Polynomial optimization and a Jacobi–Davidson type method for commuting matrices

Ivo W. M. Bleylevens[a], Michiel E. Hochstenbach[b], Ralf L. M. Peeters[a]

[a]*Department of Knowledge Engineering*
Maastricht University, P.O. Box 616, 6200 MD, Maastricht, The Netherlands
Tel: 0031-43-3883455, Fax: 0031-43-3884910
[b]*Department of Mathematics and Computer Science*
TU Eindhoven, P.O. Box 513, 5600 MB, Eindhoven, The Netherlands

Abstract

In this paper we introduce an new Jacobi–Davidson type eigenvalue solver for a set of commuting matrices, called JDCOMM, used for the global optimization of so-called Minkowski-norm dominated polynomials in several variables. The Stetter–Möller matrix method yields such a set of real non-symmetric commuting matrices since it reformulates the optimization problem as an eigenvalue problem. A drawback of this approach is that the matrix most relevant for computing the global optimum of the polynomial under investigation is usually large and only moderately sparse. However, the other matrices are generally much sparser and have the same eigenvectors because of the commutativity. This fact is used to design the JDCOMM method for this problem: the most relevant matrix is used only in the outer loop and the sparser matrices are exploited in the solution of the correction equation in the inner loop to greatly improve the efficiency of the method. Some numerical examples demonstrate that the method proposed in this paper is more efficient than approaches that work on the main matrix (standard Jacobi–Davidson and implicitly restarted Arnoldi), as well as conventional solvers for computing the global optimum, i.e., SOSTOOLS, GloptiPoly, and PHCpack.

Keywords: Multivariate polynomial optimization, global optimization, solving systems of polynomial equations, Stetter–Möller matrix method, commuting matrices, Jacobi–Davidson, correction equation, Arnoldi
2000 MSC: 12D10, 13P10, 13P15, 65H10, 65H17, 65F15, 65F50, 65K10

1. Introduction

Finding the global minimum of a real-valued multivariate polynomial is a problem that has several useful applications in system and control theory [9] as well as in many other fields including statistics, mathematical finance, economics, systems biology, etc. Multivariate global polynomial optimization is often challenging because of the non-convexity of the problem and the existence of local optima. In this paper we design a new Jacobi–Davidson type eigenvalue solver which efficiently computes the global optimum of a special class of polynomials.

This special class under consideration is the so-called Minkowski-norm dominated class of polynomials in several variables (see Section 2). The problem of finding a global minimum of a polynomial from this class can be reformulated as an eigenvalue problem by applying the Stetter–Möller matrix method [8, 21]. This yields a set of real nonsymmetric large commuting matrices $A_p, A_{x_1}, \ldots, A_{x_n}$, where A_p is usually moderately sparse, while the matrices $A_{x_1}, \ldots, A_{x_n}$ are generally very sparse. It turns out that the leftmost real eigenvalue (i.e., with smallest real part) of the matrix A_p gives the value of the global optimum. More details, including some algebraic backgrounds, are given in Section 2.

Email addresses: `i.bleylevens@maastrichtuniversity.nl; corresponding author` (Ivo W. M. Bleylevens), `ralf.peeters@maastrichtuniversity.nl` (Ralf L. M. Peeters)
URL: `http://www.win.tue.nl/~hochsten/` (Michiel E. Hochstenbach)

This approach for global optimization of a Minkowski-norm dominated polynomial was previously described in [1]. In that paper the focus lies on improving the efficiency of computing the *action* of the matrices $A_p, A_{x_1}, \ldots, A_{x_n}$ on a vector by the development of a matrix-free approach. Thus, the explicit construction of the large matrices is avoided by associating the system of first-order conditions with an nD-system of difference equations.

Because only one eigenvalue of the large matrix A_p is needed, it is a plausible choice to consider iterative eigenvalue solvers. We here describe a new Jacobi–Davidson (JD) [27, 11] type method for this problem. The essence of this new approach is that it computes an eigenvalue of the (moderately) sparse matrix A_p while using one of the much sparser matrices $A_{x_1}, \ldots, A_{x_n}$ in the inner loop, resulting in a much more efficient process. Hereby, it attempts to compute the smallest real eigenvalue of the matrix A_p. The assumption we make here is that all the (relevant) eigenvalues have algebraic multiplicity one. Pseudocode for the resulting JD method for a set of commuting matrices, that will be denoted by JDCOMM, is given in Section 3.

In Section 4 the proposed approach is illustrated by means of some experiments and its performance is compared to that of a conventional Jacobi–Davidson method, two methods based on implicitly restarted Arnoldi (Matlab's eigs and Krylov–Schur), and three other methods for multivariate polynomial optimization: SOSTOOLS, GloptiPoly, and PHCpack. Finally, we end with some concluding remarks in Section 5.

2. Algebraic Background

In this section we briefly recall some relevant background information; for more details we refer to [3]. The special class of polynomials under consideration is the class of Minkowski-norm dominated polynomials which was previously studied in [1]. This class concerns polynomials of the form:

$$p_\beta(x_1, \ldots, x_n) = \beta\,(x_1^{2d} + \cdots + x_n^{2d}) + q(x_1, \ldots, x_n), \tag{1}$$

where $q(x_1, \ldots, x_n)$ is a real polynomial of total degree less than $2d$, and β is a positive real number. The total degree of the polynomial

$$q(x_1, \ldots, x_n) = \sum_{\gamma_1, \ldots, \gamma_n} a_{\gamma_1, \ldots, \gamma_n}\, x_1^{\gamma_1} \cdots x_n^{\gamma_n} \tag{2}$$

is defined as usual as: $\max\{\gamma_1 + \cdots + \gamma_n \;:\; a_{\gamma_1, \ldots, \gamma_n} \neq 0\}$ (see [2]). This class of dominated polynomials is of interest because information about the global minimum of q may be obtained from p_β by letting β tend to zero; see [8, 13]. The presence of the term $\beta\,(x_1^{2d} + \cdots + x_n^{2d})$ ensures that p_β has a global minimum because of its dominating behavior. The first-order conditions of a polynomial p_β, given by:

$$\frac{\partial p_\beta}{\partial x_1}(x_1, \ldots, x_n) = 0, \quad \ldots \quad , \frac{\partial p_\beta}{\partial x_n}(x_1, \ldots, x_n) = 0, \tag{3}$$

yield a system of polynomial equations. The location of the global minimum of p_β corresponds to a solution of the set of first-order conditions (3).

Because $2d$ exceeds the total degree of the polynomial $q(x_1, \ldots, x_n)$, the leading monomials of $\frac{\partial p_\beta}{\partial x_i}$, for $i = 1, \ldots, n$, are x_i^{2d-1}, with respect to any ordering of the monomials of $\frac{\partial p_\beta}{\partial x_i}$ based on their total degree (total degree monomial ordering). Because of this property (which is due to the special structure of the polynomial p_β) the system of first-order conditions is in a Gröbner basis form with respect to any total degree monomial ordering (see [2]). Recall that computation of a Gröbner basis for a system of polynomials can be seen as a generalization of Gaussian elimination for a system of linear equations.

Proposition 1. *$I = \langle \frac{\partial p_\beta}{\partial x_1}, \ldots, \frac{\partial p_\beta}{\partial x_n} \rangle$, the ideal generated by the partial derivatives of p_β, is a zero-dimensional ideal in $\mathbb{R}[x_1, \ldots, x_n]$. This implies that the system of first-order conditions (3) admits a finite number of solutions in $\mathbb{C}^n$.*

Proof. See [3], [8, Thm. 2.1, Prop. 3.1], and the references given therein. □

Now we consider the quotient space $\mathbb{R}[x_1,\ldots,x_n]/I$, which is a finite dimensional linear vector space of dimension $N = (2d-1)^n$. We let

$$B = \{x_1^{\alpha_1}\cdots x_n^{\alpha_n} \ : \ \alpha_1,\ldots,\alpha_n \in \{0,1,\ldots,2d-2\}\} \tag{4}$$

be the monomial basis for this quotient space. For $i = 1,\ldots,n$, we introduce the linear multiplication operators $\mathcal{A}_{x_i}$ on $\mathbb{R}[x_1,\ldots,x_n]/I$ by:

$$\mathcal{A}_{x_i} \ : \ \mathbb{R}[x_1,\ldots,x_n]/I \ \to \ \mathbb{R}[x_1,\ldots,x_n]/I \ \ : \ \ g \ \ \mapsto \ \ x_i \cdot g. \tag{5}$$

Each operator represents multiplication by x_i modulo the ideal I. With respect to the monomial basis B, each (linear) operator $\mathcal{A}_{x_i}$ is represented by an $N \times N$ real nonsymmetric matrix A_{x_i}. In such a matrix A_{x_i} the entry in position (k,l) denotes the coefficient of the kth monomial of B in the normal form of the product of x_i and the lth monomial of B. The matrix A_{x_i} is commonly called the ith companion matrix [21], the multiplication table [31] or the representation matrix [22] for the variable x_i. The matrices A_{x_i} have the following natural properties.

Proposition 2. *We have the following properties:*

(i) The matrices $A_{x_1},\ldots,A_{x_n}$ commute pairwise:

$$A_{x_i}A_{x_j} = A_{x_j}A_{x_i} \quad \textit{for all} \quad i,j \in \{1,\ldots,n\}. \tag{6}$$

(ii) The set $A_{x_1},\ldots,A_{x_n}$ is a matrix solution for the system of equations (3):

$$\frac{\partial p_\beta}{\partial x_i}(A_{x_1},\ldots,A_{x_n}) = 0 \quad \textit{for all} \quad i = 1,\ldots,n. \tag{7}$$

Proof. See [8]. □

Because the involved matrices commute, they have a common basis of eigenvectors. The variety $V(I)$, the solution set of (complex) zeros of I, contains all the n-tuples $(\lambda_1,\ldots,\lambda_n)$ of eigenvalues of $(A_{x_1},\ldots,A_{x_n})$ corresponding to all common eigenvectors.

