Acta Cybernetica **26** (2023) 129–147.

Affine Iterations and Wrapping Effect: Various Approaches

Nathalie Revol[a]

Abstract

Affine iterations of the form $x_{n+1} = Ax_n + b$ converge, using real arithmetic, if the spectral radius of the matrix A is less than 1. However, substituting interval arithmetic to real arithmetic may lead to divergence of these iterations, in particular if the spectral radius of the absolute value of A is greater than 1. We will review different approaches to limit the overestimation of the iterates, when the components of the initial vector x_0 and b are intervals. We will compare, both theoretically and experimentally, the widths of the iterates computed by these different methods: the naive iteration, methods based on the QR- and SVD-factorization of A, and Lohner's QR-factorization method. The method based on the SVD-factorization is computationally less demanding and gives good results when the matrix is poorly scaled, it is superseded either by the naive iteration or by Lohner's method otherwise.

Keywords: interval analysis, affine iterations, matrix powers, Lohner's QR algorithm, QR factorization, SVD factorization

1 Introduction

The problem we consider is the evaluation of the successive iterates of

$$\begin{cases} x_{n+1} & = & Ax_n + b, \\ x_0 \text{ given,} & & \end{cases}$$

where $A \in \mathbb{R}^{d\times d}$, $x_n \in \mathbb{R}^d$ for every $n \in \mathbb{N}$ and $b \in \mathbb{R}^d$. More specifically, the focus is on the use of interval arithmetic to evaluate these iterates.

In what follows, interval quantities will be denoted in boldface.

[a] INRIA – LIP laboratory (Unité Mixte de Recherche 5668 Centre National de la Recherche Scientifique – École Normale Supérieure de Lyon – INRIA – Université Claude Bernard de Lyon), École Normale Supérieure de Lyon, France, E-mail: Nathalie.Revol@inria.fr, ORCID: 0000-0002-2503-2274

DOI: 10.14232/actacyb.295251

1.1 A Toy Example

This problem was brought to us through this example of an IIR (Infinite Impulse Response) linear filter in a state-space form:

$$x_n = 1.8 * x_{n-1} - 0.9 * x_{n-2} + 4.7.10^{-2} * (u_{n-2} + u_{n-1} + u_n)$$

for $x_0 = 0$ and $x_1 \in [1, 1.1]$. We assume that $u_n \in \mathbf{u} = [9.95\,,\, 10.05]$ for every n.

This iteration can also be written as a linear recurrence in $\mathbb{R}^2$:

$$\begin{pmatrix} x_{n-1} \\ x_n \end{pmatrix} = A. \begin{pmatrix} x_{n-2} \\ x_{n-1} \end{pmatrix} + b_n,$$

$$\text{where} \quad A = \begin{pmatrix} 0 & 1 \\ -0.9 & 1.8 \end{pmatrix} \text{ and } b_n = \begin{pmatrix} 0 \\ 4.7.10^{-2} * (u_{n-2} + u_{n-1} + u_n) \end{pmatrix}.$$

This toy example will be used to illustrate the various approaches mentioned in this paper. The first iterates, obtained using floating-point arithmetic, with random values for $x_1 \in [1, 1.1]$ and each $u_n \in [9.95\,,\, 10.05]$, are given on the left two columns of Table 1.

The system stabilizes around 14, with variations due to the random values taken by the u_n. However, the following snippet of Octave code computes the successive iterates using interval arithmetic, using the interval $[1\,,\, 1.1]$ for $\mathbf{x}_1$ and $\mathbf{u} = [9.95\,,\, 10.05]$ for the u_n, that is, we replace $u_{n-2} + u_{n-1} + u_n$ by $3 * \mathbf{u}$.

```
A=[[0 1];[-0.9 1.8]];
xn=[infsup(0,0);infsup(1,1.1)];
b=4.7e-2 * 3.0*[infsup(0,0);infsup(9.95,10.05)];
n=500; for i=1:n, i , xn=A*xn+b, wid(xn(1)), end;
```

Table 1: Comparison of the behavior of the iterates: point values on the left, interval values on the right

n	x_n	n	x_n	n	wid($\mathbf{x}_n$)	n	wid($\mathbf{x}_n$)
0	0	20	9.1518	0	0	20	$3.2293.10^{5}$
1	1.0617	30	17.0186	1	0.1000	30	$8.8808.10^{8}$
2	3.3183	40	12.4414	2	0.1941	40	$2.4423.10^{12}$
3	6.4234	50	15.0305	3	0.4535	50	$6.7164.10^{15}$
4	9.9851	60	13.6130	4	1.0051	60	$1.8470.10^{19}$
5	13.6031	70	14.3858	5	2.2313	70	$5.0794.10^{22}$
6	16.9117	80	13.9680	6	4.9350	80	$1.3969.10^{26}$
7	19.6103	90	14.1510	7	10.905	90	$3.8415.10^{29}$
8	21.4884	100	14.0949	8	24.085	100	$1.0564.10^{33}$
9	22.4394	200	14.0870	9	53.182	200	$2.6137.10^{67}$
10	22.4595	300	14.1443	10	117.42	300	$6.4663.10^{101}$
12	20.1508	400	14.1282	12	572.31	400	$1.5998.10^{136}$
15	13.8931	500	14.0828	15	6158.0	500	$3.9580.10^{170}$

On the right two columns of Table 1, the widths of the successive iterates $\mathbf{x}_n$ are given: the widths of the iterates diverge rapidly to infinity.

The explanation of this phenomenon is the following: the spectral radius of A is strictly less than 1: $\rho(A) \simeq 0.9487 < 1$, and thus the exact (and, for that matter, floating-point) iterations converge. However, the recurrence satisfied by the widths of the iterates is $\mathrm{wid}(\mathbf{x}_n) = 1.8 * \mathrm{wid}(\mathbf{x}_{n-1}) + 0.9 * \mathrm{wid}(\mathbf{x}_{n-2}) + 4.7.10^{-2} * 3 * \mathrm{wid}(\mathbf{u})$, which corresponds to the 2-dimensional iteration $w_n = |A|.w_{n-1} + w_b$, with $w_n = \mathrm{wid}(\mathbf{x}_n)$, $|A|$ the matrix whose coefficients are the absolute values of the coefficients of A and $w_b = 4.7.10^{-2} * 3 * \mathrm{wid}(\mathbf{u})$. As the spectral radius of $|A|$ is larger than 1, indeed $\rho(|A|) \simeq 2.208 > 1$, the iterations diverge.

This phenomenon is a special case of the so-called *wrapping effect*. Its ubiquity in interval computations has been put in evidence by Lohner in [8].

1.2 The Wrapping Effect

The wrapping effect is ubiquitous, as defined and developed in [8]. It can be described as the overestimation due to the enclosure of the sought set in a set of a given simple structure. In our case, this simple structure corresponds to multidimensional intervals or boxes, that is, parallelepipeds with sides parallel to the axes of the coordinate system. When the computation is iterative, and when each iteration produces such an overestimating set that is used as the starting point of the next iteration, the size of the computed set may grow exponentially in the number of iterations, even when the exact solution set remains bounded and small.

