i)}(\Delta t_i) \\ 0_n & I_n & I_n \Delta t_i \\ 0_n & 0_n & I_n \end{bmatrix},$$

whereas the approximate solution y_{i+1} given in (4.25) can be obtained by adding to y_i the first n components of a vector

$$e^{C_i \Delta t_i} v_i, \tag{4.26}$$

where

$$C_i = \begin{bmatrix} J_i & I_n & 0_n \\ 0_n & 0_n & I_n \\ 0_n & 0_n & 0_n \end{bmatrix}, v_i = \begin{bmatrix} 0_{n\times 1} \\ f_i \\ g_i \end{bmatrix}.$$

The matrix-vector product $e^{C_i \Delta t_i} v_i$ can be obtained by a Krylov subspace method [Saa92, Sid98]. Given $A \in \mathbb{R}^{n\times n}$ and $v \in \mathbb{R}^n$, it is possible to compute an approximation to vector $e^A v$ by using the Arnoldi method. This approximation is given by

$$e^A v \cong v_{opt} = \beta V_p e^{H_p} e_1, \tag{4.27}$$

where $H_p = (h_{ij}) \in \mathbb{R}^{p\times p}$ is the Hessenberg matrix obtained from the Arnoldi method and $V_p = [v_1, v_2, \ldots, v_p] \in \mathbb{R}^{n\times p}$, with $\{v_i\}_{i=1,2,\ldots,p}$ an orthonormal basis of the Krylov subspace $K_p = \text{span}\{v, Av, \ldots, A^{p-1}v\}$, $\beta = \|v\|_2$ and $e_1 = [1, 0, \cdots, 0]^T$.

In order to reduce computational and storage costs when we want to compute vector y_{i+1}, it is necessary to modify the classical Arnoldi algorithm without explicitly forming the matrix $C_i \Delta t_i$. Algorithm 4.13 solves IVPs for non-autonomous ODEs by the above piecewise-linearized method based on a Krylov subspace approach. This algorithm uses Algorithm 4.14 , which computes the approximate solution at t_{i+1} of IVP (4.22) for non-autonomous ODEs, obtained after the piecewise-linearized process, by a block-oriented implementation of the Krylov subspace approach. Its computational cost is $2n^2p + 6np(p+1) + 2(q + j_{H_p} + 1/3)p^3$ flops, where $j_{H_p} = \text{máx}(0, 1 + int(\log_2(\|H_p\|)))$. It is possible to reduce the computational and storage costs of Algorithm 4.13 when IVP (4.22) is autonomous.

4.3.3. Experimental results

The main objective of this section is to compare the MATLAB implementations of algorithm developed in Section 2 with the implementations developed by the authors of this paper in [IHAR09].

As test battery, four case studies of stiff ODEs, which come from biology and chemical kinetics problems, were considered. Numerous tests were made on them. For each case study and algorithm, the characteristic parameters were varied, although only the parameters which offered the same accuracy for the two implementations with the lower computational cost are presented.

Algorithm 4.13 Solves IVP (4.22) by a piecewise-linearized method based on a Krylov subspace approach.

Function $y = \texttt{inolkr}(t, \texttt{data}, x_0, p, tol, q)$
Inputs: Time vector $t \in \mathbb{R}^{l+1}$; function `data` computes $f(\tau, y) \in \mathbb{R}^n$, $J(\tau, y) \in \mathbb{R}^{n\times n}$ and $g(\tau, y) \in \mathbb{R}^n$ $(\tau \in \mathbb{R},\ y \in \mathbb{R}^n)$; vector $x_0 \in \mathbb{R}^n$; dimension $p \in \mathbb{N}$ of the Krylov subspace; tolerance $tol \in \mathbb{R}^+$; order $q \in \mathbb{N}$ of the diagonal Padé approximation of the exponential function
Output: Matrix $Y = [y_1, \ldots, y_l] \in \mathbb{R}^{n\times l}$, $y_i \in \mathbb{R}^n$, $i = 1, 2, \ldots, l$
1: Compute the vectors c_1 and c_2 that contain the coefficients of terms of degree greater than 0 in the diagonal Padé approximation of the exponential function
2: $y_0 = x_0$
3: **for** $i = 0 : l - 1$ **do**
4: $\quad [J_i, f_i, g_i] = \texttt{data}(t_i, y_i)$
5: $\quad \Delta t_i = t_{i+1} - t_i$
6: $\quad y_{i+1} = \texttt{inlbkr}(J_i, f_i, g_i, y_i, \Delta t_i, p, tol, c_1, c_2)$ (Algorithm 4.14)
7: **end for**