In the case of polynomial optimization, the N eigenvalues $\lambda_i^{(1)},\ldots,\lambda_i^{(N)}$ of a matrix A_{x_i} are the values of the ith coordinates, x_i, at the stationary points of the polynomial $p_\lambda(x_1,\ldots,x_n)$. Computing *all* real eigenvalues of the matrices $A_{x_1},\ldots,A_{x_n}$ provides a way to determine *all* locations of the stationary points of p_β (solutions of (3)).

The method of rewriting the problem of finding solutions of a set of polynomial equations into an eigenvalue problem of a set of commuting matrices, is called the Stetter–Möller matrix method and can only be applied to systems of polynomial equations which generate a zero-dimensional ideal. This method is described in [21] and a similar approach can be found in [8].

More generally, for a polynomial $f(x_1,\ldots,x_n)$ a similar set-up can be used to introduce a linear operator $\mathcal{A}_{f(x_1,\ldots,x_n)}$ as follows:

$$\mathcal{A}_{f(x_1,\ldots,x_n)} \ : \ \mathbb{R}[x_1,\ldots,x_n]/I \ \to \ \mathbb{R}[x_1,\ldots,x_n]/I \ \ : \ \ g \ \mapsto \ f(x_1,\ldots,x_n)\cdot g. \tag{8}$$

The (linear) operator $\mathcal{A}_f$ can be represented by the $N \times N$ matrix $A_{f(x_1,\ldots,x_n)}$ with respect to the same monomial basis B. The following property will be key to the approach we propose.

Proposition 3. *Let f be a polynomial in $\mathbb{R}[x_1,...,x_n]$. The matrix A_f commutes with the matrices $A_{x_1},\ldots,A_{x_n}$ and satisfies $A_{f(x_1,\ldots,x_n)} = f(A_{x_1},\ldots,,A_{x_n})$.*

Proof. Commutativity of $A_{x_1}, \ldots, A_{x_n}$ makes the definition of $f(A_{x_1}, \ldots, A_{x_n})$ unambiguous. Together with linearity of the operators, the proposition follows. See [1]. □

Therefore, if $\mathbf{v}$ is a common eigenvector of the matrices $A_{x_1}, \ldots, A_{x_n}$ for the eigenvalues $(\lambda_1, \ldots, \lambda_n)$, then $\mathbf{v}$ is also an eigenvector of $A_{f(x_1,\ldots,x_n)}$ with corresponding eigenvalue $f(\lambda_1, \ldots, \lambda_n)$:

$$A_f \mathbf{v} = f(A_{x_1}, \ldots, A_{x_n})\, \mathbf{v} = f(\lambda_1, \ldots, \lambda_n)\, \mathbf{v}. \tag{9}$$

Let us recall that one can compute the ith coordinates of the stationary points of the polynomial p_β by using the eigenvalues of the matrices A_{x_i}. It follows from the work of Frobenius on commutative matrices, see [5], that all the eigenvalues of $A_{x_1}, \ldots, A_{x_n}$ can be organized, in a unique way, into n-tuples $(x_1, \ldots, x_n)$ which share a common eigenvector. These n-tuples constitute the stationary points of p_β. For every polynomial f, the eigenvalues of A_f evaluate f at the stationary points. In particular, this holds for the polynomial $f = p_\beta$:

Proposition 4. *The eigenvalues of the matrix $A_{p_\beta(x_1,\ldots,x_n)}$ are the function values of the polynomial $p_\beta(x_1, \ldots, x_n)$ at its stationary points.*

Proof. See [1]. □

This makes clear why this approach is particularly useful for global polynomial optimization: the *leftmost real eigenvalue* (i.e., with smallest real part) of the matrix $A_{p_\beta(x_1,\ldots,x_n)}$ is the value of the global minimum of $p_\beta(x_1, \ldots, x_n)$. For convenience of notation, we will write $A_{p(x_1,\ldots,x_n)}$ or A_p instead of $A_{p_\beta(x_1,\ldots,x_n)}$ from now on; we will choose $\beta = 1$ in the experiments in Section 4.

Note that we are interested in the smallest real eigenvalue of the real-valued matrix $A_{p(x_1,\ldots,x_n)}$, for which we can select a real-valued corresponding eigenvector $\mathbf{v}$. Because the real-valued matrices A_{x_i} share the same eigenvector $\mathbf{v}$, the interesting eigenvalues of the matrices A_{x_i} (the coordinates of the stationary points) have to be real as well.

The approach described in this section has some promising properties which are used in designing a more efficient optimization method: (i) because of the commutativity of the matrices A_p, A_{x_1}, …, A_{x_n} they have common eigenvectors, (ii) the matrices A_{x_1}, …, A_{x_n} are much sparser than the matrix A_p, and finally, (iii) only the smallest real eigenvalue and corresponding eigenvector of the matrix A_p are required to locate the (guaranteed) global optimum of p_β (without addressing any (possible) local optimum p_β contains). As a side effect we here also use the fact that all the involved eigenvectors are structured in a special way, called the Stetter structure (see [1]), which makes it easy to read off the values of x_i, the locations of the stationary points, from the eigenvectors (this holds for algebraic multiplicity one).

Section 3 introduces the new Jacobi–Davidson eigenvalue solver for commuting matrices, JDCOMM, which uses all these properties to improve the performance of the optimization method described in the present section. For further background of the constructive algebra and systems theory aspects of this approach we also refer to [7].

3. A Jacobi–Davidson type method for commuting matrices

Since the size of the commuting matrices mentioned in the previous section is usually very large, i.e., of dimension $N \times N$ with $N = (2d-1)^n$, and because we are interested in only one eigenvalue, we consider iterative eigensolvers. More particularly we are interested in subspace methods.

In this paper, a new Jacobi–Davidson type method is proposed for this problem. This method consists of an outer loop, where a subspace is constructed onto which the matrix is projected, and an inner loop, used for an inexact solve of the correction equation, which may be viewed as an inexact Newton step.[1] This type of eigensolver may be preferable over, for instance, Krylov methods (such as implicitly restarted Arnoldi, as described in [28] and implemented in Matlab's eigs) for three reasons.

[1] See [27, 11] and the rest of this section for more background information on the standard Jacobi–Davidson method.

First, the eigenvalue of interest may not be near the exterior of the spectrum of A_p. This implies that it may be difficult for Krylov methods to find the wanted eigenvalue(s) (see also the experiments with Matlab's eigs and an implementation of Krylov–Schur [29] in Section 4). Second, approaches based on matrix transformations (see, e.g., [18] for an overview, and, for instance, [19]) are not applicable: shift-and-invert Arnoldi methods are infeasible as these require a known shift (target); in our problems we do not have a shift beforehand. Third, and most importantly, our method is able to make effective use of the action with sparser (and hence cheaper) commuting matrices in the inner iterations: it computes the eigenvalues of the sparse matrix A_p while using one of the much sparser matrices A_{x_1}, $\ldots$, A_{x_n} in the inner loop (see Properties (i) and (ii) in Section 2), often resulting in a much more efficient process. In our approach we target the smallest (leftmost) real eigenvalue of A_p as it corresponds to the value of the global minimum of p_β (see Property (iii) in Section 2). Still, an Arnoldi type method such as eigs may work well if the sought eigenvalue of A_p is relatively exterior.

In [20, 16], shift-and-invert Cayley transforms are exploited to find the rightmost eigenvalue. We believe that although this method could be worthwhile for difficult cases with clustered eigenvalues, it will often be not competitive in comparison with Arnoldi.

Jacobi–Davidson type methods generally spend most of the computational effort in the inner loops, and since we work with sparser matrices in these inner iterations, a speedup can be expected. As we will see in the experiments in Section 4, the new Jacobi–Davidson type method for commuting matrices may impressively outperform standard Jacobi–Davidson and methods based on implicitly restarted Arnoldi (Matlab's eigs and Krylov–Schur) as well as some other conventional optimization methods for this problem.

The Jacobi–Davidson subspace expansion works as follows. Suppose we have an approximate eigenpair $(\theta, \mathbf{v})$ for A_p, where $\mathbf{v}$ has unit norm and θ is the Rayleigh quotient of A_p and $\mathbf{v}$: $\theta = \mathbf{v}^* A_p \mathbf{v}$. We now look for an update $\mathbf{t} \perp \mathbf{v}$ such that the updated vector $\mathbf{v} + \mathbf{t}$ is an eigenvector of A_p:

$$A_p(\mathbf{v} + \mathbf{t}) = \lambda\,(\mathbf{v} + \mathbf{t}). \tag{10}$$

Rearranging the terms gives:

$$(A_p - \theta I_n)\,\mathbf{t} = -(A_p\mathbf{v} - \theta\,\mathbf{v}) + (\lambda - \theta)\,\mathbf{v} + (\lambda - \theta)\,\mathbf{t}, \tag{11}$$

where

$$\mathbf{r} := A_p\mathbf{v} - \theta\,\mathbf{v} \tag{12}$$

is the residual with respect to the matrix A_p. If we discard the $(\lambda-\theta)\,\mathbf{t}$ term, which is asymptotically second-order, and project out the unknown quantity $(\lambda - \theta)\,\mathbf{v}$ by the projector $I_n - \mathbf{v}\mathbf{v}^*$, which also fixes $\mathbf{r}$, we get the Jacobi–Davidson *correction equation* for the matrix A_p:

$$(I_n - \mathbf{v}\mathbf{v}^*)(A_p - \theta I_n)\,\mathbf{t} = -(A_p\mathbf{v} - \theta\,\mathbf{v}), \qquad \mathbf{t} \perp \mathbf{v}. \tag{13}$$

The solution of (13) is sometimes also referred to as inner iterations or inner loop. A key factor of Jacobi–Davidson is that one may solve (13) inexactly; generally it is neither necessary nor attractive to solve it exactly; see also the comments in Section 4.9. Still, the vast majority of the computational work is spent by (inexactly) solving this correction equation. Therefore, the simple but crucial idea is to make use of one of the sparser matrices A_{x_i}, $i = 1, \ldots, n$ (which have the same eigenvectors as A_p) in the correction equation (13).