Lohner also put in evidence that the affine iteration we study in this paper, namely $x_{n+1} = Ax_n + b$, or more generally $x_{n+1} = A_n x_n + b_n$ with x_{n+1}, x_n and b_n vectors in $\mathbb{R}^d$ and $A_n \in \mathbb{R}^{d \times d}$ for every $n \in \mathbb{N}$, is archetypal. It occurs in many algorithms, and the examples cited in [8] include

- matrix-vector iterations as the ones studied in this paper;
- discrete dynamical systems: $\mathbf{x}_{n+1} = f(\mathbf{x}_n)$, $\mathbf{x}_0$ given and f sufficiently smooth;
- continuous dynamical systems (ODEs): $x'(t) = g(t, x(t))$, $x(0) = x_0$, which is studied through a numerical one step method (or more) of the kind $\mathbf{x}_{n+1} = \mathbf{x}_n + h\Phi(\mathbf{x}_n, t_n) + \mathbf{z}_{n+1}$;
- difference equations: $a_0\mathbf{z}_n + a_1\mathbf{z}_{n+1} + \ldots + a_m\mathbf{z}_{n+m} = b_n$ with $\mathbf{z}_1, \ldots \mathbf{z}_m$ given;
- linear systems with (banded) triangular matrix;
- automatic differentiation.

In this paper, we concentrate on examples similar to the toy example presented above: for every initial value $x_0 \in \mathbb{R}^d$, the sequence of iterates $(x_n)_{n\in\mathbb{N}}$ converges to a finite value $x^* \in \mathbb{R}^d$, since $\rho(A) < 1$; however, the computations performed using interval arithmetic diverge because their behaviour is dictated by $\rho(|A|)$ which is larger than 1. We are interested in the iterates computed using interval arithmetic:

it is established that these iterates increase in width, however different approaches can be applied to counteract the exponential growth of the width of the iterates. Several of them, some new as in Sections 2.3.2 and 2.3.3, and some already well established as in Section 2.4, will be tried and compared, in terms of the widths of the results and the computational time.

2 Theoretical Results

2.1 Problem and Notations

Let A be a $d \times d$ matrix in $\mathbb{R}^{d \times d}$, $\mathbf{x}_0 \in \mathbb{IR}^d$ be an interval vector (boldface font is used for interval quantities and $\mathbb{IR}$ stands for the set of real intervals), x_0 a vector in $\mathbb{R}^d$ with $x_0 \in \mathbf{x}_0$, $\mathbf{b} \in \mathbb{IR}^d$ an interval vector, and b a vector in $\mathbb{R}^d$ and $b \in \mathbf{b}$. In what follows, n denotes the number of iterations.

It is assumed that $\rho(A) < 1$ and $\rho(|A|) > 1$. The fact that $\rho(A) < 1$ implies that $A^n \to 0$ when $n \to +\infty$.

A first goal is to determine the set of all fixed-points of the iteration

$$\begin{cases} x_0 \in \mathbf{x}_0, \ \ b \in \mathbf{b}, \\ x_{n+1} = A x_n + b \end{cases}$$

for every $x_0 \in \mathbf{x}_0$ and every $b \in \mathbf{b}$.

It is known that x_n can be written as

$$x_n = A^n x_0 + \sum_{i=1}^{n-1} A^i b,$$

thus

$$\{x_n : x_0 \in \mathbf{x}_0\} \subset A^n \mathbf{x}_0 + \left(\sum_{i=1}^{n-1} A^i \right) \mathbf{b}.$$

However, when the vectors x_0 and b are replaced in the iterative formula by their interval enclosures $\mathbf{x}_0$ and $\mathbf{b}$, one obtains the new interval vector $\mathbf{x}_{n+1}$, which is computed as:

$$\begin{cases} \mathbf{x}_0 \text{ and } \mathbf{b} \text{ given,} \\ \mathbf{x}_{n+1} = A\mathbf{x}_n + \mathbf{b}. \end{cases}$$

Another goal is to determine a tight enclosure for each iterate of this diverging set of intervals.

As mentioned above, the increase in widths of the iterates can be attributed to the use of parallelepipeds with sides parallel to the axes of the coordinate system, and not to the geometry of the transformation. To cure this problem, changes of coordinates will be applied, using an invertible matrix B, with $x = By \Leftrightarrow y = B^{-1}x$ and its interval counterpart $\mathbf{x} = B\mathbf{y}$. This yields the iteration

$$\begin{cases} x_{n+1} &= & B y_{n+1} \\ y_{n+1} &= & B^{-1} A B y_n + B^{-1} b \end{cases}$$

and its interval counterpart

$$\begin{cases} \mathbf{x}_{n+1} & = & B\mathbf{y}_{n+1} \\ \mathbf{y}_{n+1} & = & B^{-1}AB\mathbf{y}_n + B^{-1}\mathbf{b}. \end{cases}$$

In what follows, analogously to the approach in [10], to establish bounds and to simplify the derivation of their proofs, we assume A diagonalizable (this will not necessarily be the case for the experiments) and A can be diagonalized as $A = P^{-1}\Lambda P$ where Λ is a diagonal matrix with the eigenvalues $\lambda_1, \dots \lambda_d$ of A on the diagonal and the columns of P^{-1} are the corresponding eigenvectors.

The iteration considered in this paper corresponds to $x_n = A^n x_0 + \sum_{i=0}^{n-1} A^i b$. The numerical unstability of computing the matrix power A^n and applying it to a vector, is well known: $A^n x_0$ tends to be aligned with the eigenvector of A associated with the largest (in module) eigenvalue, and the information corresponding to the contribution of the other eigenvectors is lost. To avoid this well-known problem of the power method, we will consider orthogonal changes of coordinates. The choice of the orthogonal matrices is related to A, the matrix of the iteration.

We will first consider the QR-factorization of A: $A = QR$ with $Q \in \mathbb{R}^{d\times d}$ orthogonal, that is, $QQ' = Q'Q = I$ is the identity matrix and $R \in \mathbb{R}^{d\times d}$ is upper triangular.

The other factorization used in this paper is the SVD-factorization of A: $A = U\Sigma V'$ with U, V and $\Sigma \in \mathbb{R}^{d\times d}$, where U and V are orthogonal and Σ is a diagonal matrix with the singular values $\sigma_1, \dots \sigma_d$ of A on the diagonal. We also assume that $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n \geq 0$. This idea has been sketched but not completely developed by Beaumont in [2].

2.2 Known results

Mayer and his co-authors have extensively studied the existence of a fixed-point for the iteration studied in this paper. In [9], Mayer and Warnke have thoroughly established formulas for the fixed-point in the case of $\rho(|A|) < 1$: this fixed-point is independent of the starting interval $\mathbf{x}_0$. In [1], Arndt and Mayer have established necessary and sufficient condition on A for a fixed-point to exist, when $\rho(|A|) = 1$. In this case, the fixed-point is an interval of nonzero width, that is, a non-degenerate interval. It is well-known that the widths of the iterates diverge when $\rho(|A|) > 1$, and thus that no fixed-point exists in this case. Our goal is to study the speed of divergence of the iterates in this case.