What follows is a short description of the implemented algorithms and the characteristic parameters:

- `iaolwp` and `inolwp` solve IVPs for ODEs by a piecewise-linearized approach and a block-oriented version without scaling-squaring implementation of the diagonal Padé approximation method:
  - Order $q = 2$ of the diagonal Padé approximation of the exponential function.
- `iaolkr` and `inolkr` solve IVPs for ODEs by a piecewise-linearized method based on Krylov subspaces:
  - Dimension $p = 4$ of the Krylov subspace. In Ref. [Saa92] there is an exhaustive study of the computation of the product of the exponential of a matrix and a vector by using Krylov subspaces. We have proved experimentally that when considering low or medium dimension matrices, it is only necessary to consider a very much reduced subspace dimension. In this work $p = 4$.
  - Tolerance $tol = 10^{-6} \in \mathbb{R}^+$.
  - Order $q = 2$ of the diagonal Padé approximation of the exponential function.

For each test, the following results are shown:

- Tables which contain the relative error

$$E_r = \frac{\|x - x^*\|_\infty}{\|x\|_\infty},$$

where x^* is the computed solution and x is the analytic solution (case study 2) or the solution computed by the MATLAB function ode15s with a vector of relative error tolerances $rtol = 10^{-13}$ and a vector of absolute error tolerances $atol = 10^{-13}$ [SGT03].

Algorithm 4.14 Computes the approximate solution at t_{i+1} of IVP (4.22) for non-autonomous ODEs, obtained after the piecewise-linearized process, by a block-oriented implementation of the Krylov subspace approach.

Function $y_{i+1} = \texttt{inlbkr}(J_i, f_i, g_i, y_i, \Delta t_i, p, tol, c_1, c_2)$
Inputs: Matrix $J_i \in \mathbb{R}^{n\times n}$; vector $f_i \in \mathbb{R}^n$; vector $g_i \in \mathbb{R}^n$; vector $y_i \in \mathbb{R}^n$; step size $\Delta t_i \in \mathbb{R}$; dimension $p \in \mathbb{N}$ of the Krylov subspace; tolerance $tol \in \mathbb{R}^+$; vectors $c_1, c_2 \in \mathbb{R}^q$ with the coefficients of terms of degree greater than 0 in the diagonal Padé approximation of the exponential function
Output: Vector $y_{i+1} \in \mathbb{R}^n$ given by expression (4.26)

$V(1:n,1) = 0_n$
$V(n+1:2n,1) = f_i$
$V(2n+1:3n,1) = g_i$
$\beta = \|V(n+1:3n,1)\|_2$
if $\beta == 0$ **then**
    $y_{i+1} = y_i$
    Return
end if
$V(n+1:3n,1) = V(n+1:3n,1)/\beta$
for $j = 1:p$ **do**
    $w(1:n) = J_i V(1:n,j) + V(n+1:2n,j)$
    $w(n+1:2n) = V(2n+1:3n,j)$
    $w(1:2n) = \Delta t_i w(1:2n)$
    $w(2n+1:3n) = 0_n$
    for $i = 1:j$ **do**
        $H(i,j) = w^T V(1:3n,i)$
        $w = w - H(i,j)V(1:3n,i)$
    end for
    $s = \|w\|_2$
    if $s < tol$ **then**
        $p = j$
        Leave for loop
    end if
    $H(j+1,j) = s$
    $V(1:3n,j+1) = w/s$
end for
computes $E = e^{H_p}$
$y_{i+1} = y_i + \beta V(1:n,1:p)E(1:p,1)$