To be able to use the much sparser matrices $A_{x_1}, \ldots, A_{x_n}$ in the inner loop and to create a faster convergence, we propose the following: since $\mathbf{v}$ is an approximate eigenvector for A_p, it is also an approximation eigenvector for A_{x_i}. Let η be the Rayleigh quotient of A_{x_i} and $\mathbf{v}$: $\eta = \mathbf{v}^* A_{x_i}\mathbf{v}$. Instead of (10) we now wish to update the vector $\mathbf{v}$ such that we get an eigenvector for A_{x_i}:

$$A_{x_i}(\mathbf{v} + \mathbf{t}) = \mu\,(\mathbf{v} + \mathbf{t})$$

for a certain eigenvalue μ of A_{x_i} (this eigenvalue will be the ith coordinate of the global minimum). Using the approximate value η for A_{x_i}, this leads, similarly to (11), to:

$$(A_{x_i} - \eta I_n)\,\mathbf{t} = -(A_{x_i}\mathbf{v} - \eta\mathbf{v}) + (\mu - \eta)\,\mathbf{v} + (\mu - \eta)\,\mathbf{t}.$$

By again neglecting the higher-order term $(\mu - \eta)\,\mathbf{t}$ and projecting out the unknown term $(\mu - \eta)\,\mathbf{v}$ we get the Jacobi–Davidson *correction equation* for the matrix A_{x_i}:

$$(I_n - \mathbf{v}\mathbf{v}^*)(A_{x_i} - \eta I_n)\,\mathbf{t} = -(A_{x_i}\mathbf{v} - \eta\mathbf{v}), \qquad \mathbf{t} \perp \mathbf{v}. \tag{14}$$

The advantage of this correction equation over (13) is that the matrix-vector products with A_{x_i} spent for approximately solving this equation are generally much cheaper than matrix-vector multiplications with A_p. Typical practical examples indicate that the number of nonzeros of A_{x_i} is often 10% or less of the number of nonzeros of A_p; see the experiments in Section 4, where we also discuss the choice of i.

Remark 1. We note that preconditioning of the correction equation and the use of harmonic Ritz values is generally infeasible in our context as a target is necessary for these techniques; i.e., a reasonably good guess of the value of the wanted eigenvalue. This knowledge will generally be unavailable.

Remark 2. When the eigenvalue of A_p corresponding to the minimal value of the polynomial has an algebraic multiplicity larger than one, there are multiple minima with the same minimal value at different locations. Typically, the Jacobi–Davidson method may converge to a multiple eigenvalue, albeit at a slower speed of convergence. A detected eigenvector of the matrix A_p (which is not uniquely defined) will generally fail to have a Stetter structure and will not be an eigenvector of the A_{x_i} matrices. In this case, one may first separately determine the nullspace corresponding to the multiple eigenvalue of A_p, and then project this space onto the A_{x_i} matrices to obtain the eigenvectors. If the x_i coordinates of the minimizers are distinct (which will always be the case for at least one index i) we can retrieve the eigenvectors with Stetter structure and, as a result, read off the coordinates of the minimizers.

It may also happen that a matrix A_{x_i} has eigenvalues of multiplicity larger than one, which means that one coordinate of the (global or local) minimizers is identical. If for this (or another) reason the convergence of the method seems to be slow, it may be beneficial to switch to another operator A_{x_j} at no additional costs; see also Section 4.8.

For ease of presentation, we assume in the rest of the paper that the algebraic multiplicity of all the eigenvalues involved is one. However, we briefly mention that in an unreported experiment ($p(x_1, x_2, x_3, x_4, x_5) = (x_1^6 + x_2^6 + x_3^6 + x_4^6 + x_5^6) - 7x_1^3x_4x_5 + 8x_1^2x_2x_3^2 + 5x_2x_3x_4^2x_5 - x_2x_3^4 - 4x_3^2x_4x_5$, matrix size 3125×3125), JDCOMM is able to handle the case of numerically multiple eigenvalues well (JDCOMM finds the correct minimizer, and is 5 times faster than JD).

In the following algorithm we give the pseudocode for the JDCOMM method, the Jacobi–Davidson type method for commuting matrices. Note that in the subspace extraction phase of every (outer) iteration (line 5 of the algorithm) we need to work with A_p, since we should head for the smallest real eigenvalue of A_p.

Algorithm JDCOMM: A Jacobi–Davidson type method for commuting matrices

---

Input: A device to compute $A_p\mathbf{v}$ and $A_{x_i}\mathbf{v}$ for arbitrary vectors $\mathbf{v}$, where the action with A_{x_i} is (much) cheaper than the action with A_p; a starting vector $\mathbf{v}_1$ (default: random) and a tolerance ε

Output: An approximate eigenpair $(\theta, \mathbf{v})$ of A_p

1: $\mathbf{t} = \mathbf{v}_1,\ V_0 = [\,]$
   for $k = 1, 2, \ldots$
2: rgs$(V_{k-1}, \mathbf{t}) \to V_k$
3: Compute kth column of $W_k = A_p V_k$
4: Compute kth row and column of $H_k = V_k^* A_p V_k = V_k^* W_k$
5: Extract the leftmost real Ritz pair $(\theta, \mathbf{c})$, an eigenpair of H_k
6: $\mathbf{v} = V_k\mathbf{c}$
7: $\mathbf{r} = W_k\mathbf{c} - \theta\,\mathbf{v}$
8: Stop if $\|\mathbf{r}\| \le \varepsilon$
9: Compute $\eta = \mathbf{v}^* A_{x_i}\mathbf{v}$
10: Solve (approximately) $\mathbf{t} \perp \mathbf{v}$ from:
$(I_n - \mathbf{v}\mathbf{v}^*)(A_{x_i} - \eta I_n)\,\mathbf{t} = -(A_{x_i} - \eta I_n)\,\mathbf{v}$

A few comments on this algorithm are in order. The starting vector $\mathbf{v}$, which is one of the input arguments for the method, is by default chosen as a random vector. Choosing random values for the starting vector is a common choice

when computing eigenvalues iteratively. In line 2, rgs stands for repeated Gram–Schmidt, or any other numerically stable orthonormalization method. In line 5, we select the leftmost real eigenvalue θ of H_k. If it happens that H_k only contains complex eigenvalues, we select one eigenvalue of the complex conjugate pair with the smallest real part as the next best alternative. Not shown in the pseudocode, but generally necessary in practice, are restarts. If the dimension of the search space reaches a specified constant maxdim in line 5, we reduce the search space to dimension mindim, continuing with the space spanned by the best mindim Ritz vectors, that is, the Ritz vectors corresponding to the most promising Ritz values. In our context, this means that we order the Ritz values θ with respect to increasing real part, first those with imaginary part satisfying $|\mathrm{Im}(\theta)| <$ threshold, for some threshold value, and subsequently the remaining ordered Ritz values. The correction equations in line 10 are solved approximately with an iterative method. In our experiments in Section 4, we take 10 steps of GMRES.

Note that, since we are interested in real eigenvalues only, it could be an option to modify the algorithm such that we obtain the solutions using only real arithmetic. This is the subject of [23] for a conventional JD method but this is outside the scope of the present paper.

Another important property of this polynomial optimization approach is that it can be used in a matrix-free fashion, which means that in the method the matrices $A_p, A_{x_1}, \ldots, A_{x_n}$ are not needed in explicit form. It is sufficient to have a routine that performs the matrix-vector products $A_p\mathbf{v}$ and $A_{x_i}\mathbf{v}$ (for at least one i) for arbitrary vectors $\mathbf{v}$. This property is not used in the present paper but it may make the method even more time efficient as well as more memory efficient. The matrix-free matrix-vector multiplication can be carried out by associating the system of first-order derivatives of p_β with an nD-system of difference equations, as described extensively in [1].

Our method inherits the favorable convergence properties of standard Jacobi–Davidson, as we show in the following propositions.

Proposition 5. *Suppose* $\mathbf{z}$ *is an eigenvector corresponding to a simple eigenvalue* λ *of* A_p *and a simple eigenvalue* μ *of* A_{x_i} *(for some i). If the correction equations (14) are solved exactly and JDCOMM converges to* $(\lambda, \mathbf{z})$*, the method converges asymptotically quadratically.*

Proof. The proof is inspired by, e.g., [12]. Suppose $\mathbf{v} = \mathbf{z} + \varepsilon\,\mathbf{e}$, where $\mathbf{e} \perp \mathbf{z}$, $\|\mathbf{z}\| = 1$, and $\|\mathbf{e}\| = 1$, where $\|\cdot\|$ denotes the two-norm. This means that $\tan(\mathbf{v}, \mathbf{z}) = \varepsilon$. It can be readily verified that θ, the Rayleigh quotient of $\mathbf{v}$ with respect to A_p, satisfies $|\lambda - \theta| = O(\varepsilon)$. Similarly, η, the Rayleigh quotient of $\mathbf{v}$ with respect to A_{x_i}, satisfies $|\mu - \eta| = O(\varepsilon)$.

