École Polytechnique Fédérale de Lausanne
Chair of Numerical Algorithms and High-Performance Computing

A Krylov subspace algorithm for evaluating φ-functions

Master Semester Project
March 20, 2023

Student: Sepehr Mousavi

Supervisors: Daniel Kressner
Yannis Voet

Contents

Todo list

Abstract

Write abstract.

1 Introduction

Add motivation and applications (exponential integrators, etc.)

1.1 Univariate Matrix Functions

This section reviews the basic definition and properties of matrix functions discussed in detail in [3]. Let $A \in \mathbb{C}^{n\times n}$ have s distinct eigenvalues $\{\lambda_i\}_{i=1}^{s}$. A can be expressed in the Jordan canonical form as $A = ZJZ^{-1} = Zdiag(J_1, J_2, \ldots, J_K)Z^{-1}$. Let $ind_{\lambda_i}(A)$ denote the size of the largest Jordan block associated with λ_i. The matrix function associated with a univariate function f is defined by $f(A) := g(A)$, where $g(z)$ is the unique Hermite interpolating polynomial of degree less than $\sum_{i=1}^{s} ind_{\lambda_i}(A)$. $g(z)$ which satisfies

$$\frac{\partial^j}{\partial z^j} g(\lambda_i) = \frac{\partial^j}{\partial z^j} f(\lambda_i), \quad \forall g \in \{0, 1, \ldots, ind_{\lambda_i(A)} - 1\}, \quad \forall i \in \{1, 2, \ldots, s\}, \tag{1.1}$$

assuming that all required derivatives of $f(z)$ exist.

It could be shown that if $A = diag(\lambda_i)_{i=1}^{n}$ is diagonal, we have

$$f(A) = diag(f(\lambda_i))_{i=1}^{n}. \tag{1.2}$$

This is simply because $A^k = diag(\lambda_i^k)_{i=0}^{n}$ for any k.

For any invertible matrix $P \in \mathbb{C}^{n\times n}$, we have

$$f(A) = p(A) = Pp(P^{-1}AP)P^{-1} = Pf(P^{-1}AP)P^{-1} \tag{1.3}$$

If A is diagonalizable, say $P^{-1}AP = diag(\lambda_i)_{i=1}^{n} =: \Lambda$, using Equation 1.3 and Equation 1.2 we can write

$$f(A) = Pf(\Lambda)P^{-1} = Pdiag(f(\lambda_i))_{i=1}^{n}P^{-1}. \tag{1.4}$$

Matrix Exponential

The matrix function attributed to the scalar exponential function is called matrix exponential and is defined [3] as

$$\exp(A) = \sum_{k=0}^{\infty} \frac{1}{k!} A^k. \tag{1.5}$$

Summ the prope ties.

By getting the conjugate transpose of Equation 1.5, we conclude

$$\exp(A^*) = \exp(A)^*. \tag{1.6}$$

For every $B \in \mathbb{C}^{n\times n}$ that satisfies $AB = BA$, we have

$$\exp(A+B) = \exp(A)\exp(B). \tag{1.7}$$

$\exp(A)$ is always invertible and its inverse is

$$\exp(A)^{-1} = \exp(-A). \tag{1.8}$$

The derivative of $\exp(tA)$ with respect to a scalar t is given by

$$\frac{\partial \exp(A)}{\partial t} = A\exp(tA). \tag{1.9}$$

Determinant of $\exp(A)$ is the exponential of trace of A,

$$det(\exp(A)) = \exp(trace(A)). \tag{1.10}$$

1.2 Phi-functions

Add applications and motivation.

There are a group of matrix functions called φ-functions that are essential for exponential integrators for ordinary differential equations. Accurate and efficient evaluation of these functions is necessary for implementation of exponential integrators. For a scalar arguments $z \in \mathbb{C}$, they are defined [3] as

$$\varphi_0(z) = \exp(z), \qquad \varphi_p(z) = \frac{1}{(p-1)!}\int_0^1 e^{(1-\theta)z}\theta^{p-1}d\theta, \quad p \in \mathbb{N}. \tag{1.11}$$

The φ-functions satisfy [3] the recurrence relation

$$\varphi_p(z) = z^{-1}\left[\varphi_{p-1}(z) - \frac{1}{(p-1)!}\right], \quad p \in \mathbb{N} \tag{1.12}$$

We can use Equation 1.12 recursively as

$$\begin{aligned}
\varphi_p(z) &= z^{-1}\varphi_{p-1}(z) - \frac{1}{(p-1)!}z^{-1} \\
&= z^{-1}\left[z^{-1}\varphi_{p-2}(z) - \frac{1}{(p-2)!}z^{-1}\right] - \frac{1}{(p-1)!}z^{-1} \\
&= z^{-2}\varphi_{p-2}(z) - \frac{1}{(p-2)!}z^{-2} - \frac{1}{(p-1)!}z^{-1} \\
&= \cdots \\
&= z^{-p}\varphi_0(z) - \frac{1}{0!}z^{-p} - \frac{1}{1!}z^{-(p-1)} - \frac{1}{2!}z^{-(p-2)} - \cdots - \frac{1}{(p-1)!}z^{-1} \\
&= z^{-p}\exp\{z\} - \sum_{k=0}^{p-k}\frac{1}{k!}z^{-(p-l)}
\end{aligned} \tag{1.13}$$

to derive a closed form for φ_p:

$$\varphi_p(z) = z^{-p}\left(\exp(z) - \sum_{k=0}^{p-1}\frac{1}{k!}z^k\right). \tag{1.14}$$

