ELSEVIER

Parallel Computing 27 (2001) 1879–1896

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Cobra: Parallel path following for computing the matrix pseudospectrum ☆

C. Bekas, E. Gallopoulos *

Department of Computer Engineering and Informatics, University of Patras, 26500 Patras, Greece

Received 5 November 1999; received in revised form 18 January 2000; accepted 15 March 2001

Abstract

The construction of an accurate approximation of the ϵ-pseudospectrum of a matrix by means of the standard grid method is a very demanding computational task. In this paper, we describe Cobra, a domain-based method for the computation of pseudospectra that combines predictor corrector path following with a one-dimensional grid. The algorithm offers large and medium grain parallelism and becomes particularly attractive when we seek fine resolution of the pseudospectrum boundary. We implement Cobra using standard LAPACK components and show that it is more robust than the existing path following technique and faster than it and the traditional grid method. Cobra is also combined with a partial SVD algorithm to produce an effective parallel method for computing the matrix pseudospectrum.

Keywords: Path following; Matrix pseudospectrum; Parallel computing; Singular value decomposition; Partial SVD; Shared-memory multiprocessors; NUMA; SGI Origin

1. Introduction

The ϵ-pseudospectrum of a matrix $A \in \mathbb{C}^{n\times n}$, defined as

$$\Lambda_\epsilon(A) = \{z : z \in \Lambda(A+E) \text{ for some } E \in \mathbb{C}^{n\times n} \text{ with } \|E\| \leqslant \epsilon\} \tag{1}$$

and equivalently by

☆ Supported in part by European INCO–COPERNICUS Scientific Program, Project STABLE: CP 960237.

* Corresponding author.

E-mail addresses: knb@daidalos.hpclab.ceid.upatras.gr, stratis@daidalos.hpclab.ceid.upatras.gr (E. Gallopoulos).

0167-8191/01/$ - see front matter
PII: S0167-8191(01)00121-1

$$\Lambda_\epsilon(A) = \{z : \sigma_{\min}(zI - A) \leqslant \epsilon\}, \tag{2}$$

where symbols $\Lambda(\cdot)$ and $\sigma_{\min}(\cdot)$ denote the spectrum and the smallest singular value of their matrix arguments, has become a tool for the investigation of the behavior of several (non-normal) matrix-dependent algorithms, ranging from iterative methods for large linear systems to time-stepping algorithms [9]. We note, for instance, the inclusion of specific functions to that effect (`ps` and `pscont`) in the popular Test Matrix Toolbox of MATLAB [7]; the former uses definition (1) and the latter definition (2). Computing the pseudospectrum, however, is significantly more expensive than computing traditional characteristics such as the condition number, the norm, the eigenvalues and the singular values. This is illustrated in Figs. 1(left) and (right), where we apply the functions `ps` and `pscont` on a small problem, in particular the matrix `kahan` of order $n = 50$ from the previously mentioned toolbox. The figures depict the pseudospectra and the associated costs (in millions of MATLAB `flops`) for each plot. For comparison, the costs of the full SVD and full eigendecomposition of $A - zI$ for some $z \in \mathbb{C}$ were 1.5 and 1.6 M(illion), respectively.

The standard reference method for pseudospectra is based on definition (2); we call it `GRID` and note that it estimates Λ_ϵ at a region of the complex plane Ω by first discretizing the region with a grid Ω_h, then computing $\sigma_{\min}(z_k I - A)$ for all $z_k \in \Omega_h$ and finally plotting the ϵ-contours. Therefore, its cost can be modeled by $T_{\text{GRID}} \approx |\Omega_h| C_{\sigma_{\min}}$, where $|\Omega_h|$ denotes the number of points of the grid and $C_{\sigma_{\min}}$ is the average cost for extracting the $\sigma_{\min}$. Methods that attempt to reduce costs have typically fallen in one of the following two categories.

Domain-oriented. Methods that attempt to reduce $|\Omega_h|$ by exploiting properties of the ϵ-contours and the shape of the pseudospectrum domain.

Matrix-oriented. Methods that attempt to reduce $C_{\sigma_{\min}}$ at each point.

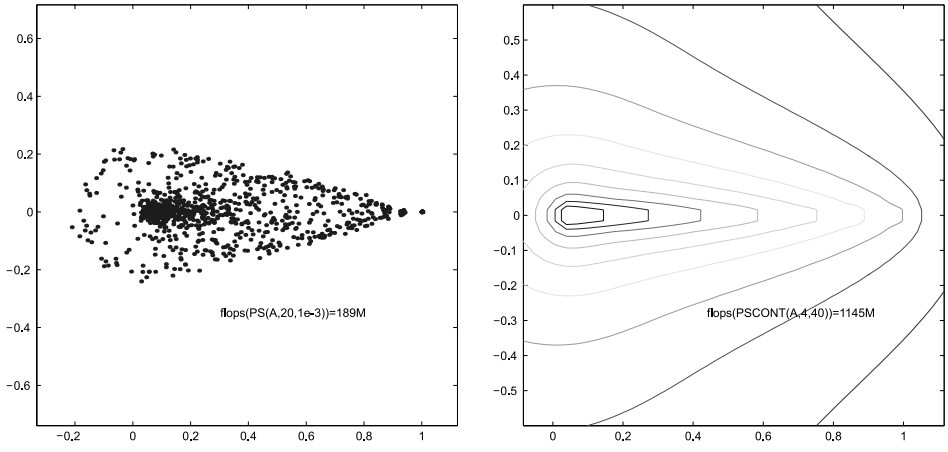

Fig. 1. Illustrations of pseudospectra for the `kahan` matrix of order $n = 50$ computed using MATLAB Test Matrix Toolbox functions `ps` (left) and `pscont` (right). The left figure shows the eigenvalues of matrix $A + E_j$ for random perturbations $E_j \in \mathbb{C}^{50\times 50}$, $j = 1, \ldots, 20$, where $\|E_j\|_2 \leqslant 10^{-3}$ and the right figure shows the level curves from a 40×40 grid defined by $\{z : \sigma_{\min}(zI - A) \leqslant \epsilon\}$ for $\epsilon = 10^{-1}$ down to 10^{-10}.

In this paper, we focus on the former category and point the reader to the recent review [10] for a comprehensive survey. Our starting point is the domain-oriented technique proposed by Brühl in [4]. This exploits the fact that the pseudospectrum $\Lambda_\epsilon(A)$ can be obtained by tracing the corresponding boundary curve(s) $\partial\Lambda_\epsilon(A)$. It is thus enough to solve the following:

Problem PSe: Given $A \in \mathbb{C}^{n\times n}$ and $\epsilon > 0$, compute $\partial\Lambda_\epsilon(A) = \{z \,|\, \sigma_{\min}(zI - A) = \epsilon\}$.

Brühl's idea was to use predictor–corrector path following to trace the pseudospectrum curve and thus offers the potential for large computational savings compared to `GRID`. As Brühl and others have reported, however, (i) the method is not always reliable, and (ii) it offers no opportunities for parallelism other than those inherent in the SVD. The contribution of this paper is `Cobra`, a method that offers increased reliability and large-grain parallelism. As we will see, the two advantages of the method are tightly coupled: large-grain parallelism is a consequence of the steps taken to improve reliability and vice-versa. `Cobra` is based on a hybrid of path following and `GRID`. It uses a small, moving one-dimensional grid that follows $\partial\Lambda_\epsilon$ almost like the neck of a cobra that follows the movements of its prey. We refer to [2], an on-line version of this work for some of the omitted technical details.