2.3 Different Approaches along with Theoretical Bounds

The main idea is to use an orthogonal change of coordinates which is related to the matrix of the iteration. As the matrix A is kept constant for all iterations (and this is not the case in the more general approach of Lohner, see Section 2.4), the change of coordinates is also kept constant and given by an orthogonal matrix B.

The two orthogonal matrices considered in what follows are either $B = Q$ from the QR-factorization of A, or $B = U$, resp. $B = V'$, from the SVD-factorization of A.

2.3.1 Orthogonal Change of Coordinates

Before diving into the specificities of these changes of coordinates, let us study the general change of coordinates using an orthogonal matrix B, that is, $B^{-1} = B'$, with $x = By \Leftrightarrow y = B^{-1}x$ and its interval counterpart $\mathbf{x} = B\mathbf{y}$. The interval iteration is

$$\left\{ \begin{array}{rcl} \mathbf{x}_{n+1} & = & B\mathbf{y}_{n+1} \\ \mathbf{y}_{n+1} & = & B^{-1}AB\mathbf{y}_n + B^{-1}\mathbf{b}. \end{array} \right.$$

Thus the iteration satisfied by the width of the $\mathbf{y}_n$ is

$$\begin{array}{rcl} \mathrm{wid}(\mathbf{y}_{n+1}) & = & |B^{-1}AB|\mathrm{wid}(\mathbf{y}_n) + |B^{-1}|\mathrm{wid}(\mathbf{b}) \\ & \leq & |B^{-1}|.|A|.|B|\mathrm{wid}(\mathbf{y}_n) + |B^{-1}|\mathrm{wid}(\mathbf{b}) \end{array}$$

where the inequalities are to be understood componentwise. By induction on n,

$$\mathrm{wid}(\mathbf{y}_n) \leq (|B^{-1}|.|A|.|B|)^n.\mathrm{wid}(\mathbf{y}_0) + \sum_{i=0}^{n-1}(|B^{-1}|.|A|.|B|)^i.|B^{-1}|.\mathrm{wid}(\mathbf{b}).$$

Taking norms, one gets

$$\begin{array}{rcl} \|\mathrm{wid}(\mathbf{y}_n)\| & \leq & \left(\| \, |B^{-1}| \, \|.\| \, |A| \, \|.\| \, |B| \, \|\right)^n \|.\mathrm{wid}(\mathbf{y}_0)\| \\ & + & \sum_{i=0}^{n-1} \left(\| \, |B^{-1}| \, \|.\| \, |A| \, \|.\| \, |B| \, \|\right)^i .\| \, |B^{-1}| \, \|.\|\mathrm{wid}(\mathbf{b})\| \\ & \leq & \left(\| \, |B^{-1}| \, \|.\| \, |A| \, \|.\| \, |B| \, \|\right)^n \|.\mathrm{wid}(\mathbf{y}_0)\| \\ & + & \frac{\left(\| \, |B^{-1}| \, \|.\| \, |A| \, \|.\| \, |B| \, \|\right)^n - 1}{\| \, |B^{-1}| \, \|.\| \, |A| \, \|.\| \, |B| \, \| - 1} \| \, |B^{-1}| \, \|.\|\mathrm{wid}(\mathbf{b})\|. \end{array}$$

Remark: if the considered norm is the matrix norm induced by the vector Euclidean norm, then $\| \, |B| \, \|_2 = \|B\|_2$ for any matrix B. Similarly, $\| \, |B| \, \|_\infty = \|B\|_\infty \leq \sqrt{d}$ for any $d \times d$ orthogonal matrix B. In such cases, the bound becomes

$$\begin{array}{rcl} \|\mathrm{wid}(\mathbf{y}_n)\| & \leq & (\kappa(B).\| \, |A| \, \|)^n \, \|\mathrm{wid}(\mathbf{y}_0)\| \\ & + & \frac{(\kappa(B).\| \, |A| \, \|)^n - 1}{\kappa(B).\| \, |A| \, \| - 1} \|B^{-1}\|.\|\mathrm{wid}(\mathbf{b})\|, \end{array}$$

where $\kappa(B)$ denotes $\|B\|.\|B^{-1}\|$, the condition number of B for the problem of solving a linear system.

Since $\|B\|_2 = \|B^{-1}\|_2 = \kappa_2(B) = 1$ for an orthogonal matrix B, this bound simplifies even further with the Euclidean norm:

$$\|\mathrm{wid}(\mathbf{y}_n)\| \quad \leq \quad \|A\|^n.\|\mathrm{wid}(\mathbf{y}_0)\| + \frac{\|A\|^n - 1}{\|A\| - 1}.\|\mathrm{wid}(\mathbf{b})\|.$$

In other words, theoretically there is no difference in the bounds on the widths of the iterates, whether an orthogonal change of coordinates takes place or not.

In what follows, we assume again A diagonalizable so as to simplify the presented proofs (this will not necessarily be the case for the experiments) and A can be diagonalized as $A = P^{-1}\Lambda P$ where Λ is diagonal. If we replace A by $P^{-1}\Lambda P$ in the iteration, one gets the mathematically equivalent formulation

$$\begin{array}{rcl} \mathbf{y}_{n+1} & = & B^{-1}P^{-1}\Lambda PB\mathbf{y}_n + B^{-1}\mathbf{b} \\ & = & (PB)^{-1}\Lambda(PB)\mathbf{y}_n + B^{-1}\mathbf{b}, \end{array}$$

thus

$$\mathrm{wid}(\mathbf{y}_n) = (|(PB)^{-1}|.|\Lambda|.|PB|).\mathrm{wid}(\mathbf{y}_n) + |B^{-1}|.\mathrm{wid}(\mathbf{b}),$$

and by induction

$$\mathrm{wid}(\mathbf{y}_n) = (|(PB)^{-1}|.|\Lambda|.|PB|)^n.\mathrm{wid}(\mathbf{y}_0) + \sum_{i=0}^{n-1}(|(PB)^{-1}|.|\Lambda|.|PB|)^i.|B^{-1}|.\mathrm{wid}(\mathbf{b}).$$

Taking the Euclidean norm of vectors and the induced matrix norm, one gets

$$\begin{array}{rcl} \|\mathrm{wid}(\mathbf{y}_n)\|_2 & \leq & (\kappa_2(PB)\|\Lambda\|_2)^n .\|\mathrm{wid}(\mathbf{y}_0)\|_2 \\ & + & \frac{(\kappa_2(PB)\|\Lambda\|_2)^n - 1}{\kappa_2(PB)\|\Lambda\|_2 - 1}.\|\mathrm{wid}(\mathbf{b})\|_2. \end{array}$$

Let us note that $\kappa(PB) = \kappa(P)$. Furthermore, as Λ is diagonal, $\|\Lambda\|$ is the largest eigenvalue (in module) of A, that is, $\|\Lambda\| = \rho(A) < 1$. This implies

$$\|\mathrm{wid}(\mathbf{y}_n)\|_2 \quad \leq \quad (\kappa_2(P)\rho(A))^n .\|\mathrm{wid}(\mathbf{y}_0)\|_2 + \frac{(\kappa_2(P)\rho(A))^n - 1}{\kappa_2(P)\rho(A) - 1}.\|\mathrm{wid}(\mathbf{b})\|_2,$$

This inequality puts in evidence the influence of the condition number of P, the matrix of eigenvectors. For instance, in the ideal case where the eigenvectors form an orthonormal basis, no overestimation occurs.