For an invertible matrix $A \in \mathbb{C}^{n\times n}$, we can do the same steps and use Equation 1.5 to get

$$\varphi_p(A) = A^{-p}\left(\exp(A) - \sum_{k=0}^{p-1}\frac{1}{k!}A^k\right) = \sum_{k=p}^{\infty}\frac{1}{k!}A^{k-p}. \tag{1.15}$$

1.3 Bivariate Matrix Functions

Let $A \in \mathbb{C}^{n\times n}$ have s distinct eigenvalues $\{\lambda_i\}_{i=1}^s$ and $B \in \mathbb{C}^{m\times m}$ have t distinct eigenvalues $\{\mu_i\}_{i=1}^t$. Consider a bivariate polynomial $p(x,y) = \sum_{i=1}^s\sum_{j=1}^t p_{ij}x^iy^j$ with $p_{i,j} \in \mathbb{C}$. Then $p\{A,B\} : \mathbb{C}^{m\times n} \to \mathbb{C}^{m\times n}$ is defined [4] by

$$\sum_{i=1}^{s}\sum_{j=1}^{t} p_{ij}A^iC(B^\top)^j. \tag{1.16}$$

The bivariate matrix function associated with a general bivariate scalar function $f(x,y)$ is a linear operator on $\mathbb{C}^{n\times m}$ and is defined [4] by $f\{A,B\} := p\{A,B\}$, where p is the unique Hermite interpolating polynomial which satisfies

$$\frac{\partial^{g+h}}{\partial x^g y^h}g(\lambda_i,\mu_j) = \frac{\partial^{g+h}}{\partial x^g y^h}f(\lambda_i,\mu_j), \quad \begin{matrix}\forall g \in \{0,1,\dots,ind_{\lambda_i(A)}-1\}, & \forall i \in \{1,2,\dots,s\},\\ \forall h \in \{0,1,\dots,ind_{\mu_j(B)}-1\}, & \forall j \in \{1,2,\dots,t\}.\end{matrix}$$

f only exists if the mixed partial derivatives in the definition exist and are continuous.

2 Methodology

2.1 Krylov Subspaces and the Arnoldi Algorithm

Describe the Arnoldi Iteration from [1].

2.2 Approximation of Phi-functions

2.2.1 Approximation

In order to evaluate $\varphi_p(A)v$ for a matrix $A \in \mathbb{C}^{n\times n}$ and a vector $v \in \mathbb{C}$ more efficiently, we will can run the Arnoldi algorithm to get an orthonormal basis $V_m \in \mathbb{C}^{n\times m}$ for the Krylov subspace of order m induced by the matrix A and the vector v:

$$\mathcal{K}_m = span\{v, Av, A^2v, \dots, A^{m-1}v\}, \quad m \le n, \tag{2.17}$$

and the projection of the action of A on this Krylov subspace $H_m = V_m^*AV_m$. The Arnoldi factorization reads

$$AV_m = V_mH_m + h_{m+1,m}v_{m+1}e_m^\top. \tag{2.18}$$

Lemma 2.1. *Considering the setting described above, we can write*

$$A^k v = V_m H_m^k V_m^* v, \quad \forall k in \{0, 1, \ldots, m-1\}. \tag{2.19}$$

Proof. For $k = 0$, it reads $V_m V_m^* v = v$ which is true because $v \in \mathcal{K}_m$ and $V_m V_m^*$ is the projection matrix of $\mathcal{K}_m$. For $k \geq 1$, we prove by induction. Assuming that the lemma holds true for $k-1$, we can write

$$A^k v = A A^{k-1} v = A V_m H_m^{k-1} V_m^* v.$$

Since $A^k v \in \mathcal{K}_m$, projecting it on $\mathcal{K}_m$ will result the same vector, thus

$$A^k v = V_m V_m^* A^k v = V_m \underbrace{V_m^* A V_m}_{H_m} H_m^{k-1} V_m^* v,$$

which completes the proof. □

We now approximate $\varphi_p(A)v$ by $\varphi_p(V_m H_m V_m^*)v$. Beacuse of orthogonality of V_m we can conclude that for any $k \in \mathbb{N}$,

$$(V_m H_m V_m^*)^k = V_m H_m^k V_m^*. \tag{2.20}$$

Since $V_m^* V_m = I_m$ and $V_m V_m^* v = v$, we have $\varphi_p(V_m H_m V_m^*)v = V_m \varphi_p(H_m) V_m^* v$. Because all columns of V_m except the first one are orthogonal to v, we have $V_m^* v = \|v\|_2 e_1$, where e_1 is the first vector in the standard basis of $\mathbb{R}^n$.

The approximation could be written as

$$\varphi_p(A)v \simeq \|v\|_2 V_m \varphi_p(H_m) e_1. \tag{2.21}$$

With this approximation, instead of evaluating the φ-function for a $n \times n$ matrix, we just need to evaluate it for H_m which is much smaller than A. Similar to [5], for $p > 0$, we compute $\varphi_p(H_m)e_1$ by computing the exponential of a slightly larger matrix $\hat{H}_m \in \mathbb{C}^{(m+p)\times(m+p)}$. If we define $\hat{H}_m$ by

$$\hat{H}_m = \begin{bmatrix} H_m & e_1 & 0 \\ 0 & 0 & I_{p-1} \\ 0 & 0 & 0 \end{bmatrix} \begin{matrix} m \\ p-1 \\ 1 \end{matrix} \quad \begin{matrix} \text{rows} \\ \text{row(s)} \\ \text{row} \end{matrix}, \tag{2.22}$$

we can read $\varphi_p(H_m)e_1$ from the first m rows of the $m + p^{\text{th}}$ column of $\exp\left(\hat{H}_m\right)$.