Section 2 reviews predictor–corrector path following for pseudospectra and then describes `Cobra`. Its implementation, performance for typical problems and various improvements are presented in Section 3 while Section 4 provides concluding remarks. We use standard notation: matrix $A \in \mathbb{C}^{n\times n}$ has singular value decomposition $A = U\Sigma V^*$, where the unitary matrix U (resp. V) contains the left (resp. right) singular vectors, Σ is the diagonal matrix of singular values whose minimum value is denoted by $\sigma_{\min}$ and the corresponding left and right singular vectors by $u_{\min}$ and $v_{\min}$.

2. Path following and pseudospectra

Numerical path following is a powerful tool in many areas of numerical mathematics. Predictor–corrector path following [1] proceeds as follows: initially, a point lying on the curve is computed; then the curve is traced by repeating: (i) a *prediction* step, during which a point $\tilde{z}_k$ along the general direction of traversal is computed, and (ii) a *correction* step during which one or more iterations are applied to correct $\tilde{z}_k$ to $z_k \in \partial\Lambda_\epsilon(A)$. The prediction is achieved using an Euler predictor, i.e., a step in the direction of the tangent to the curve, and the correction by means of Newton iteration. The originality of Brühl's contribution was his use of path following to compute $\partial\Lambda_\epsilon(A)$. Given ϵ, the goal is to numerically approximate the boundary

[1] For the remainder of this paper we will assume that whenever we refer to path following, we refer to the predictor–corrector variety.

$\partial\Lambda_\epsilon(A)$ of the ϵ-pseudospectrum $\Lambda_\epsilon(A)$. For this, we need to have knowledge of the differential properties of $\partial\Lambda_\epsilon(A)$. By definition, the curve is implicitly defined by the equation $g(z) = \epsilon$, where $g(x,y) := \sigma_{\min}((x + \mathrm{i}y)I - A)$. As in [4], we identify the complex plane $\mathbb{C}$ with $\mathbb{R}^2$ and identify with slight abuse of notation $g(z) = g(x + \mathrm{i}y) = g(x,y)$. Key is the following result ([8, Section 4.2] and references therein):

Theorem 1. *Let $z = x + \mathrm{i}y \in \mathbb{C} \setminus \Lambda(A)$. Then $g(x,y)$ is real analytic in a neighborhood of (x,y), if $\sigma_{\min}((x + \mathrm{i}y)I - A)$ is a simple singular value. The gradient of $g(x,y)$ is equal to*

$$\nabla g(x,y) = (\Re(v_{\min}^* u_{\min}), \Im(v_{\min}^* u_{\min})) = v_{\min}^* u_{\min},$$

where $u_{\min}$ and $v_{\min}$ denote the left and right singular vectors corresponding to $\sigma_{\min}$.

We denote Brühl's path-following algorithm by PF and listed in in Table 1.

The correction phase is implemented with Newton iteration along the direction of steepest ascent $d_k = \nabla g(\tilde{z}_k)$. Theorem 1 provides the formula for the computation of the gradient of $\sigma_{\min}(zI - A) - \epsilon$. The various parameters needed to implement the main steps of the procedure can be found in [4].

Unlike typical prediction–correction, the prediction direction r_k in PF is taken orthogonal to the previous correction direction d_{k-1}; this is in order to avoid an additional (expensive) triplet evaluation. Each Newton iteration at a point z requires the computation of the singular triplet $(\sigma_{\min}, u_{\min}, v_{\min})$ of $zI - A$. This is the dominant cost per step and determines the total cost of the algorithm; see also Section 2.1.4. Computational experience in [4] indicates that a single Newton step is sufficient to obtain an adequate correction. What makes PF so appealing is that by tracing $\partial\Lambda_\epsilon(A)$ it replaces a computation over a predefined two dimensional grid Ω_h with a computation over points on the pseudospectrum boundary. PF, however, also suffers from numerical and computational weaknesses. The former class of weaknesses were already described in [4] and might cause failure or incomplete termination of the algorithm; see Figs. 6(left) and 7(left). In the first case, there are locations where the pseudospectrum boundary has steep turns; these cause the curve tracing algorithm to

Table 1
The original PF algorithm [4]

Algorithm. PF
(**Initialization:* *)
Transform A to upper Hessenberg form and set $k = 0$.
Find initial point $z_0 \in \partial\Lambda_\epsilon(A)$

repeat
(* *Prediction phase* *)
Determine $r_k \in \mathbb{C}, \|r_k\| = 1$, steplength τ_k and set $\tilde{z}_k = z_{k-1} + \tau_k r_k$.
(* *Correction phase* *)
Correct along $d_k \in \mathbb{C}, \|d_k\| = 1$ by setting $z_k = \tilde{z}_k + \theta_k d_k$ where θ_k is some steplength.
until termination.

lose its track or retrace its own steps. In the second case, there are segments of the curve that are nearby; it may then happen that the algorithm will jump from one part of the curve to the other, thus producing wrong results. One remedy would be to keep the steplength τ_k sufficiently small to capture the detail of the curve; in the absence of dynamic stepsize adjustment, however, this strategy can be very expensive as its cost is determined by the minimum steplength required to avoid failure at the most difficult parts of the curve. [4] only mentions as ways to address these problems, the use of heuristics for dynamic steplength adaptation and the application of a small, local version of the standard `GRID` method at those points problematic to path following. `Cobra` is a derivative of the latter idea. The last difficulty is that if we rely on a single run of path following, we will miss components of non-simply connected pseudospectra. We do not discuss this any further but note that it can be overcome either by searching for all the components (in sequence or in parallel) or by combining path following with special strategies for locating the different components. Regarding the computational weaknesses, we already saw that the solution of problem PSe is expensive and motivates the use of parallel processing. `GRID`, of course, offers abundant "embarassing" large-grain parallelism. We can assign, for example, one or more gridpoints to each processor and let them compute the corresponding $\sigma_{\min}$; see experiments in [6,3]. Unfortunately, no computations are avoided: each processor performs $O(n^2/p)$ computations, even though we are only interested at those points z where $\sigma_{\min}(zI - A) = \epsilon$. Algorithm `PF` reduces the number of necessary $\sigma_{\min}$ computations from quadratic to linear in the number of points on $\partial\Lambda_\epsilon(A)$. A careful observation of the algorithm, however, reveals that the only opportunity for parallelism is in the computation of the singular triplets. Unless the matrix size is very large, however, this task is expected to offer only moderate speedups [5]. We also mention that an obvious method for extracting parallelism from `PF` is to apply it to obtain $\partial\Lambda_\epsilon(A)$ for several values of ϵ. This can be done by running one instance of the method per value of ϵ. We note that a weakness of this is that the amount of parallelism it entails depends on the number of values of ϵ that are of interest. Since this approach can also be used to introduce further parallelism in the method we are about to describe, we do not discuss it further. From now on, we concentrate on solving (a single instance of) PSe.