2.3.2 Use of the QR Factorization

When the orthogonal change of coordinates involves Q from the QR-factorization of A, the algorithm can be written as

$$\begin{array}{rcl} A & = & QR, \\ x_{n+1} & = & Qy_{n+1} \\ \Leftrightarrow y_{n+1} & = & Q'x_{n+1} \\ y_{n+1} & = & Q'AQy_n + Q'b \end{array}$$

In exact arithmetic, one should get

$$y_{n+1} = RQy_n + Q'b.$$

The interval counterpart is

$$\begin{array}{rcl} \mathbf{x}_{n+1} & = & Q\mathbf{y}_{n+1} \\ \mathbf{y}_{n+1} & = & Q'AQ\mathbf{y}_n + Q'\mathbf{b}. \end{array}$$

2.3.3 Use of the SVD Factorization

Our second and third proposals consist in using respectively U and V from the SVD-factorization of A: from $A = U\Sigma V'$, we use either $B = U$ or $B = V'$, which yields

$$\begin{array}{rcl} x_{n+1} & = & Uy_{n+1} \\ \Leftrightarrow y_{n+1} & = & U'x_{n+1}, \\ y_{n+1} & = & U'AUy_n + U'b. \end{array}$$

In exact arithmetic, this corresponds to

$$y_{n+1} = \Sigma VUy_n + U'b.$$

The interval counterpart is

$$\begin{array}{rcl} \mathbf{x}_{n+1} & = & U\mathbf{y}_{n+1} \\ \mathbf{y}_{n+1} & = & U'AU\mathbf{y}_n + U'\mathbf{b}. \end{array}$$

$$\begin{array}{rcl} x_{n+1} & = & V'y_{n+1} \\ \Leftrightarrow y_{n+1} & = & Vx_{n+1}, \\ y_{n+1} & = & VAV'y_n + Vb. \end{array}$$

In exact arithmetic, this corresponds to

$$y_{n+1} = VU\Sigma y_n + Vb.$$

The interval counterpart is

$$\begin{array}{rcl} \mathbf{x}_{n+1} & = & V'\mathbf{y}_{n+1} \\ \mathbf{y}_{n+1} & = & VAV'\mathbf{y}_n + V\mathbf{b}. \end{array}$$

Remark: VU is also an orthogonal matrix.

2.4 Lohner's QR Method

A well-known approach is given in Lohner, e.g. in [8] and studied in details by Nedialkov and Jackson in [10]. It is usually presented for the iteration $x_{n+1} = A_n x_n + b_n$, that is when the matrix and the affine term vary at each iteration.

Lohner's QR method consists in performing the following iteration:

$$\left\{ \begin{array}{lcl} \multicolumn{3}{l}{y_0 = x_0,\ Q_0 = I,\ [Q_1, R_1] = qr(A) \text{ that is, } A = Q_1R_1} \\ {[Q_{n+1}, R_{n+1}]} & = & qr(R_nQ_n) \\ y_{n+1} & = & Q'_{n+1}AQ_ny_n + Q'_{n+1}b \\ x_{n+1} & = & Q_{n+1}y_{n+1} \end{array} \right.$$

and its interval counterpart is

$$\left\{ \begin{array}{lcl} \multicolumn{3}{l}{\mathbf{y}_0 = \mathbf{x}_0,\ Q_0 = I,\ [Q_1, R_1] = qr(A) \text{ that is, } A = Q_1R_1} \\ {[Q_{n+1}, R_{n+1}]} & = & qr(R_nQ_n) \\ \mathbf{y}_{n+1} & = & Q'_{n+1}AQ_n\mathbf{y}_n + Q'_{n+1}\mathbf{b} \\ \mathbf{x}_{n+1} & = & Q_{n+1}\mathbf{y}_{n+1}. \end{array} \right.$$

In the case of a constant – throughout the iterations – matrix A, one can recognize Francis' and Kublanovskaya's QR-algorithm. Using the convergence of (R_n) towards the matrix of eigenvalues of A (or towards its Schur form), in [10], Nedialkov and Jackson established the following bounds:

$$w(\mathbf{x}_n) \leq \text{cond}(P)\rho(A)^n w(\mathbf{x}_0) + \frac{\text{cond}(P)\rho(A)^{n-1} - 1}{\text{cond}(P)\rho(A) - 1} w(\mathbf{b}) + \mathbf{b}$$

where we recall A diagonalizable: $A = P^{-1}\Lambda P$.

2.5 Comparison

Two aspects are compared: the complexity and the accuracy, that is, the bounds on the widths of the iterates, of each method.

Let us first examine the computational complexity. Let us recall that the QR-factorization, resp. SVD-factorization, of a $d \times d$ matrix has a computational complexity of $\mathcal{O}(d^3)$. In the algorithms of Sections 2.3.2 and 2.3.3, the factorization of a matrix is performed only once, and not at every iteration: for n iterations, these algorithms thus have complexity $\mathcal{O}(d^3 + nd^2)$. In comparison, Lohner's QR method has complexity $\mathcal{O}(nd^3)$, which is significantly larger when d is large. In comparison, the cost of the factorization is negligible when the number n of iterations is large.

Let us now compare the accuracy of these different methods, from a theoretical point of view. The bounds we get on the width of the iterate $\mathbf{x}_n$ are larger than the bounds obtained by Nedialkov and Jackson, as the condition number of the diagonalizing matrix P appears to the n-th power in the formula for the QR- and SVD-algorithms, whereas it appears without this n-th power in the bound for Lohner's QR-algorithm. As a condition number is always larger or equal to 1, this means that the bound for Lohner's QR-algorithm is tighter than the bounds for the QR- and SVD-algorithms.

3 Experiments

3.1 Experimental Setup

After the results on the widths of the iterates of the toy example given in Section 1.1, Section 3.2 presents the computation of each corner of the initial box, to illustrate that it is possible to get tight enclosures, on such a small example.