E_r	Δt=0.1	Δt=0.05	Δt=0.01	Δt=0.005	Δt=0.001
`iaolwp`	2.809e-04	7.523e-05	2.390e-06	5.840e-07	2.366e-08
`iaolkr`	2.348e-04	6.928e-05	2.759e-06	6.423e-07	2.399e-08

Tabla 4.16: Relative error (E_r) with t = 10 and Δt variable (case study 1).

T_e	Δt=0.1	Δt=0.05	Δt=0.01	Δt=0.005	Δt=0.001
`iaolwp`	0.014	0.021	0.114	0.257	6.231
`iaolkr`	0.025	0.048	0.201	0.428	6.802

Tabla 4.17: Execution time (T_e) in seconds with t = 10 and Δt variable (case study 1).

- Tables/figures with the execution time.

The algorithms were implemented in MATLAB 7.9 and tested on an Intel Core 2 Duo processor at 2.66 GHz with 2 GB main memory. Several tests have been developed in order to determine the accuracy and efficiency of the algorithms. The implemented algorithms are available online at http://www.grycap.upv.es/odelin.

4.3.3.1. Case study 1 (the pollution problem [LdS98])

This case study corresponds to a stiff IVP of dimension 20. The problem describes a chemical process consisting of 25 reactions and 20 species. The following tests were done:

- First test (Tables 4.16 and 4.17): t=10 and Δt variable.
- Second test (Table 4.18 and Fig. 4.7): Δt=0.01 and t variable.

4.3.3.2. Case study 2 (the EMEP problem [LdS98])

In this case study a stiff IVP for ODEs of dimension 66 is solved. The problem describes a problem which consists of 66 chemical species and about 140 reactions. The following tests were done:

- In the first test t=14450 was considered. With Δt = 0.1 the relative errors of the three implementations were equal to $2.219 \cdot 10^{-15}$, with executions times equal to 1.290 (`inolwp`) and 0.266 (`inolkr`) seconds.
- Second test (Tables 4.19 and 4.20, and Fig. 4.8): Δt=0.1 and t variable.

E_r	t=20	t=30	t=40	t=50	t=60
`iaolwp`	2.015e-06	1.744e-06	1.537e-06	1.374e-06	1.240e-06
`iaolkr`	2.327e-06	2.013e-06	1.775e-06	1.585e-06	1.431e-06

Tabla 4.18: Relative error (E_r) Δt = 0.01 and t variable (case study 1).

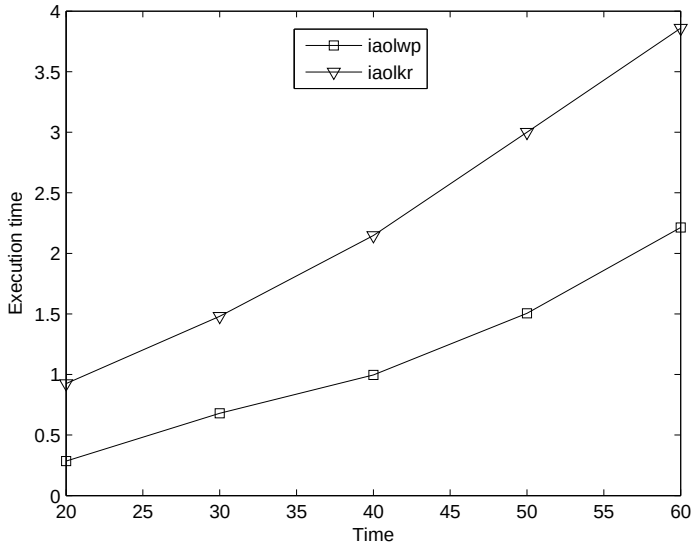


Fig. 4.7: Execution time in seconds of the MATLAB implementations considering $\Delta t = 0.01$ and varying t (case study 1).