Proof. We calculate the exponential of this matrix using Equation 1.5 and we will only focus on the part that interests us, namely its last column $\exp\left(\hat{H}_m\right)_{m+p} \in \mathbb{C}^{m+p}$. By induction, we can show that

$$(\hat{H}_m^k)_{m+p} = \begin{cases} \begin{bmatrix} \overbrace{0}^{m} & \overbrace{e_{p-k}}^{p-1} & \overbrace{0}^{1} \end{bmatrix}^\top & k \in \{1, 2, \ldots, p-1\} \\ \begin{bmatrix} \underbrace{H_m^{k-p} e_1}_{m} & \underbrace{0}_{p-1} & \underbrace{0}_{1} \end{bmatrix}^\top & k \geq p \end{cases}. \tag{2.23}$$

For $k = 1$, it holds by definition. For $k \geq 2$, we now show that assuming that it holds for $k-1$, it is also true for k. Since $\hat{H}_m^k = \hat{H}_m \hat{H}_m^{k-1}$, we can get the $m+p^{\text{th}}$ column of $\hat{H}_m^k$, represented by $(\hat{H}_m^k)_{m+p}$, by multiplying only the $m+p^{\text{th}}$ column of $\hat{H}_m^{k-1}$. For $p-1 \geq k \geq 2$ we have

$$\begin{aligned}(\hat{H}_m^k)_{m+p} &= \hat{H}_m (\hat{H}_m^{k-1})_{m+p} \\ &= \begin{bmatrix} H_m & e_1 & 0 \\ 0 & 0 & I_{p-1} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ e_{p-k+1} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ e_{p-k} \\ 0 \end{bmatrix}.\end{aligned} \tag{2.24}$$

After each multiplication, we get the previous column of $\hat{H}_m$ until we reach the $m+1^{\text{th}}$ column for $k = p$: $(\hat{H}_m^p)_{m+p} = \begin{bmatrix} e_1 & 0 & 0 \end{bmatrix}^\top$, which is consistent with Equation 2.23. With this, for $k \geq p+1$ we have

$$\begin{aligned}(\hat{H}_m^k)_{m+p} &= \hat{H}_m (\hat{H}_m^{k-1})_{m+p} \\ &= \begin{bmatrix} H_m & e_1 & 0 \\ 0 & 0 & I_{p-1} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} H_m^{k-1-p} e_1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} H_m^{k-p} e_1 \\ 0 \\ 0 \end{bmatrix}.\end{aligned} \tag{2.25}$$

Using Equation 2.23 and Equation 1.15, we conclude

$$\begin{aligned}\exp\left(\hat{H}_m\right)_{\{1,\dots,m\},m+p} &= \sum_{k=0}^{\infty} \frac{1}{k!} (\hat{H}_m^k)_{\{1,\dots,m\},m+p} \\ &= \sum_{k=p}^{\infty} \frac{1}{k!} H_m^{k-p} e_1 = \varphi_p(H_m) e_1\end{aligned}. \tag{2.26}$$

□

2.2.2 Error Estimation

The method described in this section could be generalized to any function f that is analytic on a domain including the eigenvalues of A.

Lemma 2.2. *For all $\Pi_{m-1} \ni g(z) = \sum_{k=0}^{m-1} \alpha_k z^k$, the approximation is exact; where Π_{m-1} is the set of all polynomials of degree up to $m-1$.*

Proof. Using the definition of g and 2.1, we have

$$\begin{aligned}g(A)v = \sum_{k=0}^{m-1} \alpha_k A^k v = \sum_{k=0}^{m-1} \alpha_k V_m H_m^k V_m^* v = V_m \underbrace{\left(\sum_{k=0}^{m-1} \alpha_k H_m^k \right)}_{g(H_m)} \underbrace{V_m^* v}_{\|v\|_2 e_1} \\ = \|v\|_2 V_m g(H_m) e_1,\end{aligned}$$

which show that the approximation is exact. □

Theorem 2.3. *Let $A \in \mathbb{C}$ be a Hermitian matrix. For a general scalar function $f : \mathbb{C} \to \mathbb{C}$ that is analytic on a domain $\Omega \subset \mathbb{C}$ containing the spectrum of A, $\Lambda(A) \in \mathbb{R}$, the error of the approximation $f_m := \|v\|_2 V_m f(H_m) e_1$ is bounded by the maximum error of the polynomial that approximates f the best in Ω:*

$$\|f(A)v - f_m\|_2 \leq 2 \|v\|_2 \min_{g \in \Pi_{m-1}} \max_{z \in \Lambda(A)} |f(z) - g(z)| \tag{2.27}$$