All algorithms presented in this paper, namely the naive (or brute-force) application of the iteration, the QR-algorithm of Section 2.3.2, the two versions of the SVD-algorithm of Section 2.3.3, and Lohner's QR algorithm given in Section 2.4 have been implemented in Octave using Heimlich' interval package [4], then in Matlab using Rump's Intlab package [13]. Two other methods have been implemented and compared. The first technique [11] consists in the determination of k such that $\rho(|A^k|) < 1$, then it computes only one iterate every k step, in other words it computes

$$x_{(k+1)n} = A^k x_{kn} + \sum_{i=0}^{k-1} A^i b :$$

this iteration converges even when interval arithmetic is employed. The other technique is the use of affine arithmetic [3], as advocated by Rump in a private communication. The affine arithmetic employed here is the one available in Intlab [12]. In the experimental results presented below, each technique is associated to a color, as shown in Table 2.

Table 2: Color code for each method

algorithm	**color**
brute force	black
QR	cyan
SVD U	red
SVD V	magenta
Lohner's QR	dark blue
every k-th iterate	green
affine arithmetic	gray

The factorizations use only the basic QR and SVD factorizations available in Matlab, but neither the pivoted QR recommended by Lohner in [8] nor more elaborate versions presented by Higham in [5]. Very preliminary experiments with Lohner's pivoted QR factorization (not presented here) do not exhibit any difference in the width of divergence.

Sections 3.3 and 3.4 contain the evolution of the radii of the iterates computed by these different techniques, for two matrix dimensions: 10×10 and 100×100. The y-axis for the radii uses a logarithmic scale. For both dimensions, four kinds of matrices A have been used for the experiments. On the one hand, matrices which are well-conditioned (with a condition number of order 10^2) and ill-conditioned (with a condition number of order 10^{10}) have been generated. On the other hand, the scaling of the matrices varies: matrices which are well-scaled and matrices which are ill-scaled, with the order of magnitude of their coefficients varying between 1 and 10^{10}. More specifically, well-scaled matrices A are generated by a call to `randsvd` with the prescribed condition number. It is known that these matrices are well-scaled: all coefficients are of the same order of magnitude, they are said to be scaled, or equilibrated [6, Section 7.2]. Ill-scaled matrices are obtained as DAD^{-1} where the matrices A are generated by a call to `randsvd` as before and D is a diagonal matrix with diagonal elements varying in magnitude between 1 and 10^{10}: the coefficients of the resulting matrix also vary in magnitude, they are unscaled.

These denominations of "well-scaled" and "ill-scaled", "well-conditioned" and "ill-conditioned" are qualitative and not totally quantitative, as the matrices, originally generated by a call to Matlab's `randsvd` were then added to a multiple of the identity matrix and multiplied by a constant, in order to satisfy both $\rho(A) < 1$ and $\rho(|A|) > 1$. It can also be noted that degrading the scaling of the matrix also degrades its condition number; in other words, a "well-conditioned ill-scaled" matrix has a much worse condition number than a "well-conditioned well-scaled" matrix, even if the required condition numbers, in the call to `randsvd`, are initially the same.

Preliminary experiments with interval matrices, that is, matrices with interval coefficients, put into evidence two difficulties. First, the divergence is much faster:

with an interval matrix whose width is a thousandth of the center matrix, it takes less than 5 iterations to get a width that is several millions larger than the width of the iterates obtained using the center matrix, or even more. Second, accounting for interval matrices in the algorithms must be done with care: indeed, QR and SVD factorizations apply only to point matrices, thus they are used for the center matrices, but then the rest of the algorithm must correctly incorporate the interval parts of the matrices.

All experiments have been performed on a 2.7 GHz Quad-Core Intel Core i7 with 16GB RAM. Timings are averaged over 100 executions, except for affine arithmetic where at most 10 executions were performed.

3.2 Toy Example

First, the toy example presented in Section 1.1 is considered. As the iteration is affine, one can compute separately the images of the endpoints of the initial vector, to get the endpoints of the successive iterates. That is, we compute separately

$$x_n = 1.8 * x_{n-1} - 0.9 * x_{n-2} + 4.7.10^{-2} * 3 * \mathbf{u}$$

for $x_0 = 0$ and $x_1 = 1$ and for $x_0 = 0$ and $x_1 = 1.1$. However, we use the interval vector $\mathbf{u} = [9.95, 10.05]$ in the iteration. The convex hull of the 10 first iterates are represented on the left part of Figure 1. It is obvious that the width of the successive iterates grows rapidly.

Then we compute separately

$$x_n = 1.8 * x_{n-1} - 0.9 * x_{n-2} + 4.7.10^{-2} * 3 * u$$

for $x_0 = 0$ and $x_1 = 1$ and for $x_0 = 0$ and $x_1 = 1.1$, and for $u = 9.95$ and $u = 10.05$. The convex hull of the 10 first iterates are represented on the right part of Figure 1. In this case, the width of the successive iterates remains small, of the order of magnitude of 1% of the midpoint of the interval.

In this toy example, the iterations had to be performed 4 times, that is, once for each corner of the initial values, in order to get a tight enclosure. This can clearly not be generalized to high dimensions, as the number of corners grows as 4^d with the dimension d of the problem.

3.3 Example of dimension 10

Figure 2 gives the radii (in logarithmic scale) for the successive iterates computed by the methods detailed above. When the number of iterations is large (visually, above 30 or 40 iterations), the iterates computed by all methods presented in Section 2 diverge rapidly, as can be seen on the plots on the left. When one concentrates on the first iterations, the behaviours compare differently. One can also note that unscaling the matrix A speeds the divergence, for all methods. On the contrary, the k-step method and the use of affine arithmetic preserve the convergence of the iterates.