If we solve (14) exactly, we get:

$$\mathbf{v} + \mathbf{t} = (\mathbf{v}^*(A - \eta I)^{-1}\mathbf{v})^{-1}\,(\mu - \eta)^{-1}(\mathbf{z} + \varepsilon\,(\mu - \eta)(A - \eta I)^{-1}\mathbf{e}).$$

Since by assumption $(A - \eta I)^{-1}$ is bounded on $\mathbf{z}^\perp$, for the updated vector $\mathbf{v} + \mathbf{t}$ we have:

$$\tan(\mathbf{v} + \mathbf{t}, \mathbf{z}) = O(\varepsilon^2),$$

which shows the quadratic convergence. □

However, in practice we often solve the correction equations inexactly, which leads us to the next result. Introduce the notation

$$\widetilde{\mathbf{r}} := A_{x_i}\mathbf{v} - \eta\mathbf{v}$$

for the residual associated with A_{x_i} (cf. (12) and (14)).

Proposition 6. *We make the same assumptions as in Proposition 5, except that we now solve the correction equations (14) inexactly with a norm reduction:*

$$\|(I_n - \mathbf{v}\mathbf{v}^*)(A_{x_i} - \eta I_n)\widetilde{\mathbf{t}} + \widetilde{\mathbf{r}}\| \le \xi\,\|\widetilde{\mathbf{r}}\|, \qquad \widetilde{\mathbf{t}} \perp \mathbf{v}.$$

If JDCOMM converges to $(\lambda, \mathbf{z})$*, the method converges asymptotically linearly, provided that:*

$$\xi < (\kappa((A - \eta I)|_{\mathbf{z}^\perp}))^{-1},$$

where κ *denotes the condition number of a matrix, and* $(A - \eta I)|_{\mathbf{z}^\perp}$ *indicates the restriction of the operator as a map from* $\mathbf{z}^\perp$ *onto itself.*

Proof. The assumptions imply that there exists a $\widetilde{\xi} \leq \xi$ and $\mathbf{f} \perp \mathbf{v}$, $\|\mathbf{f}\| = 1$, with:

$$(I_n - \mathbf{v}\mathbf{v}^*)(A_{x_i} - \eta I_n)\,\mathbf{t} = -\widetilde{\mathbf{r}} + \widetilde{\xi}\,\|\widetilde{\mathbf{r}}\|\,\mathbf{f}, \qquad \mathbf{t} \perp \mathbf{v}.$$

As in [12, Th. 5.3], it follows that:

$$\|\widetilde{\mathbf{t}} - \mathbf{t}\| = \widetilde{\xi}\,\kappa((A - \eta I)|_{\mathbf{z}^\perp})\,\|\mathbf{t}\| + O(\|\mathbf{t}\|^2),$$

where $\mathbf{t}$ is the exact solution of (14). This yields an asymptotically linear convergence. □

4. Numerical experiments

4.1. Brief overview

All results in this section are obtained with Matlab version R2007B running on an AMD Dual-Core Opteron (F2216, 2.4 GHz) machine with 8GB of internal memory. For the numerical experiments in this section we test an implementation of the new JDCOMM method described in this paper. We compare our method with an implementation of a conventional Jacobi–Davidson method, denoted by the JD method and described in [11]. Moreover, we add timings of Matlab's eigs, based on implicitly restarted Arnoldi [28], with the option "SR" (smallest real part), until the first real eigenvalue is detected. Also, we compare with another method based on an implicitly restarted Arnoldi technique: an implementation of the Krylov–Schur [29] method, where in every iteration we select the smallest real Ritz value.[2]

4.2. Experiment 1

For the first experiment a polynomial p_β with $n = 4$, $d = 4$, and $\beta = 1$ is considered:

$$\begin{aligned} p_1(x_1, x_2, x_3, x_4) = (x_1^8 + x_2^8 + x_3^8 + x_4^8) + 2.4x_1^4x_3^2 + 0.5x_1^2x_2x_4 \\ + 9.4x_1x_2^4x_3 + 9.7x_1x_2^3x_4^2 - 1.6x_2^3x_4^3 - 10.4x_3x_4^2. \end{aligned} \tag{15}$$

Using the involved quotient space $\mathbb{R}[x_1, x_2, x_3, x_4]/I$ of dimension $N = (2d-1)^n = 2401$, the matrices $A_p, A_{x_1}, A_{x_2}, A_{x_3}$, and A_{x_4} are constructed explicitly. We are interested in the smallest real eigenvalue and corresponding eigenvector of the matrix A_p. Table 1 shows the differences in the number of nonzero elements of all matrices involved. The matrices A_{x_i}, $i = 1, \ldots, 4$, are much sparser than the matrix A_p. See also Figure 1 for a representation of the sparsity structure of the matrices.

Table 1: Sparsity of the 2401×2401 matrices A_p and $A_{x_1}, A_{x_2}, A_{x_3}, A_{x_4}$: number of nonzeros (nnz), and relative filling (Experiment 1).

Matrix	**nnz**	**% filled**
A_p	182604	3.17
A_{x_1}	9571	0.17
A_{x_2}	8196	0.14
A_{x_3}	9783	0.17
A_{x_4}	8028	0.14

[2] We sort the Ritz values on increasing real part, and then select the first Ritz value θ with $|\mathrm{Im}(\theta)| \leq 10^{-12}$. If there is none, we select the first other Ritz value.

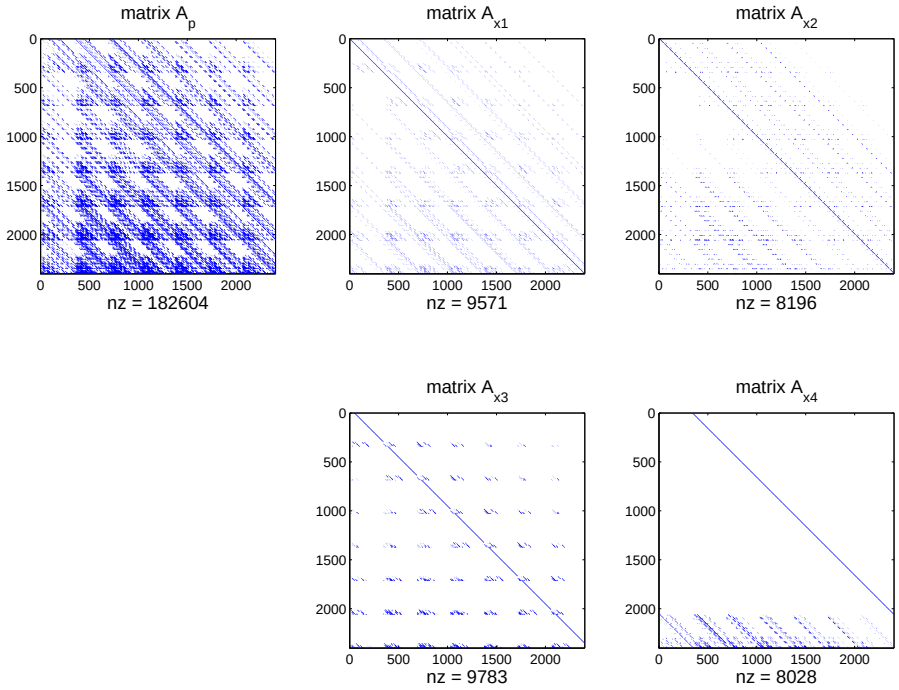

Figure 1: Sparsity structure of the matrices A_p and A_{x_i}, $i = 1, \ldots, 4$ (Experiment 1).

In this case, we can compute *all* the eigenvalues of the matrix A_p using a direct solver and select the smallest real one as being the global optimum of polynomial $p_1(x_1, x_2, x_3, x_4)$. Alternatively, we can compute *all* the eigenvalues of the matrices A_{x_1}, A_{x_2}, A_{x_3}, and A_{x_4} and read off the coordinates of the stationary points from the corresponding eigenvectors. The global optimum of $p_1(x_1, x_2, x_3, x_4)$ can then be selected from all the stationary points by computing the function values of them and by picking out the smallest real one. Computing all the eigenvalues (using Matlab's eig method) of the matrices A_p, A_{x_1}, A_{x_2}, A_{x_3}, and A_{x_4} using a direct solver takes 105, 29, 25, 39, and 36 seconds respectively. The global minimizer we are looking for has value ≈ -616.8 and is attained at the point $x_1 \approx -1.956$, $x_2 \approx 2.380$, $x_3 \approx 1.810$, and $x_4 \approx 2.098$.

A more efficient way of computing the global optimum is by using a Jacobi–Davidson type method to compute only the smallest real eigenvalue of the matrix A_p as described in this paper. Table 2 shows the results of a conventional JD method, applied to the matrix A_p, and the new JDCOMM method, which also works on A_p but exploits one of the sparser matrices A_{x_i} to solve the correction equation. However, note that we may also use the JDCOMM method in a matrix-free fashion. The settings used for both the JD and JDCOMM methods are as follows: the tolerance for the residual norm for the convergence of the eigenpair is 10^{-6} and the minimal and maximal dimensions of the search spaces, mindim and maxdim, respectively, are 30 and 75. All correction equations are solved with 10 steps of GMRES.

The second and third columns of Table 2 contain the number of matrix vector products (denoted by MV) of the method with the matrix A_p and the matrices A_{x_i}, respectively. Thus, these columns represent the number of iterations of the outer loop and inner loop to solve the correction equation with one of the matrices A_{x_i} in the algorithm. The fourth column shows the number of floating point operations (flops) needed to perform all matrix-vector products. The number of flops is defined here as the sum of the products of the number of multiplications with the number of nonzeros of the corresponding matrices. For instance, the JDCOMM method that iterates with the matrix A_{x_1} in the inner loop and with the matrix A_p in the outer loop requires 130 iterations with the matrix A_p and 680 iterations with

Table 2: Comparison between performance of JD and JDCOMM (Experiment 1).