E_r	t = 15400	t = 16400	t = 17400	t = 18400	t = 19400
`inolwp`	4.410e-14	8.833e-14	1.431e-13	1.980e-13	2.528e-13
`inolkr`	4.410e-14	8.833e-14	1.431e-13	1.980e-13	2.528e-13

Tabla 4.19: Relative error (E_r) with $\Delta t = 0.1$ and t variable (case study 2).

T_e	t = 15400	t = 16400	t = 17400	t = 18400	t = 19400
`inolwp`	52.830	157.465	316.257	529.469	790.217
`inolkr`	26.547	102.401	227.630	400.672	623.248

Tabla 4.20: Execution time (T_e) in seconds with $\Delta t = 0.1$ and t variable (case study 2).

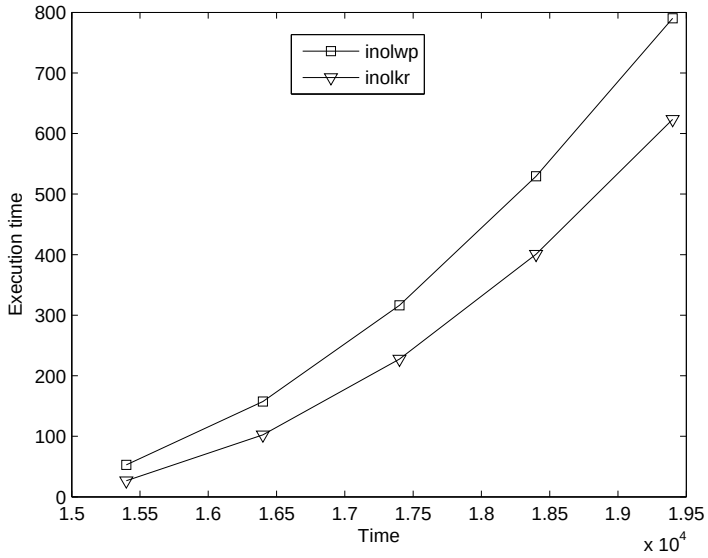


Fig. 4.8: Execution time in seconds of the MATLAB implementations considering $\Delta t = 0.1$ and varying t between 15400 and 19400 (case study 2).

E_r	Δt=0.01	Δt=0.001	Δt=0.0001	Δt=0.00001
`inolwp`	1.572e-02	1.726e-03	1.741e-04	1.742e-05
`inolkr`	1.663e-02	1.728e-03	1.741e-04	1.742e-05

Tabla 4.21: Relative error (E_r) considering $n = 100$, $t = 1$ and Δt variable (case study 3).

4.3.3.3. Case study 3 (the Medical Akzo Nobel problem [LdS98])

This case study corresponds to a stiff non-autonomous ODE [LdS98] of variable dimension $2N$. This problem studies the penetration of radio-labeled antibodies into tissue infected by a tumor.

The following tests were made:

- First test (Tables 4.21 and 4.22): $n = 100$ ($N = 50$), t=1 and Δt variable.
- Second test (Tables 4.23 and 4.24): Δt=0.001, $t = 1$ and varying n from 50 to 250 ($N = 25$ to 125).

T_e	Δt=0.01	Δt=0.001	Δt=0.0001	Δt=0.00001
`inolwp`	0.301	4.926	144.484	6362.490
`inolkr`	0.036	0.538	50.044	5263.304

Tabla 4.22: Execution time (T_e) in seconds considering $n = 100$, $t = 1$ and Δt variable (case study 3).