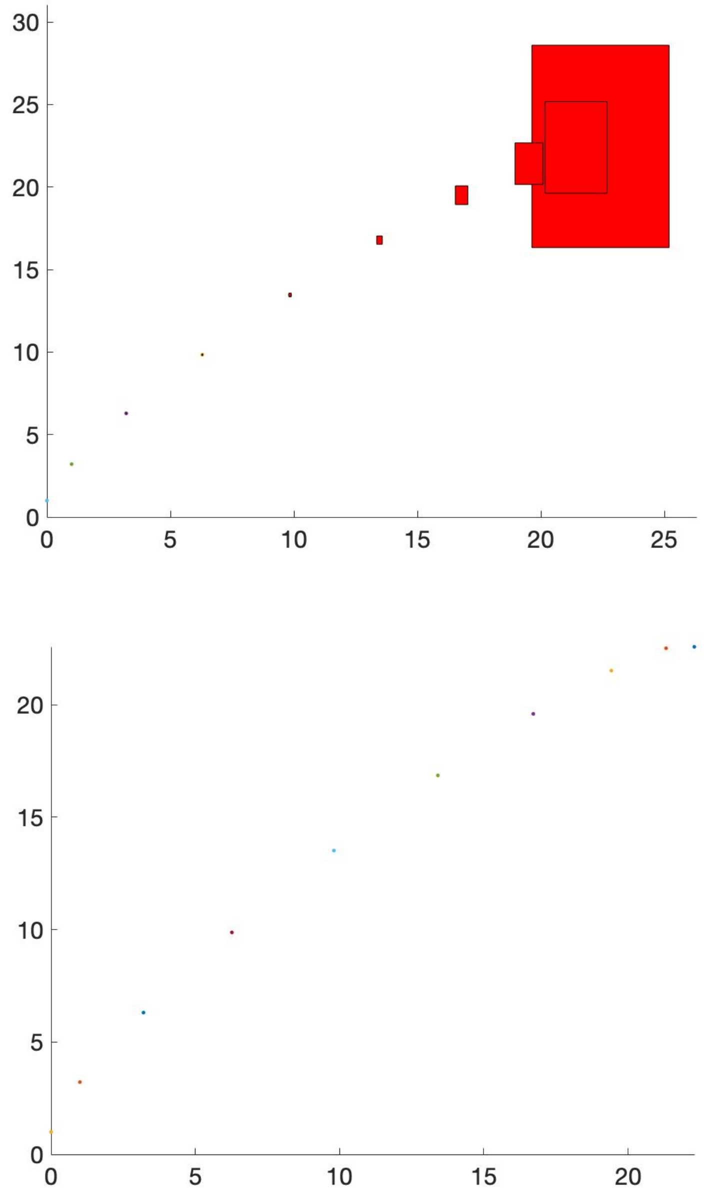
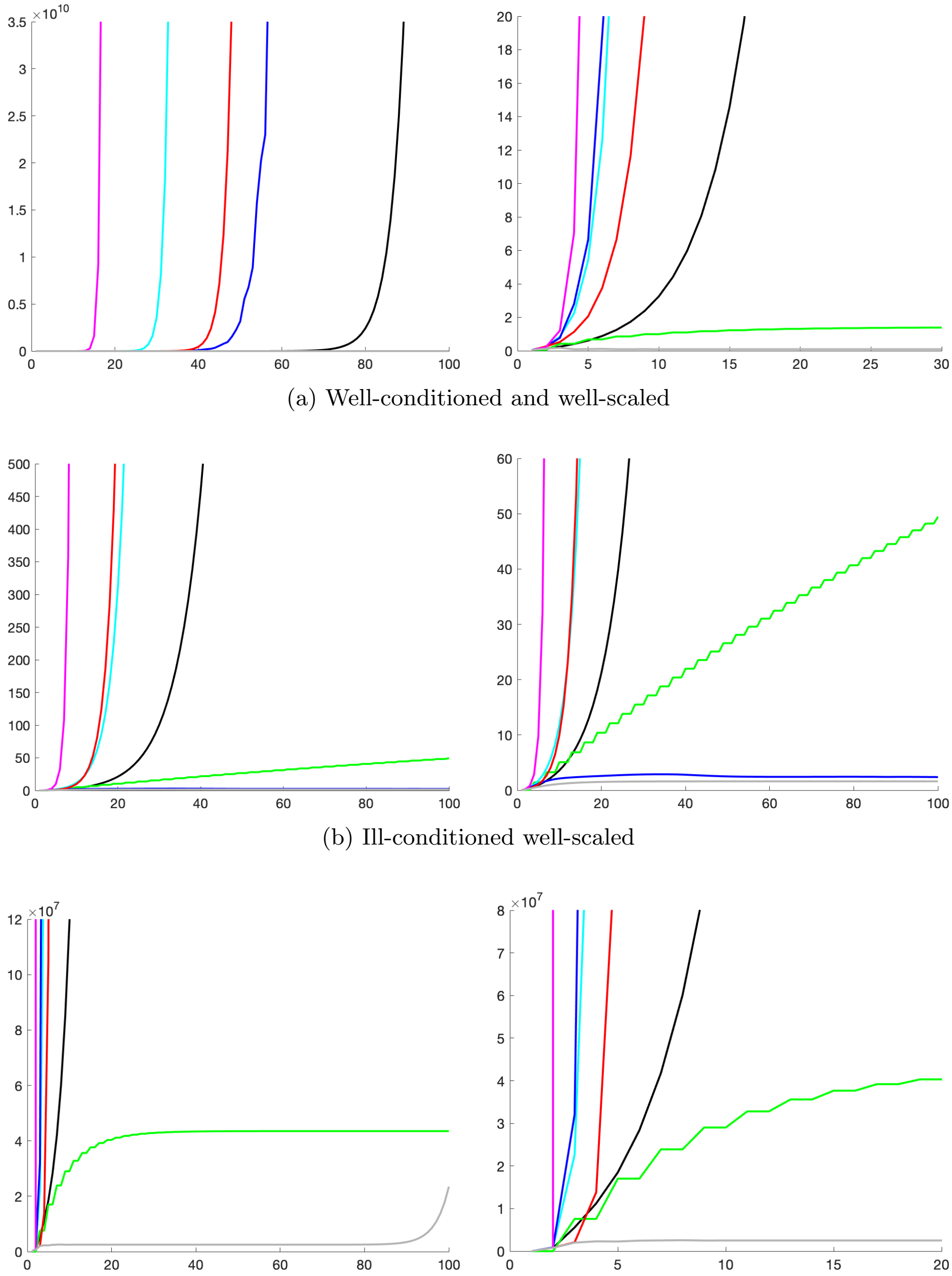


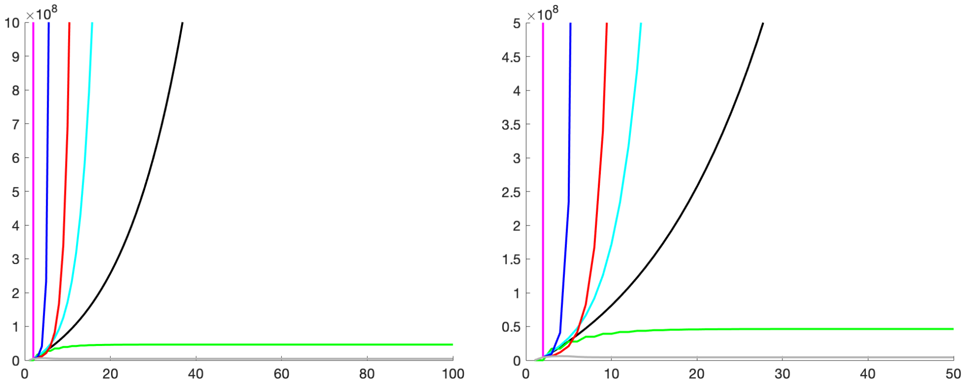
Figure 1: Top: the 10 first iterates of the toy example, where the endpoints of $\mathbf{x}_0$ are considered separately. Bottom: the 10 first iterates of the toy example, computed corner by corner.



(a) Well-conditioned and well-scaled

(b) Ill-conditioned well-scaled

(c) Well-conditioned ill-scaled



(d) Ill-conditioned ill-scaled

Figure 2: The case of a 10 × 10 matrix (right part: zoom of the left part)

Table 3: Timings for 10 × 10 examples

method	**well-cond. well-scaled**	**ill-cond. well-scaled**	**well-cond. ill-scaled**	**ill-cond. ill-scaled**
naive	0.0088	0.0084	0.0093	0.0086
***k*-th step**	0.0044	0.0015	0.0043	0.0045
QR	0.0161	0.0155	0.0160	0.0157
SVD U	0.0162	0.0156	0.0161	0.0166
SVD V	0.0147	0.0145	0.0324	0.0332
Lohner's QR	0.0170	0.0164	0.0165	0.0177
affine arith.	8.1859	8.7911	8.6595	8.2754

The timings in seconds are given in Table 3.