2.1. The `Cobra` algorithm

In this section, we present an efficient parallel algorithm that expands the basic ideas of predictor–corrector path following into a robust scheme for the construction of $\partial\Lambda_\epsilon(A)$. Assuming that a point $z_{k-1}^{\text{piv}} \in \partial\Lambda_\epsilon(A)$ that we name *pivot* of the current step is already available, `Cobra` proceeds in three phases. The first mimics `PF` and uses prediction–correction to determine a *support* point, say z_k^{sup} on $\Lambda_\epsilon(A)$. The second phase consists of two steps. The first uses the current pivot and support points in order to compute m initial approximations $\zeta_{j,0},\ j = 1,\ldots,m$ for $\partial\Lambda_\epsilon(A)$. The second step performs corrections in this initial group in order to obtain acceptable approximations $z_k^i \in \partial\Lambda_\epsilon(A),\ i = 1,\ldots,m$. The third phase uses a selection procedure to mark one of the $z_k^i \in \partial\Lambda_\epsilon(A),\ i = 1,\ldots,m$ as the next pivot. Fig. 2

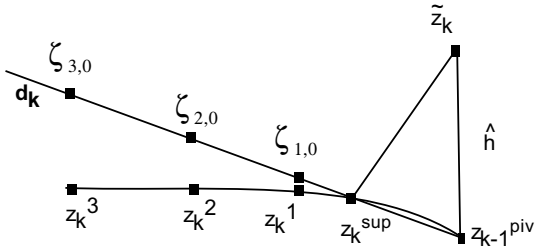

Fig. 2. Position of pivot z_{k-1}^{piv}, initial prediction $\tilde{z}_k$, support z_k^{sup}, first-order predictors $\zeta_{j,0}$ and corrected points z_k^j. The figure is not in the proper scale since in general $\hat{h} \ll H$.

illustrates the procedure. We next describe some of the design issues; for more details see [2].

2.1.1. Calculation of initial point and support points

This segment of the algorithm is identical with that used by the original PF algorithm. We seek a straight line crossing $\partial\Lambda_\epsilon(A)$ as well as its point of intersection with the curve. The procedure is formally stated as follows. We seek a solution of the equation $f(\theta) = g(\lambda + \theta d) - \epsilon = 0$ on the line $\lambda + \theta d$, where $\lambda,\ d \in C,\ |d| = 1$, $\theta \in R$, and $\lambda,\ d$ are fixed. If we assume that $\theta_0 = 0$ and that g is differentiable at λ and $\nabla g(\lambda) = v_{\min}^* u_{\min} \neq 0$, then the first iterate θ_1 is defined by $\theta_1 = \theta_0 - (f(\theta_0)/f'(\theta_0)) = -(g(\lambda) - \epsilon/f'(0))$. Theorem 1 shows that $g(\lambda) = \sigma_{\min}$ and $f'(0) = \Re(\overline{d} v_{\min}^* u_{\min})$, where $u_{\min}$ and $v_{\min}$ are the minimum singular vectors of $\lambda I - A$. Therefore, the next point is determined by $\theta_{k+1} = \theta_k - (\sigma_{\min} - \epsilon/\Re(\overline{d} v_{\min}^* u_{\min}))$ and $z_{k+1} = \lambda + \theta_{k+1} d$. The fact that the pseudospectrum is symmetric with respect to the real axis when the matrix is real was exploited in the literature in order to halve the cost of its computation. The following lemma shows that in the real case, there is is additional information that the procedure for finding the first point can exploit.

Lemma 2. *Let $A \in \mathbb{R}^{n\times n}$. Then the following hold.*

1. *There exists $\hat{\epsilon} > 0$ such that for all $\epsilon \geqslant \hat{\epsilon}$, the real axis will intersect at least one component of $\partial\Lambda_\epsilon(A)$.*
2. *If n is odd, then for any $\epsilon > 0$ the real axis will intersect at least one component of $\partial\Lambda_\epsilon(A)$.*
3. *If the matrix has only non-negative elements, then for any $\epsilon > 0$ the real axis will intersect at least one component of $\partial\Lambda_\epsilon(A)$. This component corresponds to the eigenvalue of A whose magnitude equals the spectral radius of A.*

Proof. We first note that pseudospectra are nested around the eigenvalues, that is, $\Lambda(A) = \Lambda_0(A) \subset \Lambda_{\epsilon_1}(A) \subset \Lambda_{\epsilon_2}(A)$ for $\epsilon_1 < \epsilon_2$. Item (1) follows from the fact that we can always find $E \in \mathbb{C}^{n\times n}$ such that $A + E$ has at least one real eigenvalue, so we can

use $\epsilon = \|E\|$. Item (2) holds since when n is odd, the characteristic polynomial of A will have at least one real root. Item (3) follows from standard theory of non-negative matrices. □

For all these cases, the first point in the path following procedure can be computed without using complex arithmetic by forcing the search for the first point on the real axis. A side benefit of a real starting point is that the algorithm will first traverse all the points of the segment of $\partial\Lambda_\epsilon(A)$ that lie in one half plane, after which it can stop because of the symmetry of $\partial\Lambda_\epsilon(A)$. `Cobra`, like `PF`, computes each time only that component of the pseudospectrum containing the initial point. To compute the support point $z_k^{\sup}$, we employ the same prediction–correction scheme as in `PF`. The correction step is performed using a single step of Newton iteration while the predictor uses a small stepsize $\hat{h}$.

2.1.2. Determination of starting values $\zeta_{j,0}$ and computation of $z_k^j \in \partial\Lambda_\epsilon(A)$, $j = 1, \ldots, m$

At the neighborhood of the current pivot z_{k-1}^{piv}, the curve $\partial\Lambda_\epsilon(A)$ is approximated by a first-order predictor, that is a straight line segment d_k such that $z_{k-1}^{\text{piv}} \in d_k$. The direction of d_k is determined by z_{k-1}^{piv} and the current support point $z_k^{\sup}$. The line segment extends from z_{k-1}^{piv} along d_k and for a predefined "cobra neck" length H. The line segment is discretized using m equidistant points $\zeta_{j,0}$, $j = 1, \ldots, m$ so that $\zeta_{j,0} = z_{k-1}^{\text{piv}} + jh$, where $h = H/m$ is the internal steplength for this phase. A correction procedure based on Newton iterations leads from the initial to the (approximate) points of the pseudospectrum curve, i.e., $\zeta_{j,0} \to z_k^j$. These iterations are independent from each other and can be computed in parallel. Based on the constraints for the correction paths, we implemented two types of corrections. The first one we call *constrained* and denote by VH.C (vertical–horizontal correction); in this, the correction directions are parallel to the real or imaginary axes; Let $z_k = (x_k, y_k) \in \mathbb{C}$ be a generic point and $z_k^{\sup}$ a support point, then we choose the search lines of the correction step to be horizontal or vertical depending on the value of $|x_k - \tilde{x}_k|$. We have that

$$\theta_{s+1} = \theta_s - \text{sign}(-\psi)\frac{\sigma_{\min} - \epsilon}{\Re(e^{i\psi} v_{\min}^* u_{\min})} \quad \text{and} \quad \zeta_{j,s+1} = \zeta_{j,s} + e^{i\psi}\theta_{s+1},$$

where $\psi = 0$ (resp. $\psi = \pi/2$) when we choose the horizontal (resp. vertical) direction and the sign function satisfies $\text{sign}(0) = 1$; $\theta_0 = 0$ and $(\sigma_{\min}, u_{\min}, v_{\min})$ is the triplet corresponding to the point $\zeta_{j,s}$. The second correction scheme we call *unconstrained* and denote by SD.C (steepest–direction correction); in this, corrections lie along the path of steepest descent for the error. Note that this is the same strategy used in the first phase of `Cobra` as well as in `PF`, except that here several corrections can be performed concurrently. Further details of both methods can be found in [2]. The above algorithm can be modified without much difficulty and few extra computations to use second-order information to predict the starting points $\zeta_{j,0}$. In general, these points cannot be far from z_{k-1}^{piv}. The (cobra neck) length $H = mh$ depends on the curve, which is not known a priori; in our implementation H was a moderate

multiple of the steplength $\hat{h}$ used in the first phase. When all independent iterations terminate according to some stopping criterion, the process returns m new point approximations to $\partial\Lambda_\epsilon$. We then set $z_k^j \leftarrow \zeta_{j,s_j},\ j = 1,\ldots,m$, where s_j denotes the number of correction steps performed for point $\zeta_{j,0}$. As was done in PF, we also enforced $s_j = 1$.