Proof. Considering an arbitrary polynomial $g \in \Pi_{m-1}$, we define the error $e = f - g$, we use Theorem 2.2 and the Cauchy-Schwarz inequality to get

$$\begin{aligned}
\|f(A)v - f_m\|_2 &= \left\| f(A)v \overbrace{-g(A)v + \|v\|_2 V_m g(H_m)e_1}^{zero} - \|v\|_2 V_m f(H_m)e_1 \right\|_2 \\
&\leq \|f(A)v - g(A)v\|_2 + \|v\|_2 \overbrace{\|V_m\|_2}^{=1} \|g(H_m)e_1 - f(H_m)e_1\|_2 \\
&\leq \|v\|_2 \left(\|e(A)\|_2 + \|e(H_m)\|_2\right) \\
&\leq 2\|v\|_2 \max_{z\in\Lambda(A)} |e(z)| .
\end{aligned}$$

In the last step, we used that $\Lambda(H_m) \subset \Lambda(A) \subset \Omega$ holds due to eigenvalue interlacing. □

To specialize Theorem 2.3 to phi-functions, we follow two approaches. The first one is using truncated Taylor series to get an approximation for the left-hand-side of Equation 2.27, which results in the following theorem.

Lemma 2.4. *For a phi-function $\varphi_p(z)$ defined in Equation 1.14 and a Hermitian matrix $A \in \mathbb{C}^{n\times n}$ with the spectrum $\Lambda(A) \subset [-\alpha, 0] \subset \mathbb{R}$ for $\alpha > 0$, the error of the approximation $\varphi_{p,m} := \|v\|_2 V_m \varphi_p(H_m)e_1$ scales with α^m:*

$$\|\varphi_p(A)v - \varphi_{p,m}\|_2 \leq 2\|v\|_2 \frac{\alpha^m}{(m+p)!} \tag{2.28}$$

Proof. We replace the exponential in Equation 1.14 by its truncated Taylor expansion around zero and we keep the first $m + p - 1$ to get

$$\begin{aligned}
\varphi_p(z) &= z^{-p}\left(\sum_{k=0}^{m+p-1} \frac{1}{k!}z^k + \frac{1}{(m+p)!}|z|^{m+p}\exp(z') - \sum_{k=0}^{p-1}\frac{1}{k!}z^k\right) \\
&= z^{-p}\left(\sum_{k=p}^{m+p-1} \frac{1}{k!}z^k + \frac{1}{(m+p)!}|z|^{m+p}\exp(z')\right) \\
&\leq \sum_{k=0}^{m-1}\frac{1}{k!}z^k + \frac{1}{(m+p)!}|z|^m, \quad \forall z \in \Lambda(A)
\end{aligned}$$

where $-\alpha \leq z' \leq 0$. By choosing $g(z) = \sum_{k=0}^{m-1}\frac{1}{k!}z^k$ and taking the maximum of the absolute value of both sides for $z \in \Lambda(A)$, we can write

$$\max_{z\in\Lambda(A)} |\varphi_p(z) - g(z)| \leq \frac{1}{(m+p)!}\max_{z\in\Lambda(A)}|z|^m = \frac{1}{(m+p)!}\alpha^m.$$

With this, we have found a polynomial $g \in \Pi_{m-1}$ that has this error bound, so we can be sure that the minimum of the error bound over all polynomials in Π_{m-1} is less than this bound. Combining this result with Theorem 2.3 completes the proof. □

Using the polynomial approximation given in [2, Lemma A.1], we can get a better error bound for approximating $\varphi_p(A)v$. This approximation is only given for $p = 1$, but with a minor modification

in the proof, it can be shown that the same bound holds true for $p > 1$ as well. The polynomial approximation states that

$$\min_{g\in\Pi_{m-1}} \max_{z\in[-\alpha,0]} |\varphi_p(z) - g(z)| \leq \begin{cases} \frac{5\alpha^2}{2m^3} \exp\left(-\frac{4m^2}{5\alpha}\right)^m & \sqrt{\alpha} \leq m \leq \frac{\alpha}{2} \\ \frac{32}{12m-5\alpha} \left(\frac{e\alpha}{4m+2\alpha}\right)^m & m \geq \frac{\alpha}{2} \end{cases}. \tag{2.29}$$

Using this polynomial approximation error bound and combining it with Equation 2.27 gives the following theorem for the Arnoldi method approximation of $\varphi_1(A)v$ for Hermitian matrices.

Theorem 2.5. *For a phi-function $\varphi_p(z)$ defined in Equation 1.14 and a Hermitian matrix $A \in \mathbb{C}^{n\times n}$ with the spectrum $\Lambda(A) \subset [-\alpha, 0] \subset \mathbb{R}$ for $\alpha > 0$, the error of the approximation $\varphi_{p,m} := \|v\|_2 V_m \varphi_p(H_m) e_1$ scales with α as*

$$\|\varphi_p(A)v - \varphi_{p,m}\|_2 \leq \begin{cases} \|v\|_2 \frac{\alpha^m}{(m+p)!} \frac{5\alpha^2}{m^3} \exp\left(-\frac{4m^2}{5\alpha}\right)^m & \sqrt{\alpha} \leq m \leq \frac{\alpha}{2} \\ \|v\|_2 \frac{\alpha^m}{(m+p)!} \frac{64}{12m-5\alpha} \left(\frac{e\alpha}{4m+2\alpha}\right)^m & m \geq \frac{\alpha}{2} \end{cases} \tag{2.30}$$