E_r	n=50	n=100	n=150	n=200	n=250
`inolwp`	1.636e-03	1.726e-03	1.746e-03	1.743e-03	1.736e-03
`inolkr`	1.637e-03	1.728e-03	1.752e-03	1.763e-03	1.781e-03

Tabla 4.23: Relative error (E_r) considering $\Delta t = 0.001$, $t = 1$ and n variable (case study 3).

T_e	n=50	n=100	n=150	n=200	n=250
`inolwp`	0.720	3.531	20.863	63.944	143.920
`inolkr`	0.288	0.482	0.740	1.159	1.367

Tabla 4.24: Execution time (T_e) in seconds considering $\Delta t = 0.001$, $t = 1$ and n variable (case study 3).

4.3.3.4. Case study 4 (the Brusselator problem) [HW96, pp. 6]

This case study corresponds to a stiff non-autonomous ODE of variable dimension N. This problem comes from chemical kinetics where the model of Lefever and Nicolis [LN71] is used and the method of lines is applied on a grid of N points:

- First test (Tables 4.25 and 4.26): $n = 100$ ($N = 50$), t=1 and Δt variable.
- Second test (Tables 4.27 and 4.28): $t = 1$, Δt=0.001 and n variable.

4.3.4. Conclusions and future work

In this work a new piecewise-linearized approach for solving ODEs based on Krylov subspaces has been presented. Two algorithms based on this approach (`inolkr` and `iaolbk`) have also been proposed and compared to the piecewise-linearized algorithms `iaolwp` and `inolwp` based on Padé approximants developed by the authors of this paper in [IHAR09].

Numerous test have been made on four case studies that come from stiff biology and chemical kinetics problems. Experimental results show the advantages of the proposed algorithms, especially when they are integrating stiff problems. According to the experimental results, the new algorithms offer in general similar precision and smaller computational cost when the problem size is increased. For example, Algorithm 4.13 (`inolkr`) was up to 111 times faster than `inolwp` for $n = 250$ and $t = 1$ in case study 3. This is because in the new approach the vector $e^A v$, $A \in \mathbb{R}^{n \times n}, v \in \mathbb{R}^n$, is approximated by the expression $\beta V_p e^{H_p} e_1$, where $p << n$. Nevertheless, when the problems are of small dimension, computational costs of piecewise-linearized algorithms based on diagonal Padé approximants are smaller than the computational costs of piecewise-linearized algorithms based on Padé approximants. In general, all algorithms offer accuracy and good behaviour with stiff problems.

E_r	Δt=0.01	Δt=0.001	Δt=0.0001	Δt=0.00001
`inolwp`	2.162e-02	3.673e-04	3.715e-05	3.719e-06
`inolkr`	2.263e-02	3.672e-04	3.715e-05	3.719e-06

Tabla 4.25: Relative error (E_r) considering $n = 100$, $t = 1$ and Δt variable (case study 4).

T_e	Δt=0.01	Δt=0.001	Δt=0.0001	Δt=0.00001
`inolwp`	0.140	1.688	55.297	4106.738
`inolkr`	0.031	0.465	38.122	3710.951

Tabla 4.26: Execution time (T_e) in seconds considering $n = 100$, $t = 1$ and Δt variable (case study 4).

E_r	n=50	n=100	n=150	n=200	n=250
`inolwp`	5.033e-04	3.673e-04	3.308e-04	3.170e-04	3.108e-04
`inolkr`	5.033e-04	3.672e-04	3.307e-04	3.169e-04	3.107e-04

Tabla 4.27: Relative error (E_r) considering $\Delta t = 0.001$, $t = 1$ and n variable (case study 4).

T_e	n=50	n=100	n=150	n=200	n=250
`inolwp`	0.521	1.894	2.988	7.616	16.391
`inolkr`	0.279	0.506	0.701	0.963	1.256

Tabla 4.28: Execution time (T_e) in seconds considering $\Delta t = 0.001$, $t = 1$ and n variable (case study 4).