2.1.3. Selection of next pivot

This is formally implemented as $z_k^{\text{piv}} = \Phi(z_{k-1}^{\text{piv}}, z_k^{\text{sup}}, z_k^1, \ldots, z_k^m)$. In our implementation, Φ selects the point z_k^m. In general, we can consider a strategy for selecting the next pivot point according to the distance from z_{k-1} and the relative error of each point. The criterion is general enough to allow the return of points as far back as z_{k-1}^{piv}, if the iterations within the cobra neck are not successful, or even z_{k-1}^{piv} if both prediction–correction phases fail (in which case the procedure stalls until we modify the stepsizes). The steps of Cobra are summarized in Table 2. Using the notation established above, we tabulate the points that are computed during step k of Cobra, assuming pivot z_{k-1}^{piv} is available.

Notation	Point on $\partial\Lambda_\epsilon(A)$?	Type of point	Cobra phase
$\tilde{z}_k$	–	Predicted	1st phase
z_k^{sup}	Yes	Corrected support	1st phase
$\zeta_{1,0},\ldots,\zeta_{m,0}$	–	Predicted	1st step, 2nd phase
$z_k^j \leftarrow \zeta_{j,s_j}$	Yes	Corrected	2nd step, 2nd phase
z_k^{piv}	Yes	Pivot selected by Φ	3d phase

We already saw that the second phase of Cobra offers large-grain parallelism. The three-phase approach has additional advantages. The first phase is used to obtain the chordal direction linking the pivot and support points. For this, we can use a stepsize $\hat{h}$ that is small enough to capture the necessary detail of the curve and

Table 2
The Cobra algorithm

Algorithm [Cobra]
(* 0. *Initialization* *)
0.1. Transform A to upper Hessenberg form and set $k = 0$.
0.2. Find initial point z_0^{piv}
repeat
(* *1. First prediction-correction phase* *)
1.1. Set $k = k + 1$ and predict $\tilde{z}_k$.
1.2. Correct using Newton and compute z_k^{sup}.
(* *2. Second prediction-correction phase* *)
2.1. Predict $\zeta_{1,0},\ldots,\zeta_{m,0}$.
2.2. Correct using Newton and compute $z_k^1,\ldots,z_k^m$.
(* *3. Third phase: Pivot selection* *)
Determine next pivot z_k^{piv} using Φ.
until termination.

also be close to the tangential direction. Nevertheless, the chordal direction is often better for our purpose: for instance, if for points in the neighborhood of the current pivot $\partial\Lambda_\epsilon(A)$ is downwards convex and lies below the tangent, then the predicted points $\{\zeta_{j,0}\}_{j=1}^m$ lie closer to $\partial\Lambda_\epsilon(A)$ than if we had chosen them along the tangent. We also note that `Cobra` offers a hybrid of `PF` with `GRID`. Unlike `PF`, however, the runtime of the algorithm is not adversely affected if $\hat{h}$ is small. Instead, it is the cobra neck size H that acts as the effective stepsize. Finally, the selection procedure gives us the opportunity to reject one or more points. In the rare cases that the parallel corrections all fail, this could even return the support or pivot points; cf. Section 2.1.3. This would be equivalent to performing one step of `PF` with (small) stepsize $\hat{h}$. As illustrated in Section 3, these characteristics make `Cobra` significantly more robust than `PF`.

2.1.4. Complexity

Let C_{triplet} and C_{SVD} denote, respectively the average costs of computing the minimum singular triplet and full SVD of a complex upper Hessenberg matrix of order n over all gridpoints of the domain. We approximate the cost of each Newton iteration by the cost of a triplet computation C_{triplet}. Let the number of Newton iterations conducted during the prediction phase be $s_0^{(k)}$ and let the number of correction iterations conducted at step k be denoted by the m-tuple $(s_1^{(k)},\ldots,s_m^{(k)})$. Then the cost during step k (excluding the initialization) is $(s_0^{(k)}+\sum_{j=1}^m s_j^{(k)})C_{\text{triplet}}$. The total cost of `Cobra`, excluding initializations, is $C_{\texttt{Cobra}} = C_{\text{triplet}}\sum_k(s_0^{(k)}+\sum_{j=1}^m s_j^{(k)})$, where the outer sum is taken over the pivot points. We separate the term s_0 to emphasize that the corrections are performed only after the prediction phase has terminated; this affects the parallel implementation. Enforcing a single Newton iteration, the total sequential cost per step becomes $(1+m)C_{\text{triplet}}$. Consider a computational system that employs $p \leqslant m$ tasks running concurrently on p processors to compute the singular triplets necessary during the correction phase. Equidistributing gridpoints to processors, the total cost of one step of `Cobra` is approximately $(1+\lceil\frac{m}{p}\rceil)C_{\text{triplet}}$. A more detailed approximation has to take into account the fact that `Cobra` presents opportunities for parallelism at several levels. We are interested primarily in the *large-grain parallelism*, available when correcting $\zeta_{j,0}\to z_k^j$ for each $k=1,\ldots,m$; and the *medium-grain parallelism* available in each singular triplet evaluation. To model at this level we have to modify C_{triplet} to reflect the cost of computing the triplet on a parallel system. We consider a model system organized in loosely coupled clusters, each cluster containing a number of tightly coupled processors; we use (L,p) to denote the system configuration, where L is the number of clusters and p the number of processors per cluster. We assume that the cost of exchanging information between processors within each cluster (e.g., a shared memory multiprocessor) is very small and can be ignored. Restricting the algorithm to perform one Newton iteration, the cost per `Cobra` step is modeled by

$$C_{\texttt{Cobra-step}} = C_{\text{triplet},p\times L} + \left\lceil\frac{m}{L}\right\rceil C_{\text{triplet},p},$$