The methods presented in Sections 2.3.2, 2.3.3 and 2.4 all exhibit similar execution times. The naive method performs less operations and is thus faster. The k-th step method is fast as well, the variations in its execution time are due to the preprocessing, that is to the determination of the power k such that $\rho(|A^k|) < 1$: the execution time is larger when k is larger. With this method, the convergence is good. The use of affine arithmetic significantly slows down the computations, however the iterates converge.

3.4 Example of dimension 100

Figure 3 gives the radii (in logarithmic scale) for the successive iterates computed by the methods detailed in Section 3.1. When the number of iterations is large (visually, above 40 or 50 iterations), the iterates computed by all methods, except the k-step method, diverge rapidly, as can be seen on the plots on the left. Again, when one concentrates on the first iterations, the behaviours compare differently.

The timings in seconds are given in Table 4.

Table 4: Timings for 100 × 100 examples

method	**well-cond.** **well-scaled**	**ill-cond.** **well-scaled**	**well-cond.** **ill-scaled**	**ill-cond.** **ill-scaled**
naive	0.0163	0.0145	0.0142	0.0142
k**-th step**	0.0046	0.0071	0.0048	0.0072
QR	0.1577	0.0363	0.0408	0.0409
SVD U	0.0427	0.0420	0.0437	0.0437
SVD V	0.0423	0.0390	0.0643	0.0638
Lohner's QR	0.0611	0.0758	0.0646	0.0804
affine arith.	70.8724	72.6441	76.6102	71.2518

The comments on the timings apply again, with the exception of the use of affine arithmetic, which is still much slower but does not manage any more to preserve the convergence very long. Indeed, the exponential growth applies when affine arithmetic is applied, as well as when other methods are employed; as the exponential growth occurs to the smaller terms (called the *noise*), affine arithmetic simply delays the divergence phenomenon.

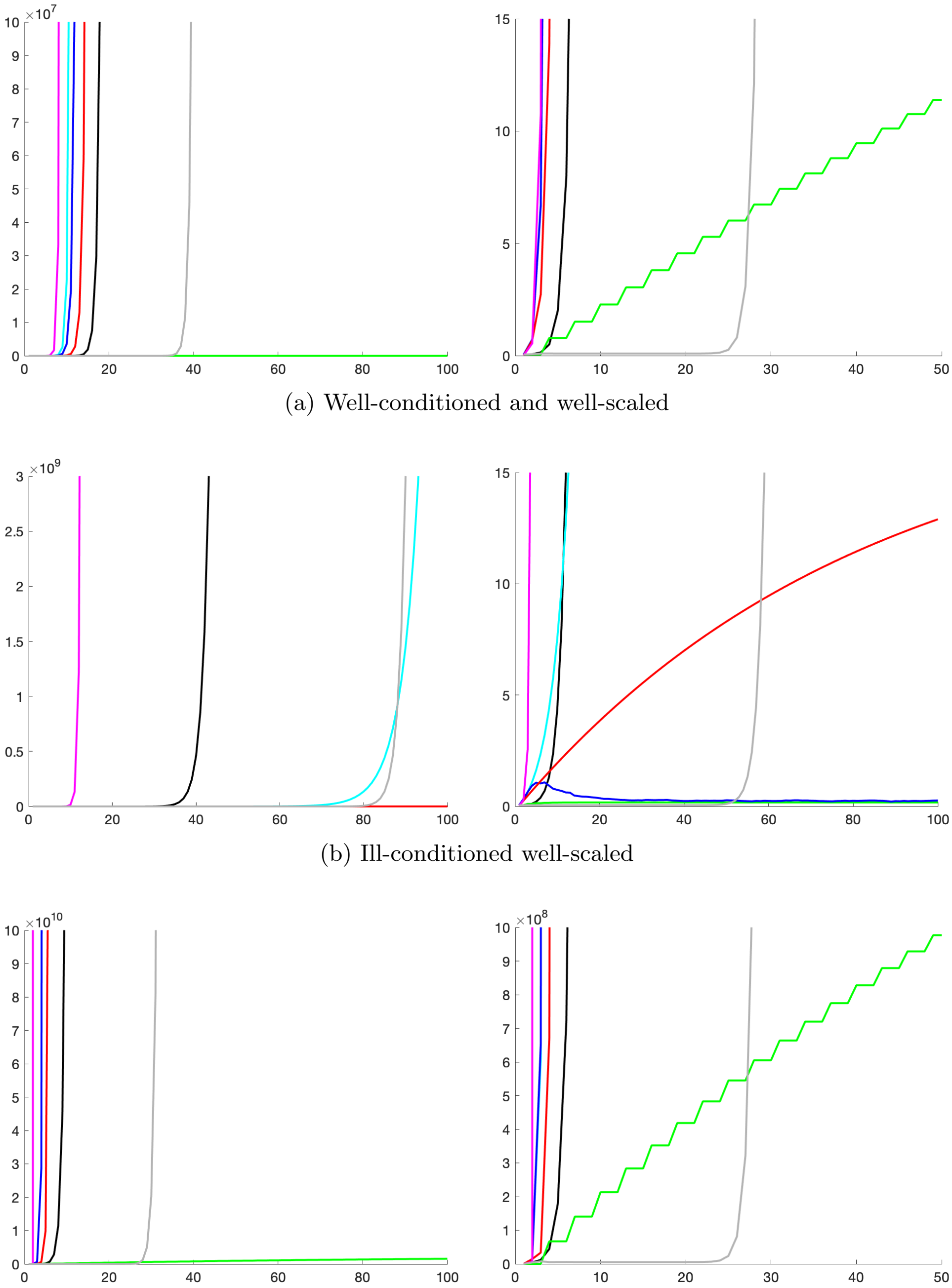
3.5 Comments

One can note that the k-step method, that is the method that resorts to a convergent interval iteration, performs very well at a moderate computation cost. Even the preprocessing time to determine the value of k has a negligible cost.

This method is a totally ad hoc approach for this problem and cannot be generalized. However, in the framework of filters and control theory, it has a physical meaning: the divergence of the iterations can be attributed to a sampling time which is too small to allow for significant variations to be observed. Multiplying the sampling time by k means sampling less frequently (by a factor k) and thus being able to measure the evolution of the observed quantities.

The use of affine arithmetic, on the contrary, is a very general method and it exhibits a very good accuracy, even if it eventually diverges (see the experiments with the 100×100 matrices in Section 3.4). The counterpart is the execution time, which is at least a thousand times larger than for the other methods. This is not an issue for the experiments presented here, as the time is of order of magnitude of a minute.