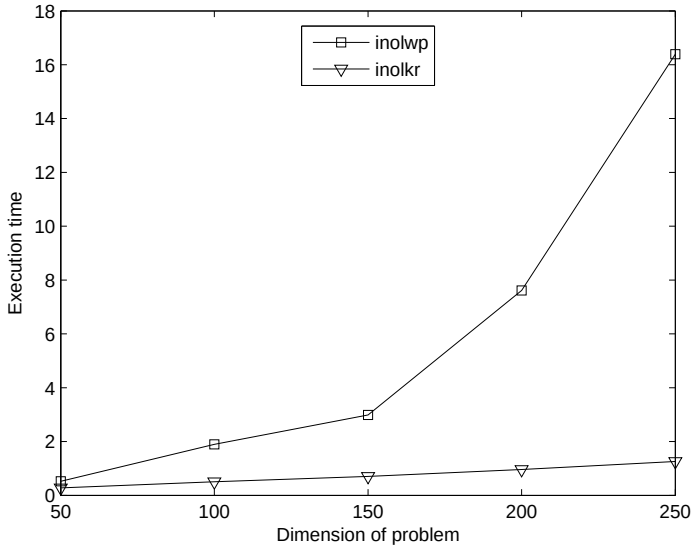


Fig. 4.9: Execution time in seconds of the MATLAB implementations considering $\Delta t = 0.001$ and varying t between 50 and 250 (case study 4).

As future work new improvements will be developed such as:

1. Implementing algorithms based on the piecewise-linearized approach with error control in order to vary the step size dynamically. The tests reported here considered constant step size. It is possible to improve the algorithms developed, using a variable step size which can be used to estimate the error committed in each iteration [GL98].

2. Carrying out parallel implementation of the algorithms presented in this work in a distributed memory platform, using the message passing paradigm MPI [GLS94] and BLACS [DG91] for communications, and PBLAS [CDO+95] and ScaLAPACK [BCC+97] for computations.

Bibliografía

[ABB+92] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM, 1992.

[AP98] U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, 1998.

[App] http://www.grycap.upv.es/odelin/.

[BBH89] P. N. Brown, G. D. Byrne, and A. C. Hindmarsh. Vode: A variable-coefficient ode solver. *SIAM J. Sci. Statist. Comput.*, 10:1038–1051, 1989.

[BCC+97] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, and I. Dhillon. *ScaLAPACK Users' Guide*. SIAM, 1997.

[But08] J. C. Butcher. *Numerical Methods for Ordinary Differential, Second Edition*. Wiley, 2008.

[C. 03] C. M. García-López. Piecewise-linearized and linearized θ-methods for ordinary and partial differential equation problems. *Computer & Mathematics with Applications*, 45:351–381, 2003.

[CDO+95] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, and D. Walker. A proposal for a set of parallel basic linear algebra subprogram. Technical Report UT-CS-95-292, Department of Computer Science, University of Tennessee, 1995.

[CH52] C. F. Curtiss and J. O. Hirschfelder. Integration of stiff equations. In *Proc. Nat. Acad. Sci.*, volume 38, pages 235–243, 1952.

[DCHH88] J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN Basic Linear Algebra Subroutines. *ACM Transactions on Mathematical Software*, 14:1–17, 1988.

[DG91] J. J. Dongarra and R. A. Van De Geijn. Two dimensional basic linear algebra communications subprograms. Technical report, Department of Computer Science, University of Tennessee, 1991.

[Gea71] C. W. Gear. Algorithm 407-difsub for solution of ordinary differential equations. *Comm. ACM*, 14:185–190, 1971.

[GL96] Gene H. Golub and Charles Van Loan. *Matrix Computations*. Johns Hopkins Studies in Mathematical Sciences. The Johns Hopkins University Press, third edition, 1996.

[GL98] C. M. García-López. *Métodos de Linealización para la Resolución Numérica de Ecuaciones Diferenciales*. PhD thesis, Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, 1998.

[GLS94] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 1994.

[Gra03] Silicon Graphics. *SGI Altix Applications Development an Optimization*, Release August 1st edition, 2003.