where $C_{\text{triplet},p}$ denotes the cost of the triplet computation on a single cluster of p processors. We note that the complexity of the triplet computations depends on whether we use a general algorithm that computes all triplets anyway (and not only the minimum one) or whether we apply a special algorithm that targets only the minimum triplet. For reasons of accuracy, our primary concern in this paper is the former, which carries a complexity of $O(n^3)$. This is by far the largest cost in all phases of the algorithm. We now assume that $C_{\text{triplet},p\times L} = c_{\text{triplet},p\times L} + r_{\text{triplet},L}$. The term $r_{\text{triplet},L}$ is due to the communication overhead involved in running the triplet computation on L clusters while the term $c_{\text{triplet},p\times L}$ is due to the computational and other costs of running the code on $p \times L$ processors. If we do not exploit the intercluster parallelism, the term $C_{\text{triplet},p\times L}$ is replaced by $C_{\text{triplet},p}$. We next consider the influence of communication in each phase of `Cobra` under the assumption that each cluster is responsible for the triplet and SVD computations corresponding to the points z_k^j assigned to its processors. For the purposes of this discussion, we designate one of the clusters as root or master. What remains is to specify how to implement phase 1: to avoid communication, we force it to execute on the root cluster. The remaining clusters can be used for other tasks. In that case, the data that need to be exchanged between clusters are at the end of step 1.2, the "broadcast" of z_k^{sup} to all clusters, at the end of step 2.2 the "gather" of $z_k^j, j = 1, \ldots, m$ by the root cluster, and at the end of phase 3 the broadcast of the new pivot to all clusters. It is clear that resulting amount of communication across clusters is small. Even less communication is required if we decide to have all clusters replicate the first phase of the computation. This work is redundant so we do it only if the other clusters have nothing else to do. The effect is that the broadcast of the support point at the end of step 1.2 becomes unnecessary. In summary, the communication required is at most the broadcast and gather of $O(m)$ data elements across the clusters. Remembering the fact that the computational costs for the Newton and triplet or SVD calculations can range from $O(n)$ to $O(n^3)$ and that $n \gg m$, it follows that communication is really insignificant and can be ignored in our discussion, unless we decide to perform the first phase across clusters, in which case we need to add the $r_{\text{triplet},L}$ to the cost.

We next compare the above cost with that of the original path following method. Enforcing a single Newton iteration per step, the approximate cost per step of `PF` is $C_{\text{PF-step}} = C_{\text{triplet},p\times L}$. Therefore, the total cost of `PF` relative to that of `Cobra` is

$$\frac{C_{\texttt{PF}}}{C_{\texttt{Cobra}}} = \frac{\sum_{\texttt{PF-steps}} C_{\text{triplet},p\times L}}{\sum_{\texttt{Cobra-steps}} (C_{\text{triplet}}, p \times L + \lceil m/L \rceil C_{\text{triplet},p})},$$

which simplifies to $(C_{\texttt{PF}}/C_{\texttt{Cobra}}) \approx (NC_{\text{triplet},L})/((N/m)(C_{\text{triplet},L} + C_{\text{triplet},1}))$ if (as is the case in the experiments that follow) $m = L,\ p = 1$ and we use the same number of points, say N (a multiple of m) to build the curve both in `Cobra` and `PF`. As long as we are not in a region of superlinear speedup, we can safely assume that the cost of a triplet computation on L processors is at best L times faster than the triplet computation on a single processor, and at worst equal to it. Therefore $(C_{\text{triplet}}, 1)/L \leqslant C_{\text{triplet},L} \leqslant C_{\text{triplet},1}$ and it follows from above that the speedup of `Cobra` over `PF` is bounded by $m/(m+1) \leqslant (C_{\texttt{PF}}/C_{\texttt{Cobra}}) \leqslant (m/2)$.

The above analysis shows that m, the number of gridpoints on the cobra neck, determines the amount of large-grain parallelism available in `Cobra` entirely due to path following and that the maximum speedup for `Cobra` over `PF` on $p \times L = m$ processors is $m/2$. Clearly, as the number of points on the cobra neck m increases, the better use we expect to make of an increased number of processors. This implies that the algorithm is extremely well suited for obtaining $\partial\Lambda_\epsilon(A)$ at very fine resolutions. It is fair to mention, however, that since the neck length H cannot be arbitrarily large, there is an upper bound to the number of points that are needed to adequately represent $\partial\Lambda_\epsilon(A)$. This appears to limit the amount of large-grain parallelism that can be obtained in the second phase of `Cobra`; the problem is naturally resolved as the size of the matrix increases by the work created during the triplet calculations. In summary, any growth of m or n rapidly increases the availability of large and medium grain parallelism.

3. Numerical experiments

We next conduct experiments to test and validate our previous discussion. `GRID` is bound to be more expensive than the path following methods, and hence it is not considered any further. The machine used in our studies was a non-uniform memory access (NUMA) parallel architecture, namely, the popular SGI Origin 2000. The configuration installed at the University of Patras had 8 MIPS R10000 processors and a total of 768 MB of RAM with 1 MB cache memory per processor running the IRIX 6.4 OS. The codes were written in MIPS POWER Fortran 90. We used the LAPACK implementation offered by the manufacturer's `sgicomplib` as well as its parallel version included in system library `sgicomplib_mp`. Our implementation exploits both levels of parallelism described above. The parallel implementation of the prediction phase is straightforward, since we can use the parallel version of the LAPACK routines that compute the SVD. The correction phase was performed by allocating p processors for each of the m triplets, thus computing m/L concurrently. As the running version of the O/S did not permit nested parallel calls, we did not use cluster-type organization and assumed $p = 1$ and $L = 8$ throughout. Our implementation used library calls to evaluate SVDs in the prediction phase and concurrent calls to single processor SVDs for the multiple correction phase. The directives and assertions `!$DOACROSS` and `!*$* ASSERT CONCURRENTCALL` were used to enable better parallelization. The flags used to compile the code were `-O3, -mips4, -mp`. These determined the optimization level, use of R10000 code and enabled parallelism. To measure wall clock runtime, the code segments under consideration were instrumented by enclosing them between calls to the Fortran 90 subroutine `system_clock`. For the purposes of this study, we used dense matrix methods. Since we did not want to consider here the effects of any inaccuracies in the computation of the singular triplets (resulting say from the use of inverse iteration), in the experiments of this section, we restricted ourselves to the most robust LAPACK routines coordinated by module `zgesvd` [1].

We tested our methods on non-normal matrices from [7]: (i) the upper triangular matrix `kahan` with elements $a_{kk} = s^{k-1}$ and $a_{kj} = -s^{k-1}c$ when $j > k$, where $s^{n-1} = 0.1$ and $s^2 + c^2 = 1$ [9]; (ii) the pentadiagonal Toeplitz matrix `grcar` $A = \texttt{Toeplitz}([-1, \underline{1}, 1, 1, 1])$, where the underlined element is in the main diagonal; (iii) matrix `smoke` (complex) that has unit elements in the superdiagonal and in position $(n, 1)$, powers of roots of unity along the diagonal and zero everywhere else. The next two matrices are special cases of matrix $\texttt{pentoep}(n, a, b, c, d, e)$, defined as the pentadiagonal matrix $A = \texttt{Toeplitz}([a, b, \underline{c}, d, e])$ and take their names from the shape of their pseudospectrum when $n = 32$; (iv) `fish` defined by $\texttt{pentoep}(n, 0, 1/2, 1, 1, 1)$; (v) `propeller` defined by $\texttt{pentoep}(n, 0, 1/2, 0, 0, 1)$. The matrices in categories (i–v) are already in Hessenberg form, therefore Hessenberg reduction was unnecessary.