Method	**MV** A_p	**MV** A_{x_i}	**Flops** $\cdot 10^8$	**Time (s)**
JD	823	0	1.50	6.0
JDCOMM A_{x_1}	130	680	0.30	2.6
JDCOMM A_{x_2}	130	680	0.29	2.6
JDCOMM A_{x_3}	163	943	0.39	3.8
JDCOMM A_{x_4}	128	658	0.29	2.5

the matrix A_{x_1}. This takes $130 \cdot 182604 + 680 \cdot 9571 \approx 0.30 \cdot 10^8$ flops for the matrix-vector products. Finally, column five shows the total computation time required.

We mention that all methods successfully compute the correct leftmost real eigenvalue ≈ -616.8, corresponding to the global minimum of the polynomial. The performance of the JDCOMM method for any i is better than that of the JD method, both in terms of computation time and in required floating point operations.

Figure 2 shows the plot of the norms of the residuals in the eigenvalue equation (at each JD outer step) against the number of matrix vector products with A_p, which represents the main part of the computational effort. Note that JDCOMM also carries out matrix-vector products with the A_{x_i} matrices, but these are much cheaper.

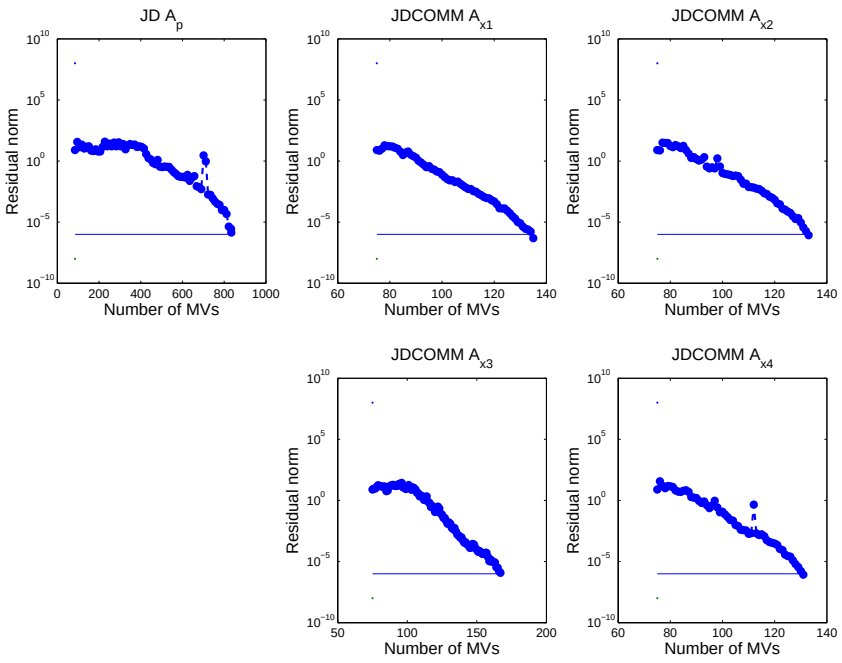

Figure 2: Residual norms against matrix vector products with A_p for the JD and JDCOMM methods (Experiment 1).

To reveal the performance of a Krylov based eigenvalue solver on this example, we also apply the Matlab eigs method, which is an implicitly restarted Arnoldi method [28], to this problem. This method computes the requested smallest real eigenvalue in 7.4 seconds as the 43rd eigenvalue. The Krylov–Schur method [29] applied to A_p, with the same minimum and maximum subspace dimensions, computes the desired eigenvalue as 11th in 2.3 seconds. We note that this is the only example where Krylov–Schur is faster than JDCOMM.

To put the performance of the JDCOMM approach of this paper into perspective, we will briefly discuss the results of the computation of the global minimum of polynomial (15) by the software packages SOSTOOLS (v2.0), Glop-

tiPoly (v3.0), and PHCpack (v2.3.66) which employ totally different approaches. SOSTOOLS is a Matlab toolbox for formulating and solving sum of squares (SOS) problems [24, 25]. This toolbox uses the Matlab solver SeDuMi [30] to solve the involved semi-definite programs (SDP). GloptiPoly [10] solves a multi variable polynomial optimization problem by building and solving convex linear matrix inequality (LMI) relaxations of the problem, also using SeDuMi. It produces a series of lower bounds which converge to the global optimum of interest [14, 15]. PHCpack [32] uses polynomial homotopy continuation methods to numerically solve systems of polynomial equations. Note here that PHCpack is able to compute all the solutions of the system of equations. We have here used the PHCpack interface to call its functionality from within Matlab [6]. Using default parameter settings, SOSTOOLS, GloptiPoly, and PHCpack successfully compute the same global optimum, in 6.6, 5.9, and 32.9 seconds, respectively.

4.3. Experiment 2

Let us consider a Minkowski-norm dominated polynomial in 5 variables ($n = 5$), a maximal total degree of 6 ($d = 3$) and $\beta = 1$:

$$\begin{aligned} p_1(x_1, x_2, x_3, x_4, x_5) = (x_1^6 + x_2^6 + x_3^6 + x_4^6 + x_5^6) + 18x_1^3x_2x_5 + 13.1x_1^2x_2 \\ - 5.3x_1x_2x_3^3 - 1.2x_1x_2x_4^2x_5 - 5.9x_1x_2x_4 - 8.3x_1x_3x_4^2 \\ - 16.9x_1x_4x_5 - 8.2x_2^3x_3x_5 - 10.1x_2x_4^3x_5 - 1.5x_3^2x_4x_5. \end{aligned} \tag{16}$$

The corresponding quotient space $\mathbb{R}[x_1, x_2, x_3, x_4, x_5]/I$ has dimension $N = (2d - 1)^n = 3125$ and using this, one can construct the matrices A_p, A_{x_1}, A_{x_2}, A_{x_3}, A_{x_4}, and A_{x_5} of dimensions 3125×3125. Table 3 shows the differences in the number of nonzero elements of all the involved matrices.

Table 3: Sparsity of the 3125×3125 matrices A_p and $A_{x_1}, \ldots, A_{x_5}$: number of nonzeros (nnz), and relative filling (Experiment 2).

Matrix	**nnz**	**% filled**
A_p	2566878	26.28
A_{x_1}	163384	1.67
A_{x_2}	168203	1.72
A_{x_3}	165275	1.69
A_{x_4}	155427	1.59
A_{x_5}	166238	1.70

Here the outer tolerance is chosen as 10^{-8} and the minimal and maximal dimensions of the search spaces as mindim=40 and maxdim=90. The detected eigenvalue $\approx -1.4266 \cdot 10^5$ is the smallest real eigenvalue of the matrix A_p, which corresponds to the global minimum of the polynomial. The coordinates of this global minimizer are $x_1 \approx -7.991$, $x_2 \approx -7.633$, $x_3 \approx 6.344$, $x_4 \approx 6.094$, and $x_5 \approx -7.102$.

The times and the number of matrix vector products needed by the JD and JDCOMM methods for the computation of the leftmost real eigenvalue are summarized in Table 4.

Table 4: Comparison between performance of JD and JDCOMM (Experiment 2).

Method	**MV** A_p	**MV** A_{x_i}	**Flops** $\cdot 10^9$	**Time (s)**
JD	5974	0	15.33	237.0
JDCOMM A_{x_1}	177	947	0.61	13.9
JDCOMM A_{x_2}	194	1134	0.69	16.5
JDCOMM A_{x_3}	167	837	0.57	12.7
JDCOMM A_{x_4}	215	1299	0.75	18.9
JDCOMM A_{x_5}	193	1123	0.68	16.4

Also in this example, the JDCOMM methods with the matrices A_{x_i} again need less computation time. Matlab's eigs did not successfully compute the same smallest real eigenvalue of the matrix A_p. Krylov–Schur uses much more time: 4623 seconds, but also needs larger subspace dimensions (minimum 51, maximum 101, as the eigenvalue is detected as 51st) for convergence. To compute the same global minimizer including its location, SOSTOOLS and

PHCpack need 5.3 and 100 seconds. GloptiPoly indicates after 5.7 seconds that it can not accurately solve this problem and gives a global minimum close to the wanted one.

4.4. Experiment 3

In the third experiment we consider a polynomial p_β with $n = 4$, $d = 5$, and $\beta = 1$:

$$p_1(x_1, x_2, x_3, x_4) = (x_1^{10} + x_2^{10} + x_3^{10} + x_4^{10}) - 11.2x_1^7x_3 - 1.3x_1^2x_2^4x_4 + 6.7x_1^2x_2x_3 + 12x_1^2x_3x_4^5 \\ + 13.4x_1x_2^5 - 8.8x_1x_2^4x_4 - 3.4x_1x_2^2x_3^6 - 5.1x_1x_2x_4^5 - 1.5x_2^3x_3^4. \tag{17}$$

The quotient space $\mathbb{R}[x_1, x_2, x_3, x_4]/I$ has dimension $N = (2d-1)^n = 6561$, which yields matrices A_p, A_{x_1}, A_{x_2}, A_{x_3}, and A_{x_4} of dimensions 6561×6561. Table 5 shows the differences in the number of nonzero elements of all the involved matrices; see also Figure 3.

Table 5: Sparsity of the 6561×6561 matrices A_p and $A_{x_1}, \ldots, A_{x_4}$: number of nonzeros (nnz), and relative filling (Experiment 3).