[Hig04] N. J. Higham. The scaling and squaring method for the matrix exponential revisited. Technical Report 452, Manchester Centre for Computational Mathematics, 2004.

[Hin80] A. C. Hindmarsh. LSODE and LSODI, two new initial value ordinary differential equation solvers. *ACM-Signum Newslett.*, 15:10–11, 1980.

[HW96] E. Hairer and G. Wanner. Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems. In *Springer Series in Computational Mathematics*, volume 14. Springer-Verlag, 1996.

[IHAR09] J. Ibáñez, V. Hernández, E. Arias, and P. Ruiz. Solving initial value problems for ordinary differential equations by two approaches: BDF and piecewise-linearized methods. *Computer Physics Communications*, 180(5):712–723, 2009.

[IHRA11] J. Javier Ibáñez, Vicente Hernández, Pedro A. Ruiz, and Enrique Arias. A piecewise-linearized algorithm based on the krylov subspace for solving stiff odes. *Journal of Computational and Applied Mathematics*, 235(7):1798 – 1804, 2011. Advances in Computational and Mathematical Methods in Science and Engineering.

[Kel95] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. SIAM, Philadelphia, 1995.

[LdS98] W. M. Lioen and J. J. B. de Swart. Test set for initial value problems solvers. Release 2.0, December 1998.

[LN71] R. Lefever and G. Nicolis. Chemical instabilities and sustained oscillations. *J. Theor. Biol*, 30:267–284, 1971.

[ML03] C. B. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*. *SIAM Review*, 45:3–49, 2003.

[RGL97] J. I. Ramos and C. M. García-López. Piecewise-linearized methods for initial-value problems. *Applied Mathematics and Computation*, 82:273–302, 1997.

[Saa92] Y. Saad. Analysis of some Krylov subspace approximations to the matrix exponential operator. *SIAM Journal on Numerical Analysis*, 29:209–228, 92.

[SGT03] L. S. Shampine, I. Gladwell, and S. Thomson. *Solving ODEs with MAtlab*. Cambridge University Press, 2003.

[Sid98] R. B. Sidje. Expokit: a software package for computing matrix exponentials. *ACM Trans. Math. Software*, 24:130–156, 1998.

[VAR06] Jesús Vigo-Aguiar and Higinio Ramos. A new eighth-order a-stable method for solving differential systems arising in chemical reactions. *Journal of Mathematical Chemistry*, 40(1):71–83, 2006.

Capítulo 5

Conclusiones

En el marco de esta tesis se han desarrollado nuevos métodos, algoritmos y *software* para el cálculo de funciones matriciales. El mayor esfuerzo se ha invertido en el cálculo de la función exponencial, donde poco a poco, a lo largo del proceso investigador, se han ido mejorando los métodos y algoritmos implementados, hasta alcanzar un nivel de precisión y eficiencia que supera en la mayoría de los casos a cualquier otro método del estado del arte. Todo este esfuerzo ha dado lugar a numerosas publicaciones, algunas de las cuales han conformado el segundo capítulo de esta memoria.

Los métodos utilizados para el cálculo de la exponencial están basados en series de polinomios matriciales, haciendo especial hincapié en aproximaciones de Taylor, aunque también en series matriciales de Hermite. Estos métodos, al contrario de lo que se pensaba, han demostrado ser igual de buenos, y en muchos casos mejores en cuanto a precisión y prestaciones, que los métodos más utilizados tradicionalmente, basados principalmente en aproximaciones racionales de Padé.

Además de la función exponencial, también se han desarrollado nuevos métodos y algoritmos para el cálculo de las funciones seno y coseno, de nuevo utilizando las series de Taylor y Hermite. Los resultados han sido muy satisfactorios, obteniendo mejores prestaciones que los algoritmos del estado del arte, basados también en aproximaciones de Padé. Las publicaciones más significativas a las que ha dado lugar esta parte de la investigación se pueden ver en el tercer capítulo de la tesis.