2.3 Approximation of Bivariate Matrix Functions

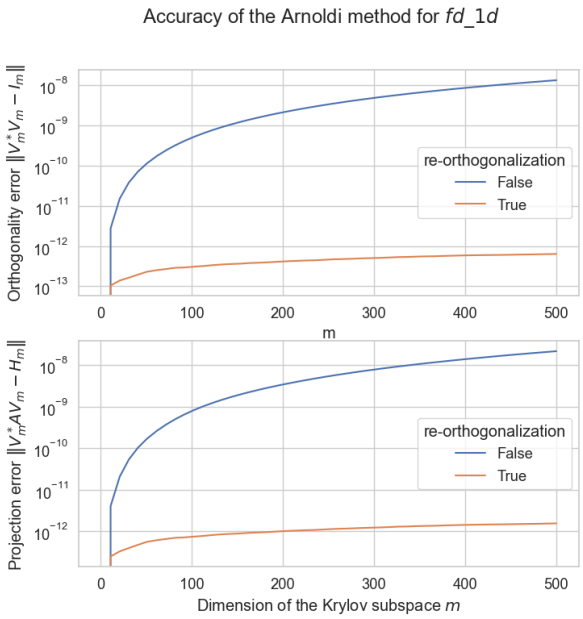
Summerize the algorithm in [2].

3 Implementations and Experiments

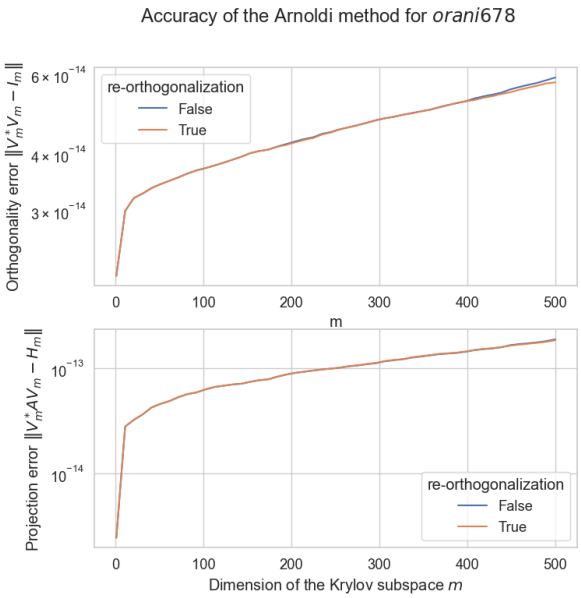
In this section, the implementations of the methods described in section 2 are presented and evaluated. The matrices along with their properties are listed in Table 3.1. The first three matrices are stiffness matrices that appear for solving the Poisson's equation with finite difference method over an uniform grid. The rest are collected from [6].

Table 3.1: Properties of the test matrices. The last column is the subset that includes the eigennvalues of the matrix, reported only if the matrix is Hermitian.

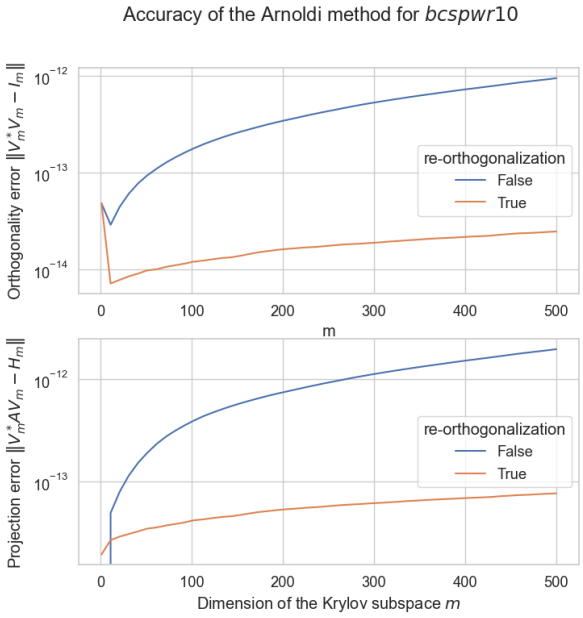
Name	Size	Density	Hermitian	Full numerical rank	Condition number	$\Omega \subset \mathbb{R}$
`fd_1d`	4096	0.0007	✓	✓	6.80e+06	$[-4, 0]$
`fd_2d`	4096	0.0012	✓	✓	1.71e+03	$[-8, 0]$
`fd_3d`	4096	0.0012	✓	✓	1.16e+02	$[-8, 0]$
`orani678`	2529	0.0141	×	✓	9.58e+03	-
`bcspwr10`	5300	0.0008	✓	×	1.58e+17	$[-3, 7]$
`gr_30_30`	900	0.0096	✓	✓	1.95e+02	$[0, 12]$
`helm2d03`	392257	< 0.0001	✓	✓	?	$[0, 11]$



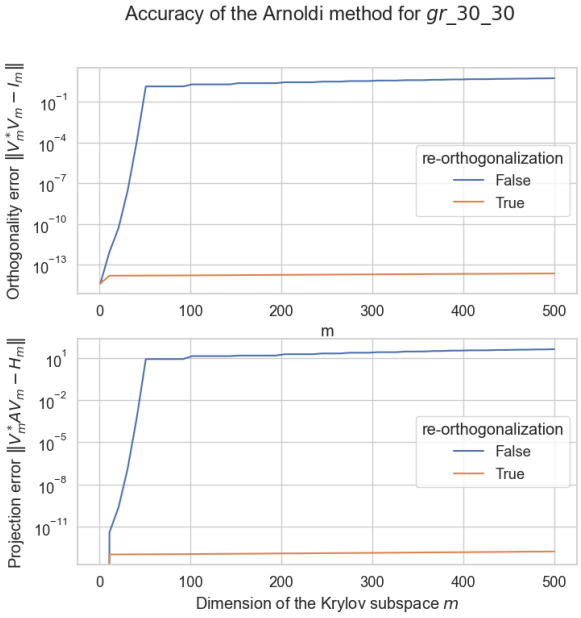
(a) `fd_1d`.