Matrix	**nnz**	**% filled**
A_p	4056866	9.42
A_{x_1}	96171	0.22
A_{x_2}	93643	0.22
A_{x_3}	101322	0.24
A_{x_4}	123148	0.29

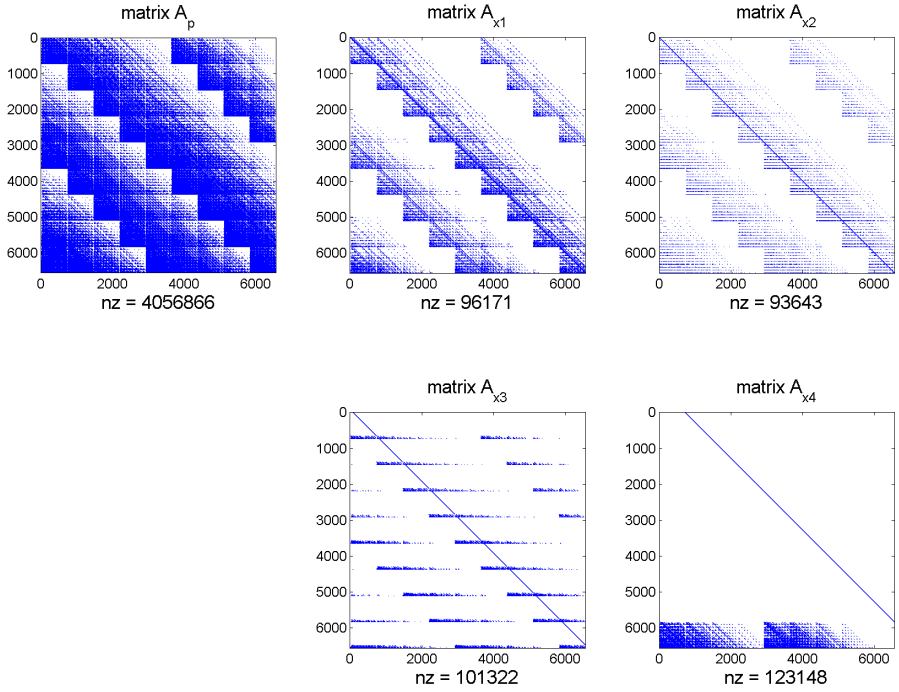

Figure 3: Sparsity structure of the matrices A_p and A_{x_i}, $i = 1, \ldots, 4$ (Experiment 3).

Computing all the 6561 eigenvalues of the matrices using the direct method Eig takes 1286, 602, 692, 645, and 628 seconds for the matrices, A_p and $A_{x_1}, \ldots, A_{x_4}$, respectively. The results of computing the smallest real eigenvalue of

the matrix A_p using the JD and JDCOMM methods, are displayed in Table 6. The parameters for the JD methods are chosen here as 10^{-8} for the tolerance on the residual norm and mindim=50 and maxdim=90 for the restart parameters of the minimal and maximal dimensions of the search spaces. Again, all the methods compute the same global

Table 6: Comparison between performance of JD and JDCOMM (Experiment 3).

Method	**MV** A_p	**MV** A_{x_i}	**Flops** $\cdot 10^8$	**Time (s)**
JD	1299	0	52.70	107.0
JDCOMM A_{x_1}	174	914	7.94	22.8
JDCOMM A_{x_2}	174	914	7.91	22.5
JDCOMM A_{x_3}	148	628	6.64	17.6
JDCOMM A_{x_4}	185	1035	8.78	25.6

optimum of $\approx -2.959 \cdot 10^4$ and this global optimum is attained at the stationary point with coordinates $x_1 \approx 3.029$, $x_2 \approx -2.388$, $x_3 \approx 2.792$, and $x_4 \approx -2.793$. In this case, the JDCOMM methods again perform better than the conventional JD method in terms of required flops and computation time. Computing the same smallest real eigenvalue using an Arnoldi based method yields the following: assuming the prior knowledge that one should require eigs to compute the 16th eigenvalue, it needs 42 seconds to compute this eigenvalue. Krylov–Schur on A_p uses 30 seconds.

SOSTOOLS and PHCpack compute the same global optimum and same location in 49.0 and in, as much as, 9967.2 seconds. GloptiPoly indicates after 37.1 seconds that global optimality can not be assured (status=0) and returns an inaccurate approximation of the sought eigenvalue.

4.5. Experiment 4

In this experiment we study an optimization problem of the same dimensions as the previous one: $n = 4$, $d = 5$, and $\beta = 1$, with the following Minkowski-dominated polynomial:

$$\begin{aligned} p_1(x_1, x_2, x_3, x_4) = (x_1^{10} + x_2^{10} + x_3^{10} + x_4^{10}) + 3x_1^5 x_2 x_3 x_4 + x_1^5 x_3 x_4^2 \\ + 7x_1^5 x_3 x_4 - 5x_1^3 x_3^2 x_4 - 7x_1^2 x_2 x_3^2 x_4^2 - 5x_1 x_3^3 x_4^3 - 5. \end{aligned} \tag{18}$$

The involved quotient space and thus also the matrices A_p, A_{x_1}, A_{x_2}, A_{x_3}, and A_{x_4} are of dimension 6561×6561. Table 7 shows the differences in the number of nonzero elements between the matrices.

Table 7: Sparsity of the 6561×6561 matrices A_p and $A_{x_1}, A_{x_2}, A_{x_3}, A_{x_4}$: number of nonzeros (nnz), and relative filling (Experiment 4).

Matrix	**nnz**	**% filled**
A_p	1255231	2.92
A_{x_1}	45717	0.11
A_{x_2}	59756	0.14
A_{x_3}	41129	0.10
A_{x_4}	41712	0.10

The global minimizer, the smallest real eigenvalue of the matrix A_p we are looking for, has value ≈ -206.5 and is located at the point with coordinates $x_1 \approx 1.79$, $x_2 \approx 1.43$, $x_3 \approx -1.54$, and $x_4 \approx 1.54$, which are the corresponding eigenvalues of the matrices $A_{x_1}, A_{x_2}, A_{x_3}$, and A_{x_4}. The location of these eigenvalues with respect to the entire eigenvalue spectrum of each matrix is depicted with big red dots in Figure 4.

The smallest real eigenvalue of the matrix A_p is not located very close to the exterior of the spectrum: there are 34 complex valued eigenvalues with a real part smaller than -206.5; see the top-left subfigure of Figure 4.

The smallest real eigenvalue is computed with the JD and JDCOMM method using various search space parameters. The minimal (mindim) and maximal (maxdim) dimensions of the search space are varied between 5 and 100. See Table 8 for the results. Blanks in this table indicate that the corresponding method with these settings is unable to converge to the requested eigenvalue using a maximum of 1000 outer iterations. Note here that the standard JD method

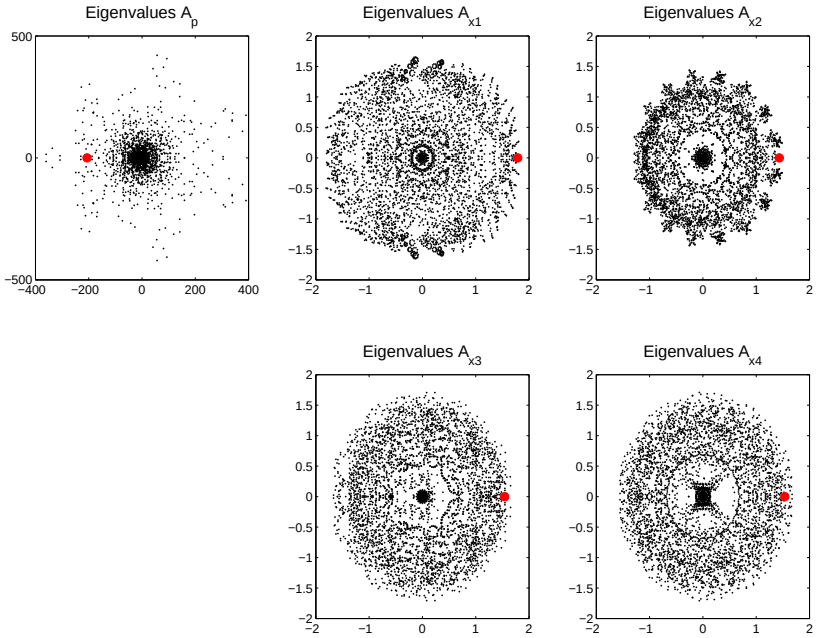

Figure 4: Eigenvalue spectra of the matrices A_p and A_{x_i}, $i = 1, \ldots, 4$ (Experiment 4).

on the matrix A_p computes the requested eigenvalue only in the case when the maximum search space dimension maxdim is 100, whereas the JDCOMM method using the matrix A_{x_2} works in all cases with a superior performance. Note also that the increase of maxdim from 25 to 50 more than halves the computation time of the JDCOMM method on the matrix A_{x_2}.

Table 8 also shows the difficulty of choosing the optimal search space parameters for the JD and JDCOMM method. We remark that determining the best parameter settings for a certain problem is still very much an open problem.

Table 8: Comparison between performance of JD and JDCOMM using various search space settings (Experiment 4).

		Time (s)				
mindim	**maxdim**	**JD**	**JDCOMM** A_{x_1}	**JDCOMM** A_{x_2}	**JDCOMM** A_{x_3}	**JDCOMM** A_{x_4}
5	30			11.6		
10	15			10.0		
10	25			15.4		
10	50		10.7	6.1		
10	75			6.8	41.5	
10	100	140	10.1	7.7	20.8	20.7

In Table 9 we give the more detailed results (with respect to required matrix vector products and flops) of the situation where all the eigenvalue methods succeed to compute the requested eigenvalue; mindim is 10 and maxdim is 100 (corresponds to last row of Table 8).

Figure 5 shows the plot of the norms of the residuals in the eigenvalue equation (at each JD outer step) against the number of matrix vector products with A_p. All methods compute the same global minimum and the identical location.

We also apply the Matlab eigs method to this problem: it computes the requested smallest real eigenvalue in 58 seconds as the 151st eigenvalue. Krylov–Schur uses much more time: 258 seconds, but also needs larger subspace dimensions (minimum 16) for convergence.

Table 9: Details of the performance of JD and JDCOMM with mindim = 10 and maxdim = 100 (Experiment 4).