3.1. Performance of `Cobra`: timings, accuracy, robustness

We first show that `Cobra` works correctly and compare its performance with `PF`. Parallelism in `PF` is exploited at the level of the SVD computation; to this end we used the appropriate module from library `sgicomplib_mp`. Our first experiment is with `kahan` and $\epsilon = 10^{-1}$. All eight processors of the system were utilized to solve the problem. Our timings were based on version VH.C of `Cobra`. Since we enforce a single Newton step in both the prediction and correction phases of `Cobra`, we expect similar runtimes for SD.C and VH.C. Both `PF` and `Cobra` successfully compute $\partial\Lambda_\epsilon(A)$. The timings applied to matrices of order $n = 100, 200, 300$, and 400 using $m = 8$ gridpoints for the cobra neck are depicted in Fig. 3(left). Both `Cobra` and `PF` computed the same number of points on $\partial\Lambda_\epsilon(A)$. The distance between gridpoints was set to around 10^{-2}. The exact number of points for each of the above problem

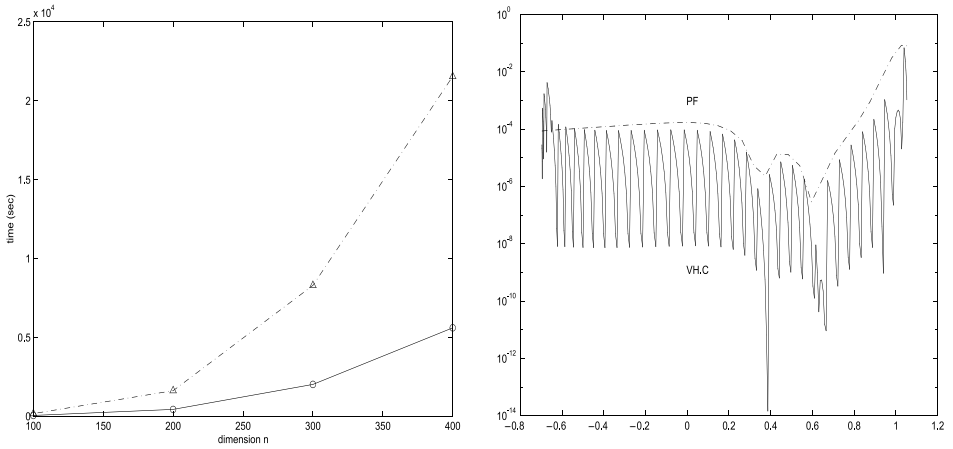

Fig. 3. (Left) Runtime for `Cobra (VH.C)` and `PF` when applied on `kahan`; (right) relative errors for `Cobra` VH.C and `PF` applied on `kahan` of order $n = 50$ with $\epsilon = 10^{-2}$. The x-axis is the x-coordinate of $\partial\Lambda_\epsilon(A)$. The dotted top curve is the `PF` error; $\hat{h} = 0.025,\ h = 0.008,\ H = 0.064$.

Table 3
Performance of PF and Cobra using $\epsilon = 10^{-2}$

Matrix	Order n	Total points N	PF time (s)	Cobra time (s)	Speedup
fish	100	1620	450	125	3.6
	200	1620	4473	1235	3.6
propeller	100	1215	405	113	3.6
	200	1620	4502	1305	3.5

sizes were $N = 320, 432, 560$ and 640. The figure shows that Cobra achieves significantly better parallel performance than PF. Table 3 presents timings for plotting $\partial\Lambda_\epsilon(A)$ for other test matrices when $\epsilon = 10^{-2}$. The number of points was chosen so that both algorithms return an accurate plot. The resulting speedups of Cobra over PF are close to 4, which is the upper bound predicted by our model (analysis at the end of Section 2.1.4) in the case of a system with eight processors, modeled by $(L, p) = (8, 1)$ with $m = L$. We note that N is quite large, which means that we obtain $\partial\Lambda_\epsilon(A)$ at fine resolution.

It was already noted in [4] that PF is more accurate than GRID. We next compare the accuracy of Cobra and PF for kahan with $n = 50$ and $\epsilon = 10^{-2}$ whose pseudospectrum level curves were plotted in Fig. 1. We first applied PF and the VH.C and SD.C versions of Cobra. Fig. 3(right) shows the relative error $|\sigma_{\min}(zI - A) - \epsilon|/\epsilon$ at points $z \in \partial\Lambda_\epsilon(A)$ computed with VH.C for Cobra and PF. We exploit symmetry and consider the error only at points lying below the real axis.

We observe that at all parts of the boundary curve the relative error of Cobra is at worst comparable to that of PF and that in general, Cobra returns more accurate results. Similar results, depicted in [2], demonstrate that SD.C Cobra obtains almost the same accuracy as VH.C. The sawtoothed shape of the error for Cobra is due to the fact that at each iteration of Cobra, the smallest error is achieved at the point lying closest to the pivot and support points (this is typically z_k^1), while the error is larger as that distance increases. We next investigate the errors for grcar of order $n = 50$. Fig. 4 shows the computed pseudospectrum curve corresponding to $\epsilon = 10^{-2}$ (left) and the relative error achieved by Cobra SD.C. The largest relative error is observed in the neighborhood of sharp turns (parts 1 and 2). We observe that these local peaks die off quickly; overall, Cobra still returns a satisfactory approximation of the pseudospectrum. Fig. 5 depicts Cobra's behavior near the 'difficult' parts 1 and 2.

We next compare with PF for grcar with $n = 64$ and $\epsilon = 10^{-2}$. Fig. 6(left) shows that PF with stepsize $\tau = 0.025$ fails while the right figure depicts the results from Cobra with neck size H and stepsizes $\hat{h} = 0.015,\ h = 0.015$. It is clear that Cobra is able to capture the pseudospectrum whereas PF fails near the steep turns. This experiment also shows the advantage of the three-phase strategy. In particular, we note that PF is successful for $\tau = 0.015$; however, these steps are executed sequentially, whereas when we use this small stepsize in Cobra we sweep a length of $H = 0.12$ per step. Since each Cobra step costs two PF steps, Cobra achieves the predicted speedup of $m/2 = 4$. The previous examples also show that neither version of Cobra

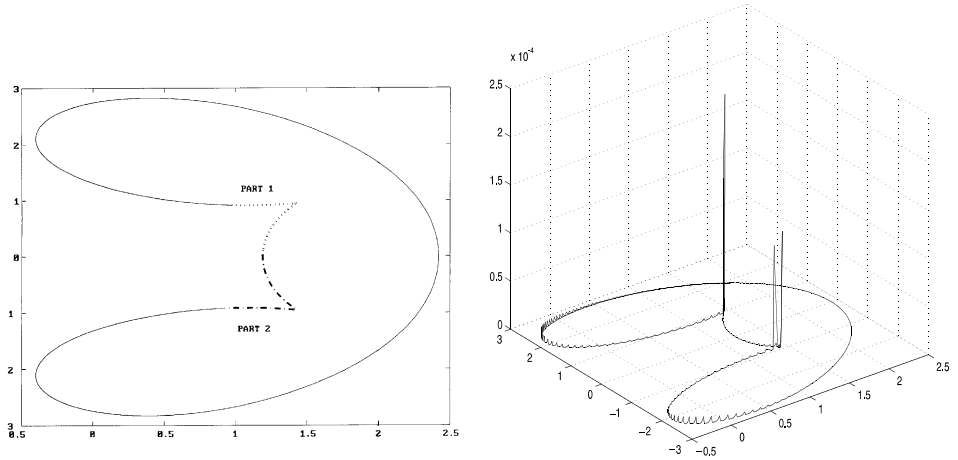

Fig. 4. (Left) $\partial\Lambda_\epsilon(A)$ for `grcar` with $n = 50$ and $\epsilon = 10^{-2}$ computed using `Cobra` SD.C; (right) relative error of $\partial\Lambda_\epsilon(A)$ computed using `Cobra` SD.C for `grcar` with $n = 50$ and $\epsilon = 10^{-2}$.