(b) `orani678`.



(c) `bcspwr10`.



(d) `gr_30_30`.

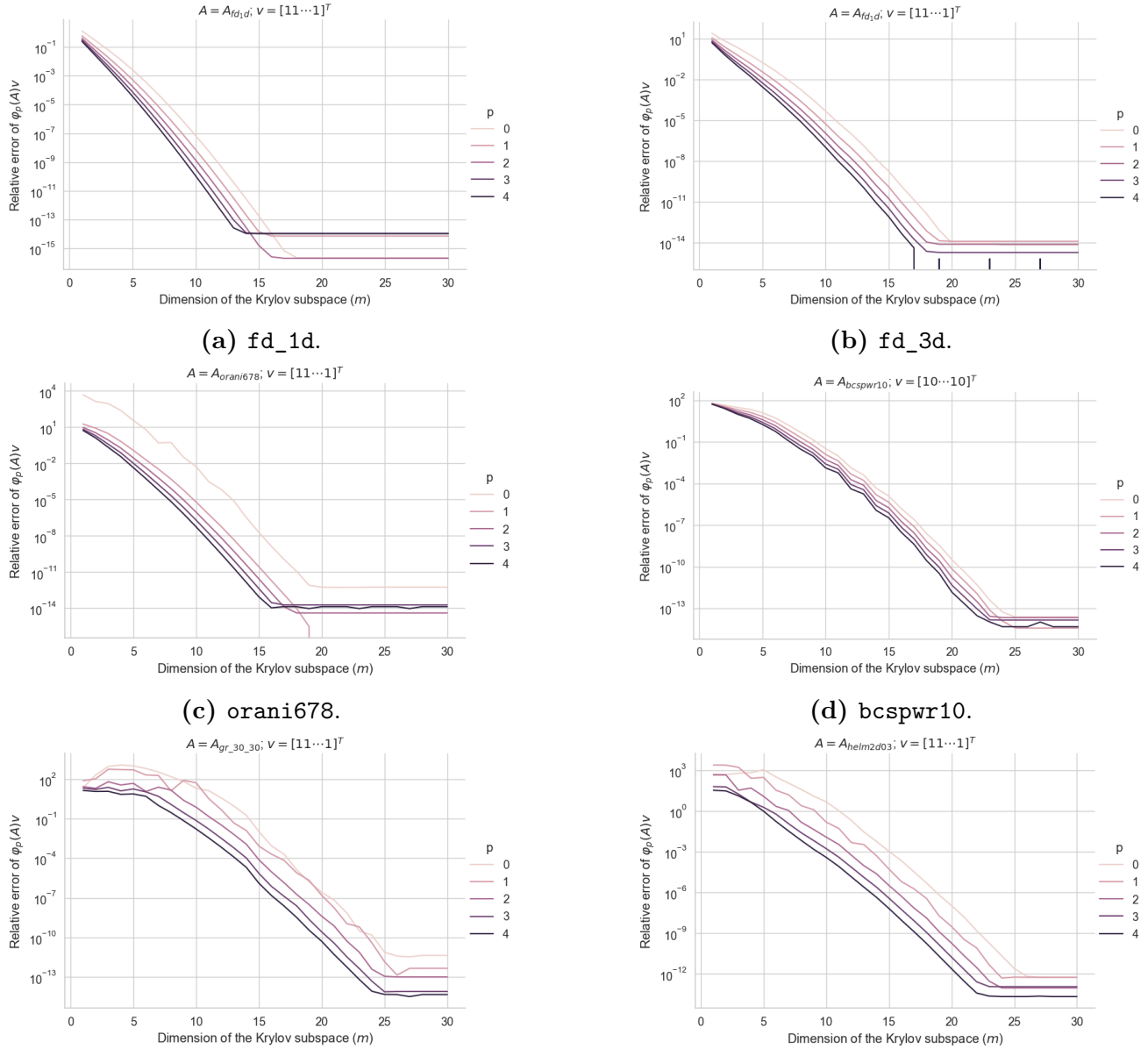
Figure 3.1: Evaluation of the implemented Arnoldi algorithm with different matrices.

3.1 Arnoldi Algorithm

The Arnoldi Algorithm described in subsection 2.1 is implemented as in (Listing 1). In order to make sure that the implementation is working well, it is applied on the test matrices with different dimensions of the Krylov subspace m. Each experiment is done once with the re-orthogonalization steps and once without it. To evaluate the quality of the outputs, we measure two things: 1. the deviation of $V_m^* V_m$ from the identitiy matrix I_m to check the orthogonality of the basis vectors; 2. the deviation of the projection of the action of A on the Krylov subspace given by $V_m^* A V_m$

from the output H_m. For both measures, we report the L_2-norm of the matrices.