Method	**MV** A_p	**MV** A_{x_i}	**Flops** $\cdot 10^9$	**Time (s)**
JD	5941	0	7.5	140
JDCOMM A_{x_1}	237	1607	0.4	10.1
JDCOMM A_{x_2}	302	2322	0.5	7.6
JDCOMM A_{x_3}	725	6975	1.2	20.8
JDCOMM A_{x_4}	636	5996	1.1	20.7

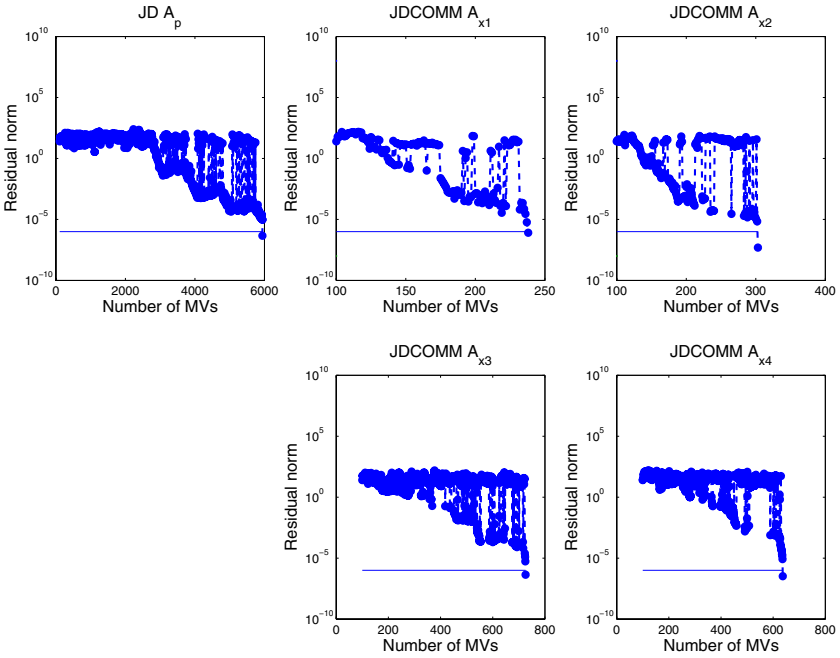

Figure 5: Residual norms against matrix vector products with A_p for the JD and JDCOMM methods (Experiment 4).

Comparing the performance of the JDCOMM method in this example with SOSTOOLS, GloptiPoly, and PHCpack, we find that these methods compute the same minimizer and require 42, 39, and 141 seconds, respectively.

4.6. Experiment 5

As a second-last experiment, we briefly mention the following problem. Let

$$\begin{aligned} p_1(x_1,x_2,x_3,x_4,x_5) = (&x_1^6 + x_2^6 + x_3^6 + x_4^6 + x_5^6) + 5x_1^2x_2x_4x_5 \\ &- 7x_1^2x_3^2 - 6x_1^2x_4^3 + 14x_1^2x_4^2 + 4x_1x_2^3 \\ &+ 9x_1x_2^2 - 4x_1x_2x_5 + 2x_1x_3^3x_4 - 3x_2^2x_5^2 \\ &- x_2^2x_3^3 + 3x_2x_4^2 + x_3^2x_5^2 + 8x_3x_4x_5^3 + 7x_4^4 - 5, \end{aligned} \tag{19}$$

with global minimum ≈ -2063.72 assumed in $x_1 \approx -3.56$, $x_2 \approx 3.24$, $x_3 \approx 3.59$, $x_4 \approx 3.50$, and $x_5 \approx -3.92$. In this case, there are 80 (complex) eigenvalues left of the leftmost real eigenvalue of A_p. While, Matlab's eigs takes 28 seconds to compute this eigenvalue (as the 156th), JDCOMM takes 4.4 seconds with mindim=40, maxdim=50 for 81 MVs with A_p (3274924 nonzeros) and 881 MVs with A_{x_4} (188083 nonzeros). Here the JD method is unable to compute the interior eigenvalue -2063.72 of the matrix A_{p_1} within 1000 iterations. The fact that the eigenvalue 3.50

of the matrix A_{x_4} lies in the exterior of the corresponding eigenvalue spectrum explains the superior performance of the JDCOMM method. Krylov–Schur uses 1237 seconds, also needing larger subspace dimensions (min 86, since the value is detected as 86th eigenvalue) for convergence. The methods SOSTOOLS, GloptiPoly and PHCpack require 3.4, 5.7 and 69 seconds to come up with the same global minimizer.

4.7. Experiment 6

As a last experiment, we briefly study the following problem. Let

$$\begin{aligned} p_1(x_1,x_2,x_3) = (x_1^{22} + x_2^{22} + x_3^{22}) - 10.4x_1^8x_2x_3^8 - 0.1x_1^7x_2^2x_3^7 \\ - 14.4x_1^4x_2^2x_3^4 - 11.9x_1^4x_2x_3^6 + 0.1x_1^2x_2^9 + 15.7x_1^2x_2^2x_3^9 \\ - 5.2x_1^2x_2^2x_3 - 12.6x_1^2x_2^{11}x_3^5 + 12.3x_2^9x_3^6 - 9.9x_3^{18} \end{aligned} \tag{20}$$

with global minimum ≈ -75234.02 assumed in $x_1 \approx 1.6159$, $x_2 \approx 1.6416$, and $x_3 \approx 1.7629$. In this case, there are 42 eigenvalues left of the leftmost real eigenvalue of A_p. The dimensions of the matrices A_p, A_{x_1}, A_{x_2}, and A_{x_3} are 9261×9261. The matrix A_p is filled for 6.43% with non-zero elements and the other matrices for 0.063%, 0.074%, and 0.047%. Computing all the eigenvalues of the matrix A_p using a direct method takes 3864 seconds. The locations of the wanted eigenvalues with respect to the entire eigenvalue spectrum of the matrices A_p and $A_{x_1}, \ldots, A_{x_3}$ are depicted with red dots in Figure 6.

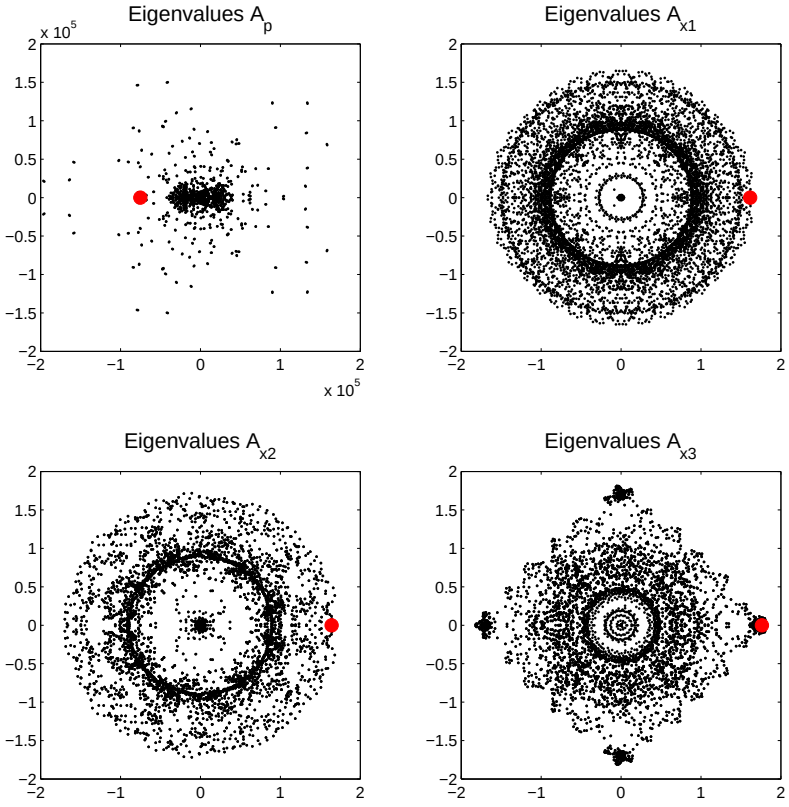

Figure 6: Eigenvalue spectra of the matrices A_p and A_{x_i}, $i = 1, \ldots, 3$ (Experiment 6).

While, Matlab's eigs takes 255 seconds, JD takes 1256 seconds to compute this minimum. However, the JDCOMM methods need only 276, 399, and 184 seconds using the sparser matrices A_{x_1}, A_{x_2}, and A_{x_3} with mindim=50,

maxdim=100; all within 2500 iterations. The total numbers of flops used by the Jacobi–Davidson methods JD and JDCOMM is $4.23 \cdot 10^{10}$, $7.99 \cdot 10^{9}$, $8.61 \cdot 10^{9}$, and $5.53 \cdot 10^{9}$. Krylov–Schur needs 256 seconds. SOSTOOLS and GloptiPoly both indicate numerical problems after 909 and 880 seconds. PHCpack computes the solution given above in 255 seconds.

Finally, we present a concise overview of all results in Table 10.

Table 10: Summary of timings of all experiments in seconds. For JDCOMM, the best of the A_{x_i} is indicated. An asterisk indicates insufficient accuracy.

Method	**Exp. 1**	**Exp. 2**	**Exp. 3**	**Exp. 4**	**Exp. 5**	**Exp. 6**
eig	105	128	1286	1005	138	3864
eigs	7.4	—	42	58	28	255
JD	6.0	237	107	140	—	1256
JDCOMM	2.5	12.7	**18**	**8**	4.4	**184**
Krylov–Schur	**2.3**	4623	30	258	1237	256
SOSTOOLS	6.6	**5.3**	49	42	**3.4**	909
GloptiPoly	5.9	100	37*	39	5.7*	880
PHCpack	32.9	5.7	9967	141	69	255

4.8. The choice of the matrix A_{x_i}

In principle, we may take any A_{x_i} in the inexact solution of the correction equation (14), since these matrices all commute with A_p. A natural thought is to select the A_{x_i} with the fewest nonzeros, which is also our general recommendation in the absence of other information. However, in practice, the success of the choice may mainly depend on how easy or difficult it is to find the correct eigenvalue of A_{x_i}. This partly depends on whether or not the wanted eigenvalue of the matrix A_p is in the interior of the spectrum or more towards the exterior, and whether the eigenvalue is a part of a cluster of eigenvalues. These properties will often be hard to predict beforehand.