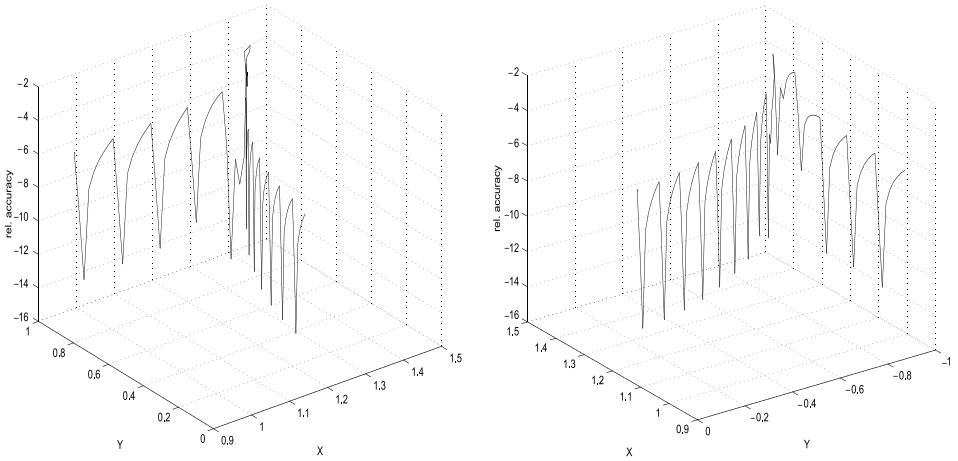

Fig. 5. Logarithm of relative error for parts 1–2 of $\partial\Lambda_\epsilon(A)$ of `grcar` with $n = 50$ and $\epsilon = 10^{-2}$ as computed by `Cobra` SD.C.

achieves uniformly better relative error, even though in most cases VH.C appears to be better. We next use `smoke` with $n = 64$ and $\epsilon = 10^{-5}$. Here the difficulty stems from the fact that in the neighborhood of $z = 1$ there are nearby segments of $\partial\Lambda_\epsilon$. These signify clustering of more than one singular values near the minimum $\sigma_{\min}$. It was noted in [4] that `PF` is likely to jump from one component to the other at those locations, unless a very small stepsize is used; see Fig. 7(left). We experimented with different stepsizes and were able to achieve a satisfactory approximation of the pseudospectrum using `PF` and over $N = 600$ steps and SVDs. Our results using

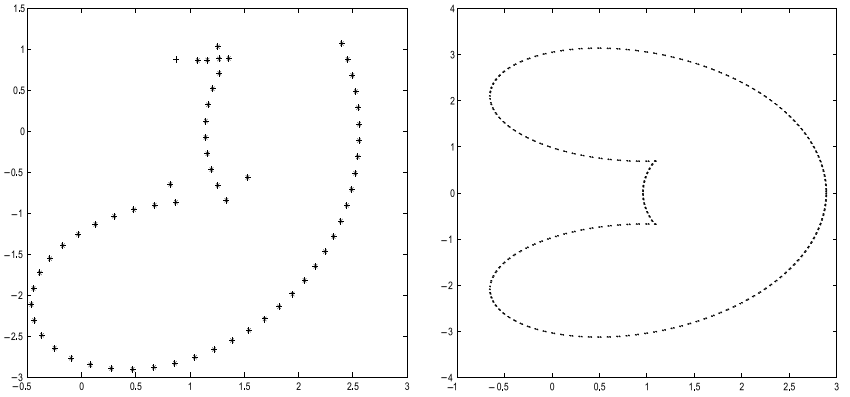

Fig. 6. Approximation of $\partial\Lambda_\epsilon(A)$ for `grcar` with $n = 64$ and $\epsilon = 10^{-2}$ using (left) `PF` and $\tau = 0.025$; (right) `Cobra` SD.C; $H = 0.12, \hat{h} = 0.015, h = 0.015$.

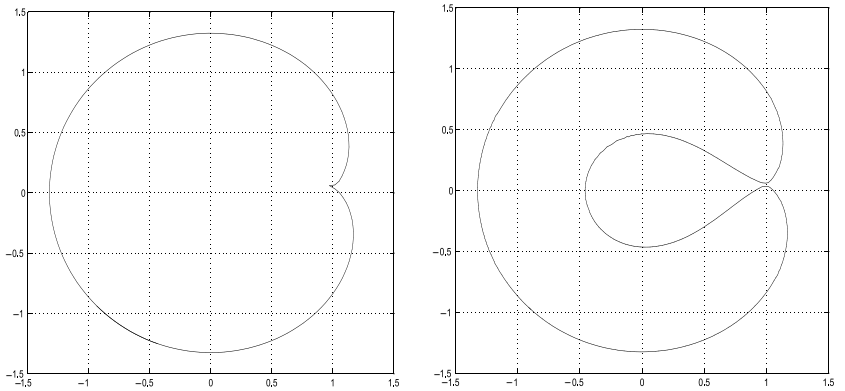

Fig. 7. Approximation of $\partial\Lambda_\epsilon(A)$ for `smoke` with $n = 64$ and $\epsilon = 10^{-5}$; (left) using `PF` and $\tau = 0.025$; (right) using `Cobra` (VH.C); $\hat{h} = 0.015$, $h = 0.0075$ and $H = 0.06$.

`Cobra` with neck size $H = 0.06$ and stepsizes $\hat{h} = 0.015,\ h = 0.0075$ are depicted in Fig. 7(right). There were 232 `Cobra` steps, each step performing $1 + 8$ triplet computations for a total of $9 \times 232 = 2088$ triplets. Since each step of `Cobra` costs about the same as two steps of `PF`, in the same amount of time `PF` would have performed only 464 steps. In fact, `PF` would have required approximately $8 \times 232 = 1856$ steps and triplet evaluations in order to create $\hat{\partial}\Lambda_\epsilon(A)$ using the same number of points as `Cobra`. We also note that `Cobra` used $h < \hat{h}$, that is the internal stepsize of the grid was smaller than the stepsize for the first phase predictor. However, `Cobra` is much faster than `PF` since it worked with neck length $H = 0.06$. In comparison, `PF` using an even smaller steplength ($\tau = 0.025$) was not successful. We conclude that `Cobra` returns a good approximation to $\partial\Lambda_\epsilon(A)$ at a much smaller parallel cost than `PF`.

3.2. Performance improvements

The analysis of Section 2.1.4 highlighted two potential areas of performance improvement: (1) for problems of medium size with general matrices, when we use the standard non-iterative algorithm implemented by LAPACK or MATLAB, we pay for unnecessary computations. This is because these implementations follow an "all or nothing" approach: either no or all singular vectors are computed; (2) the cost analysis showed that when the neck contains m gridpoints and we use m processors, the maximum potential speedup for `Cobra` over `PF` is $m/2$. This is 50% of the "perfect" speedup and is due to the predictor step.