Figure 3.1 depicts the results for four matrices. First of all, we can see that the accuracy of the method always deteriorates for higher dimensions of Krylov subspace. However, for almost all the cases, the errors are in an acceptable range for up to $m = 500$, if re-orthogonalization is applied. This is not the case for the matrix `fd_3d`, where the errors explode after around $m = 200$. Furthermore, we can see that re-orthogonalization always imporves the accuracy of the method up to several orders of magnitude, except for `orani678` where it does not have any effect. By looking at the plots corresponding to `gr_30_30`, we can see that without re-orthogonalization, the performance of the method is not acceptable even for small m's.



(a) `fd_1d`.

(b) `fd_3d`.

(c) `orani678`.

(d) `bcspwr10`.

(e) `gr_30_30`.

(f) `helm2d03`.

Figure 3.2: Evaluation of the implemented Arnoldi algorithm with different matrices.

3.2 Approximation of Phi-functions

The method described in subsection 2.2 is implemented and its approximations has been evaluated for different matrices. The implementation is presented in Listing 2. For computing $\exp\{\hat{H}_m\}$, the `scipy.linalg.expm` function from the the SciPy library [7] is used which implements a Pade approximation with a variable order that is decided based on the array data. In order to validate the implementation, we use an implementation of Equation 1.15, presented in Listing 3, and we look at the relative error of the results from the Krylov subspace method of dimension m, denoted by $\varphi_p^{(m)}(A)v$, against the vectors computed by Equation 1.15, denoted by $\varphi_p(A)v$, for different matrices and for small p's. The relative error is computed as:

$$\left\| \frac{[\varphi_p^{(m)}(A)v]_{nz} - [\varphi_p(A)v]_{nz}}{[\varphi_p(A)v]_{nz}} \right\|_2 .$$

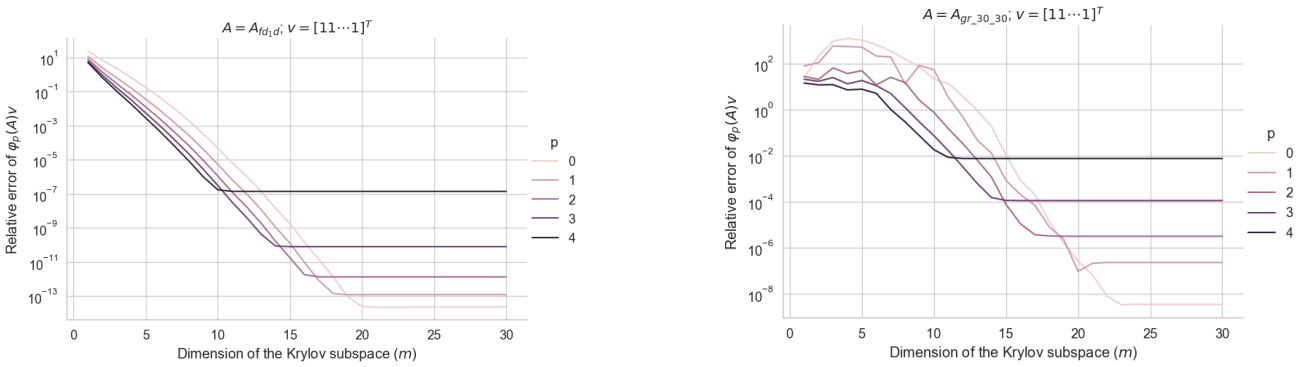
Following [5], the elements that are zero in $\varphi_p^{(m)}(A)v$ are excluded from both vectors. After the implementation is validated, we take the approximations with $m = 256$ as reference and we compute

$$\left\| \frac{[\varphi_p^{(m)}(A)v]_{nz} - [\varphi_p^{(256)}(A)v]_{nz}}{[\varphi_p^{(256)}(A)v]_{nz}} \right\|_2$$

for $1 \leq m \leq 30$ and $0 \leq p \leq 4$. The convergence plots are illustrated in Figure 3.2. For all the test matrices in Table 3.1, the approximation are carried out for a vector of ones of a suitable size, except for `bcspwr10` where the vector is $v = [1\,0\,1\,0\,\ldots\,1\,0]$.

The error estimates given in Theorem 2.4 and Theorem 2.5 show that the eigenvalues of the matrix change the rate of convergence of the method. They also show that the starting error decreases for larger p's. Although not all the test matrices satisfy the assumptions of these error estimates, we can see in Figure 3.2 that both theoretical interpretations are consistent with the numerical results. For the matrix `gr_30_30`, the error increases for small m's, which also could be justified with the theoretical estimations because the largest eigenvalue of this matrix is larger than the others.

Remark. *The reason that the relative errors are not reported against the the results of the implementation of Equation 1.15 is that this implementation seems to work poorly as p increases. One reason could be that we need to compute the inverse of A^p for this method. Although computation of the inverse is avoided by solving the corresponding system of equations instead, it can still perform poorly because the condition number of A^p grows as p is increased. This hypothesis is supported by Figure 3.3 where the convergence plots are computed against this implementation for two matrices with relatively mild condition numbers. We can see that as p is increased, the minimum error that could be reached gets worse. This shows that the approximation is performing better than the implementation of Equation 1.15. Furthermore, this pattern is less significant for the matrix `fd_3d` which has a smaller condition number than `gr_30_30`. This is also the case for other test matrices.*



(a) `fd_3d`. **(b)** `gr_30_30`.