In difficult cases, it may be hard to find interior eigenvalues without the use of a preconditioner. Therefore, if an approximate value for a coordinate x_i of the global minimum is known, this might be very beneficial for the process, as we may then use a preconditioner based on this target for (14) (see, e.g., [26] for an overview of some preconditioning techniques). In this case, it is natural to use the corresponding A_{x_i} in the inner iterations.

Moreover, we may also select the matrix A_{x_i} adaptively: starting with the one having the fewest nonzeros, we may switch to another matrix A_{x_j} during the calculations if the process stagnates for a longer period.

4.9. Exact versus inexact solutions of the correction equation

A key feature of JD methods is to use inexact solves of the correction equations (13) or (14) to speed up convergence to the required eigenvalue. The general assumption and practice in this field is that an exact solve is neither necessary nor attractive for matrices of these dimensions because there is no guarantee that the overall convergence will be faster, with some exceptions, such as the situation with very sparse matrices.

Although we expect that a direct solve of (14) with JDCOMM is not a competitive approach, we have performed the experiments with the exact approach and the results are as follows: using exact solves JDCOMM computes the eigenvalue of Experiments 1–4 in 279, 188, 904, 17073 seconds, respectively. For Experiment 5 it is not possible to compute an eigenvalue in this fashion within 22.000 seconds/2500 iterations. Experiment 6 takes 51232 seconds. Recall that the fastest computation times for all the experiments with inexact solves with JDCOMM are: 2.5, 12.7, 17.6, 7.6, 4.4, and 184 seconds.

5. Discussion and conclusions

We have designed a new and efficient Jacobi–Davidson eigenvalue solver to use in combination with a method to compute the global optimum of multivariate Minkowski-norm dominated polynomials $p_\beta(x_1, \ldots, x_n)$. To compute this global minimum we use the Stetter–Möller matrix approach, which reformulates the problem as an eigenvalue problem and which yields a commuting set of nonsymmetric real-valued matrices $A_{x_1}, \ldots, A_{x_n}, A_p$. Using the fact

that the matrices A_{x_i}, associated with monomial multiplication by x_i, commute with the target matrix A_p, we have developed a Jacobi–Davidson type method, denoted by JDCOMM. The JDCOMM method is able to exploit the generally much sparser matrices A_{x_i} in the inner iterations, instead of the matrix A_p, which greatly increases its efficiency. In this set-up, this method is able to target the smallest real-valued eigenvalue, which corresponds to the value of the global optimum.

As experiments show, the JDCOMM method computes the smallest real eigenvalue corresponding to the global minimum of the polynomial, also if its location is not close to the exterior of the spectrum. In this case experiments show that the proposed method may typically beat standard JD, eigs and a Krylov-Schur implementation (based on implicitly restarted Arnoldi), and standard polynomial optimization routines such as SOSTOOLS, GloptiPoly and PHCpack.

An important note is that, unfortunately, we generally can not be sure that we have indeed found the smallest eigenvalue; this is an inherent issue of iterative methods. For Hermitian eigenvalue problems certain techniques may be exploited to increase reliability such as finding more eigenvalues than asked for, block versions of the method, rerunning the method with another starting vector, "locking", etc. (see, e.g., [17]). A probabilistic approach was proposed in [4]. Still, these techniques generally do not give a 100% guarantee. For non-Hermitian problems, this matter is even more unexplored.

Finally, we remark that the proposed method may also be interesting in the situation of changing β (when $\beta \to 0$; see Section 2), as we may attempt to reuse information from a previous run. This is left for future research.

Acknowledgments: We thank the referees for their helpful suggestions.

References

[1] I. W. M. Bleylevens, R. L. M. Peeters, and B. Hanzon, *Efficiency improvement in an nD–systems approach to polynomial optimization*, Journal of Symbolic Computation, 42 (2007), pp. 30–53.

[2] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms. An Introduction to Computational Algebraic Geometry and Commutative Algebra*, Springer–Verlag, 1997.

[3] D. Cox, J. Little, and D. O'Shea, *Using Algebraic Geometry*, Springer–Verlag, 1998.

[4] J. L. M. van Dorsselaer, M. E. Hochstenbach, and H. A. van der Vorst, *Computing probabilistic bounds for extreme eigenvalues of symmetric matrices with the Lanczos method*, SIAM Journal on Matrix Analysis and Applications, 22 (2000), no. 3, pp. 837–852.

[5] G. Frobenius, *Über Vertauschbare Matrizen*, Sitzungsberichte der Königlich Preussischen Akademie der Wissenschaften zu Berlin, (1896), pp. 601–614.

[6] Y. Guan and J. Verschelde, *PHClab: A MATLAB/Octave interface to PHCpack*, IMA Volume 148: Software for Algebraic Geometry, edited by M. E. Stillman, N. Takayama, and J. Verschelde, Springer-Verlag (2008), pp. 15–32.

[7] B. Hanzon and M. Hazewinkel, *Constructive Algebra and Systems Theory*, Edita KNAW, (2006).

[8] B. Hanzon and D. Jibetean, *Global minimization of a multivariate polynomial using matrix methods*, Journal of Global Optimization, 27 (2003), pp. 1–23.

[9] B. Hanzon and J. Maciejowski, *Constructive algebra methods for the L_2–problem for stable linear systems*, Automatica, 32 (1996), pp. 1645–1657.

[10] D. Henrion and J.-B. Lasserre, *GloptiPoly: Global optimization over polynomials with Matlab and SeDuMi*, ACM Transactions on Mathematical Software (TOMS), 29 (2003), pp. 165–194.

[11] M. E. Hochstenbach and Y. Notay, *The Jacobi–Davidson method*, GAMM Mitteilungen, 29 (2006), pp. 368–382.

[12] M. E. Hochstenbach and G. L. G. Sleijpen, *Two-sided and alternating Jacobi–Davidson*, Linear Algebra Appl. 358 (2003), pp. 145–172.

[13] D. Jibetean, *Algebraic Optimization with Applications to System Theory*, Ph.D. Thesis, Vrije Universiteit, Amsterdam, (2003).

[14] J. B. Lasserre, *Global optimization with polynomials and the problem of moments*, SIAM Journal on Optimization, 11 (2001), pp. 796–817.

[15] ———, *An explicit equivalent positive semidefinite program for nonlinear 0–1 programs*, SIAM Journal on Optimization, 12 (2002), pp. 756–769.

[16] J. Rommes, *Arnoldi and Jacobi–Davidson methods for generalized eigenvalue problems $Ax = \lambda Bx$ with singular B*, Math. Comp. 77 (262) (2008), pp. 995–1015.

[17] J. R. McCombs and A. Stathopoulos, *Iterative validation of eigensolvers: a scheme for improving the reliability of Hermitian eigenvalue solvers*, SIAM Journal on Scientific Computing, 28 (2006), no. 6, pp. 2337–2358.

[18] K. Meerbergen and D. Roose, *Matrix transformations for computing rightmost eigenvalues of large sparse non-symmetric eigenvalue problems*, IMA J. Numer. Anal., 16 (1996), no. 3, pp. 297–346.

[19] K. Meerbergen, A. Spence, Implicitly restarted Arnoldi with purification for the shift-invert transformation, Math. Comp. 66 (218) (1997) 667–689.

[20] K. Meerbergen, A. Spence, D. Roose, Shift-invert and Cayley transforms for detection of rightmost eigenvalues of nonsymmetric matrices, BIT 34 (3) (1994) 409–423.

[21] H. M. Möller and H. J. Stetter, *Multivariate polynomial equations with multiple zeros solved by matrix eigenproblems*, Numerische Mathematik, 70 (1995), pp. 311–329.

[22] H. M. Möller and R. Tenberg, *Multivariate polynomial system solving using intersections of Eigenspaces*, Journal of Symbolic Computation, 32 (2001), pp. 513–531.
[23] T. L. Noorden and J. Rommes, *Computing a partial generalized real Schur form using the Jacobi–Davidson method*, Numerical Linear Algebra with Applications, 14 (2007), pp. 197–215.
[24] P. A. Parrilo, *Semidefinite programming relaxations for semialgebraic problems*, Mathematical Programming, 96 (2003), pp. 293–320.
[25] S. Prajna, A. Papachristodoulou, P. Seiler, and P. A. Parrilo, *SOSTOOLS: Sum of squares optimization toolbox for MATLAB (http://www.mit.edu/~parrilo/sostools)*, 2004.
[26] Y. Saad, *Iterative Methods for Sparse Linear Systems*, second ed., SIAM 2003.
[27] G. L. G. Sleijpen and H. A. van der Vorst, *A Jacobi–Davidson iteration method for linear eigenvalue problems*, SIAM Journal on Matrix Analysis and Applications, 17 (1996), pp. 401–425.
[28] D. C. Sorensen, *Implicit application of polynomial filters in a k-step Arnoldi method*, SIAM Journal on Matrix Analysis and Applications, 13 (1992), no. 1, pp. 357–385.
[29] G. W. Stewart, *A Krylov–Schur algorithm for large eigenproblems*, SIAM Journal on Matrix Analysis and Applications, 23 (2002), no. 3, pp. 601–614.
[30] J. Sturm, *Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones (http://sedumi.mcmaster.ca)*, Optimization Methods and Software, 11 (1999), pp. 625–653.
[31] B. Sturmfels, *Solving Systems of Polynomial Equations*, American Mathematical Society, CBMS Regional Conferences Series, No 97, 2002.
[32] J. Verschelde, *Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation*, ACM Trans. Math. Softw., 25 (1999), pp. 251–276.