Por último, en el cuarto capítulo se presentan dos artículos en los que se presentan nuevas aproximaciones para la resolución de problemas de valor inicial, dentro del campo de la resolución de ecuaciones diferenciales matriciales. En estas aproximaciones se hace necesario el cálculo de la exponencial matricial, para lo cual se utilizaron métodos basados en aproximantes diagonales de Padé, puesto que en aquel momento todavía no estaban suficientemente desarrollados los nuevos algoritmos basados en Taylor y Hermite que se han presentado en el capítulo 2.

Cabe destacar que todo el *software* implementado que se ha presentado en los diversos artículos ha sido puesto a disposición de la comunidad científica internacional en la web del

grupo http://hipersc.blogs.upv.es/hipersc/. Para el desarrollo de este *software* se ha seguido la siguiente metodología:

- En primer lugar se analizan teóricamente los diversos algoritmos posibles para el cálculo de la función matricial objetivo, determinando a priori los mejores candidatos en función de tres parámetros: eficiencia, precisión y estabilidad numérica.
- Posteriormente se implementan estos algoritmos en MATLAB, lo cual permite comprobar la precisión y estabilidad numérica de los algoritmos seleccionados, y también nos permite hacernos una idea del nivel de eficiencia que podrá tener el nuevo algoritmo.
- El siguiente paso consiste en la adaptación e implementación de dichos algoritmos en los lenguajes C y/o Fortran bajo los distintos paradigmas de la computación de altas prestaciones con el objetivo de conseguir la máxima eficiencia:
  - En el caso de programación secuencial, será necesaria la adaptación del código para poder sacar el máximo provecho de las librerías numéricas de altas prestaciones como son BLAS y LAPACK y conseguir de esta manera una rutina secuencial de gran eficiencia.
  - En el caso de memoria compartida, se utiliza el entorno OpenMP junto con BLAS y LAPACK para poder aprovechar las capacidades multinúcleo y multihilo de los procesadores actuales.
  - En memoria distribuida se hace uso de las librerías numéricas PBLAS y SCALAPACK. Estas librerías son un estándar ampliamente utilizado en la programación en memoria distribuida, ya que proporcionan rutinas altamente optimizadas y se encuentran disponibles para la mayor parte de las arquitecturas paralelas con memoria distribuida. Durante la realización de la tesis se han realizado implementaciones para memoria distribuida de varios de los nuevos algoritmos desarrollados con resultados bastante buenos. Sin embargo, al no dar lugar a ninguna publicación, no se han incluido en esta memoria.

Actualmente, dentro del grupo de investigación al que está vinculado el doctorado, se sigue trabajando en el desarrollo de métodos aún más eficientes para el cálculo de funciones matriciales, obteniéndose algoritmos cada vez más rápidos y eficientes. Se ha desarrollado un nuevo método de evaluación de polinomios matriciales que resulta ser más eficiente en cuanto a coste computacional que el método de Paterson–Stockmeyer, el cual ha sido aplicado con éxito en la computación de funciones de matrices. También se están adaptando los códigos para el trabajo con unidades de procesamiento gráfico de propósito general (GPGPUs) y poder aprovechar la potencia de cálculo que presentan estos dispositivos actualmente.

Como objetivos próximos se incluyen la ampliación de los conocimientos adquiridos durante esta investigación a otras funciones matriciales como son el logaritmo o la raíz n–ésima de una matriz, y la creación de librerías secuenciales y paralelas con las principales rutinas implementadas, tanto en MATLAB como en C/Fortran. Resulta claro que este tema despierta mucho interés actualmente, como demuestra la gran cantidad de publicaciones de alta calidad que genera y el interés despertado por dichas publicaciones dentro del ámbito científico. Estas librerías vendrían a suplir la carencia de *software* libre eficiente y preciso para el cálculo de funciones matriciales que existe en la actualidad.