Figure 3.3: Evaluation of the implemented Arnoldi algorithm agains the implementation of Equation 1.15 for matrices with mild condition numbers.

References

[1] Lloyd N Trefethen and David Bau III. Numerical linear algebra, vol. 50, 1997.

[2] Daniel Kressner. A krylov subspace method for the approximation of bivariate matrix functions. *Structured Matrices in Numerical Linear Algebra: Analysis, Algorithms and Applications*, pages 197–214, 2019.

[3] Nicholas J Higham. *Functions of matrices: theory and computation*. SIAM, 2008.

[4] Daniel Kressner. Bivariate matrix functions. *Operators And Matrices*, 8(2):18. 449–466, 2014.

[5] Jitse Niesen and Will M. Wright. Algorithm 919. *ACM Transactions on Mathematical Software*, 38(3):1–19, apr 2012.

[6] Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1–25, 2011.

[7] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

A Code snippets

```python
    def arnoldi(A: Matrix, v: np.ndarray, m: int, ro: bool = True) -> tuple[np.ndarray, np.ndarray]:
        """
        Arnoldi algorithm (Niesen).
        """

        # Check dimensions
        n = len(v)
        dtype = A.dtype
        assert A.shape == (n, n)
        assert m <= n

        # Initialize
        H_m = np.zeros(shape=(m, m), dtype=dtype)
        V_m = np.zeros(shape=(n, m), dtype=dtype)

        V_m[:, 0] = v / np.linalg.norm(v)
        for j in range(0, m):
            w = A @ V_m[:, j]
            w_norm = np.linalg.norm(w)
            for i in range(0, j + 1):
                H_m[i, j] = V_m[:, i].dot(w)
                w = w - (H_m[i, j] * V_m[:, i])

            # Reorthogonalization
            if ro:
                if np.linalg.norm(w) < .7 * w_norm:
                    h_hat = V_m[:, :(j+1)].conjugate().T @ w
                    H_m[:(j+1), j] += h_hat
                    w -= V_m[:, :(j+1)] @ h_hat

            if j + 1 >= m:
                break
            H_m[j + 1, j] = np.linalg.norm(w)
            V_m[:, j + 1] = w / H_m[j + 1, j]

        return V_m, H_m
```

Listing 1: Code snippet of the implemented Arnoldi algorithm.

```python
    def krylovsubspace(self, A: SparseMatrix, v: np.ndarray, m: int) -> np.ndarray:
        """Computes the phi-function evaluation using the Arnoldi iteration and
        the method described int the Niesen's paper.
        """

        # Check types
        assert isinstance(m, int)

        # Check dimensions
        n = len(v)
        dtype = A.dtype
        assert A.shape == (n, n)
```

```
        assert m <= n
        assert m > 0

        # Arnoldi method
        V_m, H_m = self.arnoldi(A, v, m)

        # Fetch p
        p = self.p

        if p == 0:
            # Define e1
            e1 = np.zeros(shape=(m,), dtype=dtype)
            e1[0] = 1
            # Calculate {\phi}_0(H_m)
            phi_H = la.expm(H_m)
            # Calculate {\phi}_0(H_m) e1
            phi_H_e1 = phi_H @ e1

        else:
            # Construct the H_{hat} matrix
            H_h = np.zeros(shape=(m+p, m+p), dtype=dtype)
            H_h[:m, :m] = H_m
            H_h[0, m] = 1
            H_h[m:m+p-1, m+1:m+p] = np.eye(p-1, dtype=dtype)
            # Calculate {\phi}_p(H_m) e_1
            phi_H_e1 = la.expm(H_h)[:m, -1]

        # Calculate the approximation of {\phi}_p(A) v
        phi_A_v = la.norm(v) * V_m @ phi_H_e1

        return phi_A_v
```

Listing 2: Code snippet of the implemented krylov subspace method for the approximation of univariate phi-functions.

```
    def recursive(self, A: SparseMatrix, v: np.ndarray) -> np.ndarray:
        """Computes the action of the matrix function on a vector using the
    recurrence relation."""

        # Read info of A
        n = A.shape[0]
        dtype = A.dtype
        if not sps.issparse(A):
            raise Exception
        format = A.format

        # Convert v to sparse matrix
        assert len(v.shape) == 1
        v = v.reshape((-1, 1))
        if format == 'csc':
            v = sps.csc_matrix(v)
        elif format == 'csr':
            v = sps.csr_matrix(v)
        else:
            raise Exception
```

```
20
21          # Compute $\exp(A) v$ and return if $p=0$
22          eAv = spla.expm_multiply(A, v)
23          if not self.p:
24              return eAv.toarray().reshape(-1)
25
26          # Compute $\varphi(A) v$ if $p>0$
27          if format == 'csc':
28              res = sps.csc_matrix((n, n), dtype=dtype)
29          elif format == 'csr':
30              res = sps.csr_matrix((n, n), dtype=dtype)
31          A_powered = sps.eye(n, dtype=dtype, format=format)
32          for k in range(self.p):
33              res = res + A_powered / np.math.factorial(k)
34              A_powered = A @ A_powered
35
36          phiAv = multiply_by_inverse(
37              A=(A ** self.p),
38              B=(eAv - res @ v),
39              mode='left',
40          )
41
42          return phiAv.reshape(-1)
```

Listing 3: Code snippet of the implemented phi-function evaluation in Equation 1.15.