Regarding item (2), we have found that in several cases an explicit predictor step is not necessary; instead, one can select the pivot and support points from those points on $\partial\Lambda_\epsilon(A)$ that were computed previously. When this works, `Cobra` can reach perfect speedup relative to `PF`. We do not discuss this further but concentrate on item (1). Our experiments till now computed singular triplets with LAPACK's `zgesvd`. To lower the cost we have followed an approach proposed by Van Huffel (see [11]) for computing selected triplets. Its key ingredients are: (i) the delayed computation of singular vectors and (ii) partial bidiagonalization. We thus modified Van Huffel's Partial SVD (PSVD) code (available from Netlib) in order to make it suitable for our environment: the necessary modules were rewritten for double precision complex arithmetic and segments of the code were replaced with calls to BLAS and LAPACK routines from SGI's `complib`. We designed two versions of PSVD: the first utilized only the delayed computation "trick" while the second also used partial bidiagonalization. Our implementations exploit an additional feature of the problem. Since we are only interested in the minimum singular triplet, we can avoid all computations related to $u_{\min}$; instead, we first compute $v_{\min}$ and then use

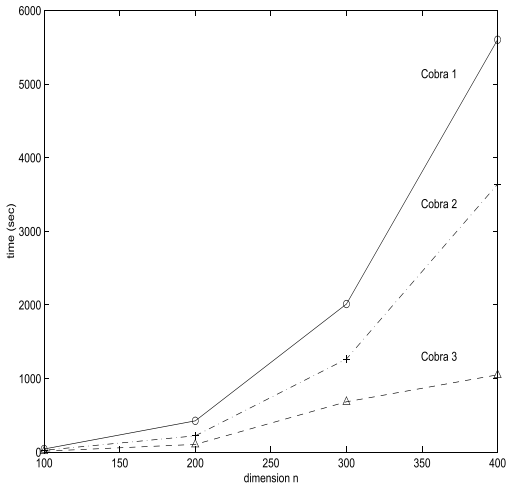

Fig. 8. Runtime for `Cobra`: with full SVD (△), with delayed reflector accumulation (○), with delayed reflector accumulation and partial bidiagonalization (+). The matrix is `kahan` and $\epsilon = 10^{-1}$.

Table 4
Performance of Cobra, $\epsilon = 10^{-2}$[a]

Matrix	Order	Total points N	Cobra + full SVD	Cobra + PSVD(1)	Cobra + PSVD(1,2)	Speedup
fish	100	1620	125 s	123 s	27 s	4.6
	200	1620	1235	962	198	6.2
propeller	100	1215	113	98	21	5.4
	200	1620	1305	997	184	7.1

[a] Col. 4 uses full SVD, col. 5 uses PSVD with delayed reflector accumulation, col. 6 uses partial bidiagonalization. Col. 7 shows speedups of col. 6 over col. 4.

relation $(H - zI)v_{\min} = \sigma_{\min}u_{\min}$. This trick further reduces the cost since it replaces the accumulation of elementary transformations of $u_{\min}$ with a single matrix vector multiplication. We ran the resulting codes using the same data we used in Section 3.1 above and compared with the results we obtained earlier. Fig. 8 consists of the same data used for Fig. 3 supplemented with the two versions of the algorithm described above. Similarly, Table 4 compares the Cobra timings presented in Table 3 together with the runtime from the two versions of PSVD. The results clearly demonstrate the significant timing improvements obtained when Cobra (or PF) is implemented using a cost effective algorithm for computing the minimum triplets. We also mention that the numerical results of the first version of the Cobra PSVD approach and standard Cobra were practically identical, whereas the results differed somewhat when both PSVD tricks were applied.

4. Concluding remarks: towards the effective computation of pseudospectra

We have shown that Cobra can be very effective for solving the pseudospectrum problem PSe, particularly when we demand a fine resolution for the construction of the boundary. Interesting research issues arise. Some concern Cobra strategies (e.g., second-order information to predict new points, analytical estimates of accuracy, etc.) for which we point to forthcoming reports at the same URL for the extended version of this paper. Another area, pursued by Bertrand and Philippe, is to use residue calculus to count eigenvalues. Cobra could also profit from adaptive stepsize selection, but the need is much less acute than in PF. Another area is to extend Cobra to bring geometry-based parallelism into other path following applications. Even though Cobra is much less susceptible to failure than PF, problems could still occur at points of non-differentiability of $\Lambda_\epsilon(A)$ and it is worth exploring techniques to handle this. We saw that PSVD speeds up Cobra significantly; it would be interesting to examine alternative methods for selected triple computation as well. Cobra of course does not solve all possible problems in pseudospectra computations. The cost could still become prohibitive for large matrices; iterative methods for computing the minimum triplets would then become necessary. Cobra can also be combined with resolvent norm estimates. These possibilities suggest that the most

effective approach would be a polyalgorithm that would extract useful information about the input matrix and the available computational resources and would then construct an optimal method from components such as `Cobra`.

Acknowledgements

A preliminary presentation of `Cobra` was given at the 1998 Copper Mountain Conference. We are grateful to our colleagues in the project STABLE (Stability of Physical Systems Using Parallel Computers) especially its coordinator, Bernard Philippe, for many fruitful discussions. We acknowledge the helpful comments of Martin Brühl regarding his method as well as discussions and comments of Kyle Gallivan, Yannis Koutis, George Moustakides, Valeria Simoncini, Ahmed Sameh, Masha Sosonkina and Layne Watson. The second author wishes to thank Jesse Barlow who was his host at the Department of Computer Science and Engineering at the PennState University where this paper was completed. We finally thank the referees who helped us improve the paper.

References

[1] E. Anderson et al., LAPACK Users' Guide, second ed., SIAM, Philadelphia, 1995.
[2] C. Bekas, E. Gallopoulos, Cobra: Parallel path following for computing the matrix pseudospectrum, At `www.hpclab.ceid.upatras.gr` in `faculty/stratis/Papers`.
[3] T. Braconnier, Fvpspack: A Fortran and PVM package to compute the field of values and pseudospectra of large matrices. Numerical Analysis Report No. 293, Manchester Centre for Computational Mathematics, Manchester, England, August 1996.
[4] M. Brühl, A curve tracing algorithm for computing the pseudospectrum, BIT 33 (3) (1996) 441–445.
[5] J. Demmel, K. Stanley, The performance of finding eigenvalues and eigenvectors of dense symmetric matrices on distributed memory computers, in: D.H. Bailey, et al. (Eds.), Proc. 7th Siam Conf. Paral. Proc. Sci. Comput, SIAM, Philadelphia, 1995, pp. 528–533.
[6] V. Frayssé, L. Giraud, V. Toumazou, Parallel computation of spectral portraits on the Meiko CS2, in: H. Liddell, et al. (Eds.), Lecture Notes in Coumpter Science: High-Performance Computing and Networking, vol. 1067, Springer, New York, 1996, pp. 312–318.
[7] N.J. Higham, The Test Matrix Toolbox for MATLAB (version 3.0). Technical Report 276, Manchester Centre for Computational Mathematics, September 1995.
[8] G.W. Stewart, J.G. Sun, Matrix Perturbation Theory, Academic Press, Boston, 1990.
[9] L.N. Trefethen, Pseudospectra of matrices, in: D.F. Griffiths, G.A. Watson (Eds.), Numerical Analysis 1991, Proc. 14th Dundee Conf., Longman Science and Technology, Essex, UK, 1991, pp. 234–266.
[10] L.N. Trefethen, Computation of pseudospectra, in: Acta Numerica 1999, vol. 8, Cambridge University Press, Cambridge, 1999, pp. 247–295.
[11] S. Van Huffel, J. Vandewalle, The Total Least Squares Problem: Computational Aspects and Analysis, SIAM, Philadelphia, 1991.