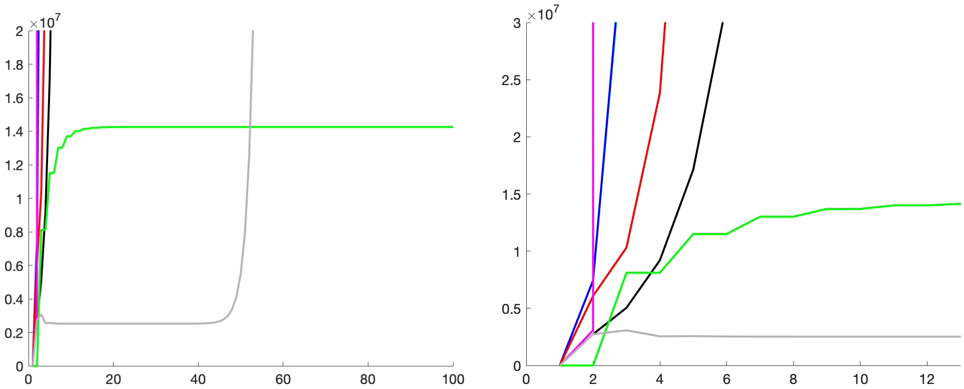
The methods based on the QR or SVD factorizations of the matrix A were developed with geometric principles in mind. For the QR-algorithm, the idea was to align the current box with the directions that are preserved by the product by A,



(a) Well-conditioned and well-scaled

(b) Ill-conditioned well-scaled

(c) Well-conditioned ill-scaled



(d) Ill-conditioned ill-scaled

Figure 3: The case of a 100 × 100 matrix (right part: zoom of the left part)

with a tradeoff between aligning the box along the eigenvectors and preserving an orthonormal system of coordinates, hence the choice of Q. For the SVD-algorithm, the idea was to align the box along the direction which gets the maximal elongation, that is along the singular vector corresponding to the largest singular value.

In both cases, the benefit of these geometric transformations is mitigated with the overestimation implied by extra computations, and there is either no clear benefit for the QR-based approach, or a delicate balance for the SVD-based approach. The SVD-algorithm is interesting when the matrix is ill-scaled, and particularly for the first iterations.

The methods of choice remain either the naive approach, when the matrix A is well-conditioned and well-scaled, or Lohner's QR method when the matrix is ill-conditioned. Surprisingly, the overhead of Lohner's QR method, in terms of computational time, is not as large as the formula for its complexity implies.

Our general recommendation is thus:

- to preprocess the matrix A in order to scale it (see [6, Section 7.3] about diagonal scaling);

- then to execute in parallel the naive approach and Lohner's QR approach, in order to converge reasonably well for any condition number of A.

Affine arithmetic is a solution of choice when other solutions fail and when the analysis and developing time is a scarce resource.

4 Conclusion and Future Work

This study, both theoretical and experimental, has compared several approaches to counteract the wrapping effect for the computation of affine iterations. Geometric considerations have led to the proposed algorithms. The benefit of these approaches is not always clear, as a better configuration is obtained through extra-computations and thus extra-overestimation. To deepen this geometric approach, we will aim at simplifying the resulting formulas, at getting formulas that are closer to the mathematically equivalent, but simpler, ones that are given after each proposed transformation. The main difficulty is to perform products such as $Q.Q'$ or $U'.U$, without replacing them by the identity, but in a certified and tight way. As the SVD-based approach seems more promising, our future work will concentrate on the use of a certified SVD factorization, as proposed by van der Hoeven and Yakoubsohn in [14]. We also plan to consider an interval version of the matrix, using the results in [7] to keep guarantees on the singular quantities involved in the computations.

References

[1] Arndt, H-R. and Mayer, G. On the semi-convergence of interval matrices. *Linear Algebra and its Applications*, 393:15–35, 2004. DOI: 10.1016/j.laa.2003.10.015.

[2] Beaumont, O. Solving interval linear systems with oblique boxes. Technical Report 1315, Irisa, 2000. ftp://ftp.irisa.fr/techreports/2000/PI-1313.ps.gz.

[3] De Figueiredo, L.H. and Stolfi, J. Self-validated numerical methods and applications. In *Monograph for 21st Brazilian Mathematics Colloquium, IMPA, Rio de Janeiro*, Volume 5, 1997.

[4] Heimlich, O. Interval arithmetic in GNU Octave. In *SWIM 2016: Summer Workshop on Interval Methods, France*, 2016. https://swim2016.sciencesconf.org/data/SWIM2016_book_of_abstracts.pdf#page=27.

[5] Higham, N.J. QR factorization with complete pivoting and accurate computation of the SVD. *Linear Algebra and its Applications*, 309:153–174, 2000. DOI: 10.1016/S0024-3795(99)00230-X.

[6] Higham, N.J. *Accuracy and Stability of Numerical Algorithms (2nd edition)*. SIAM, 2002. DOI: 10.1137/1.9780898718027.

[7] Hladik, M., Daney, D., and Tsigaridas, E. Bounds on real eigenvalues and singular values of interval matrices. *SIAM J. Matrix Analysis and Applications*, 31(4):2116–2129, 2010. DOI: 10.1137/090753991.

[8] Lohner, R. On the ubiquity of the wrapping effect in the computation of error bounds. In Kulisch, Lohner, Facius, editor, *Perspectives on Enclosures*

Methods, pages 201–217. Springer, 2001. DOI: 10.1007/978-3-7091-6282-8_12.

[9] Mayer, G. and Warnke, I. On the fixed points of the interval function $[f]([x]) = [a][x] + [b]$. *Linear Algebra and its Applications*, 363:202–216, 2003. DOI: 10.1016/S0024-3795(02)00254-9.

[10] Nedialkov, N. and Jackson, K. A new perspective on the wrapping effect in interval methods for initial value problems for ordinary differential equations. In Kulisch, Lohner, Facius, editor, *Perspectives on Enclosures Methods*, pages 219–264. Springer, 2001. DOI: 10.1007/978-3-7091-6282-8_13.

[11] Revol, N. Convergent linear recurrences (with scalar coefficients) with divergent interval simulations. In *SCAN: 11th GAMM – IMACS Int. Symp. on Scientific Computing, Computer Arithmetic, and Validated Numerics (Japan)*, 2004.

[12] Rump, Siegfried M. and Kashiwagi, Masahide. Implementation and improvements of affine arithmetic. *Nonlinear Theory and Its Applications, IEICE*, 6(3):341–359, 2015. DOI: 10.1587/nolta.6.341.

[13] Rump, S.M. INTLAB – INTerval LABoratory. In Csendes, Tibor, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999. DOI: 10.1007/978-94-017-1247-7_7.

[14] van der Hoeven, J. and Yakoubsohn, J.-C. Certified Singular Value Decomposition. Technical report, Laboratoire d'informatique de l'École Polytechnique Palaiseau, 2018. https://hal.archives-ouvertes.fr/hal-01941987.