ELSEVIER

Parallel Computing 28 (2002) 223–242

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Parallel computation of pseudospectra by fast descent ☆

C. Bekas *, E. Gallopoulos

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

Abstract

The pseudospectrum descent method (PsDM) is proposed, a new parallel method for the computation of pseudospectra. The idea behind the method is to use points from an already existing pseudospectrum level curve $\partial\Lambda_\epsilon$ to generate in parallel the points of a new level curve $\partial\Lambda_\delta$ such that $\delta < \epsilon$. This process can be continued for several steps to approximate several pseudospectrum level curves lying inside the original curve. It is showed via theoretical analysis and experimental evidence that PsDM is embarrassingly parallel, like GRID, and that it adjusts to the geometric characteristics of the pseudospectrum; in particular it captures disconnected components. Results obtained on a parallel system using MPI validate the theoretical analysis and demonstrate interesting load-balancing issues. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Pseudospectra; Parallel computation; Newton's method; ARPACK

1. Introduction and motivation

The ϵ-pseudospectrum $\Lambda_\epsilon(A)$ of a matrix $A \in \mathbb{C}^{n\times n}$ can be defined in terms of the singular value decomposition (SVD) $A = U\Sigma V^*$ as the set of points z of the complex plane that satisfy

☆ This work has been partially supported by the Greek General Secretariat for Research and Development, Project ΠΕΝΕΔ 99-07.

* Corresponding author.

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

0167-8191/02/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0167-8191(01)00137-5

$$\Lambda_\epsilon(A) = \{z \in \mathbb{C} : \sigma_{\min}(zI - A) \leqslant \epsilon\}, \tag{1}$$

where $\sigma_{\min}(A)$ denotes the smallest singular value of A. The standard algorithm (GRID) for the computation of $\Lambda_\epsilon(A)$ is the following: (i) define a mesh Ω_h on a region of the complex plane and (ii) compute $\sigma_{\min}(zI - A)$ for every node $z \in \Omega_h$. The boundary curves $\partial\Lambda_\epsilon(A)$ are obtained as the contour plots of the smallest singular values on the mesh Ω_h. The obvious advantage of GRID is its straightforward simplicity and robustness. If, for the moment, we let $C_{\sigma_{\min}}$ be a measure of the average cost for the computation of $\sigma_{\min}(zI - A)$, it is easily seen that the total cost of GRID can be approximated by

$$C_{\mathrm{GRID}} = |\Omega_h| C_{\sigma_{\min}}, \tag{2}$$

where $|\Omega_h|$ denotes the number of nodes of Ω_h. The total cost quickly becomes prohibitive with the increase of either the number of nodes or the size of A.

It has been observed that the cost formula (2) readily indicates two major methods for accelerating the computation: (a) reducing the number of nodes z and hence the number of evaluations of $\sigma_{\min}$, and (b) reducing the cost of each evaluation of $\sigma_{\min}(zI - A)$. Both approaches are the subject of active research; see [15] for a comprehensive survey of recent efforts.

The use of path following, a powerful tool in many areas of applied mathematics, in order to compute a single boundary curve $\partial\Lambda_\epsilon(A)$ was suggested by Kostin in [8]. It was Brühl in [5] who presented an algorithm to that end and showed that significant savings can be achieved compared to GRID when seeking a small number of boundary curves. The key is that tracing a single boundary curve drastically reduces the number of $\sigma_{\min}$ evaluations. Bekas and Gallopoulos [2] carried this work further in Cobra, a method that ameliorated two weaknesses of the original path following method, in particular (i) its lack of large-grain parallelism and (ii) its frequent failure near sharp turns or neighboring curves. In particular, parallelism was introduced by incorporating multiple, decoupled, corrections. More recently, Mezher and Philippe [13] suggested PAT, a new path following method that reliably traces contours. Despite its advantages over the original method of [5], Cobra maintains two weaknesses of the path following approach vis-à-vis GRID. These are (a) that in each run, a *single* boundary curve $\partial\Lambda_\epsilon(A)$ is computed (b) that is *closed*. Therefore, only one curve is computed at a time and its disconnected components are not captured, at least in a single run. While we can consider the concurrent application of multiple path following procedures to remedy these problems, the solution is less straightforward than it sounds, especially for (b). In the remainder of this paper, when our results do not depend on the exact version of path following that we choose to use, we denote these methods collectively by PF.

In this paper we propose the pseudospectrum descent method (PsDM) that takes an approach akin to PF but results in a set of points defining pseudospectrum boundaries for several values of ϵ. PsDM starts from an initial contour $\partial\Lambda_\epsilon(A)$, approximated by N points z_k computed by a PF method. These points are corrected towards directions of steepest descent, to N points $w_k \in \partial\Lambda_\delta(A)$, $\delta < \epsilon$. The process can

be repeated recursively, without the need to reuse PF, and computes pseudospectra contours. PsDM can be viewed as a "dynamic" version of GRID where the stride from mesh point to mesh point has been replaced by a PF prediction–correction step. To illustrate this fact and thus provide the reader with an immediate feeling of the type of information obtained using the method, we show in Fig. 1 the results from the application of PsDM to matrix kahan of order 100 which has been obtained from the Test Matrix Toolbox [7] and is a typical example of matrices with interesting pseudospectra. The plot shows (i) the trajectories of the points undergoing the steepest descent and (ii) the corresponding level curves. The intersections are the actual points computed by PsDM.

We note that the idea for plotting the pseudospectrum using descent owes to an original idea of Koutis for the parallel computation of eigenvalues using descent described in [3].

The rest of this paper is organized as follows. Section 2 briefly reviews path following. Section 3 describes PsDM. Section 4 illustrates the characteristics of PsDM as well as its parallel performance using an MPI implementation. We also introduce adaptivity and show that the method can reveal disconnected components. Section 5 presents our conclusions.

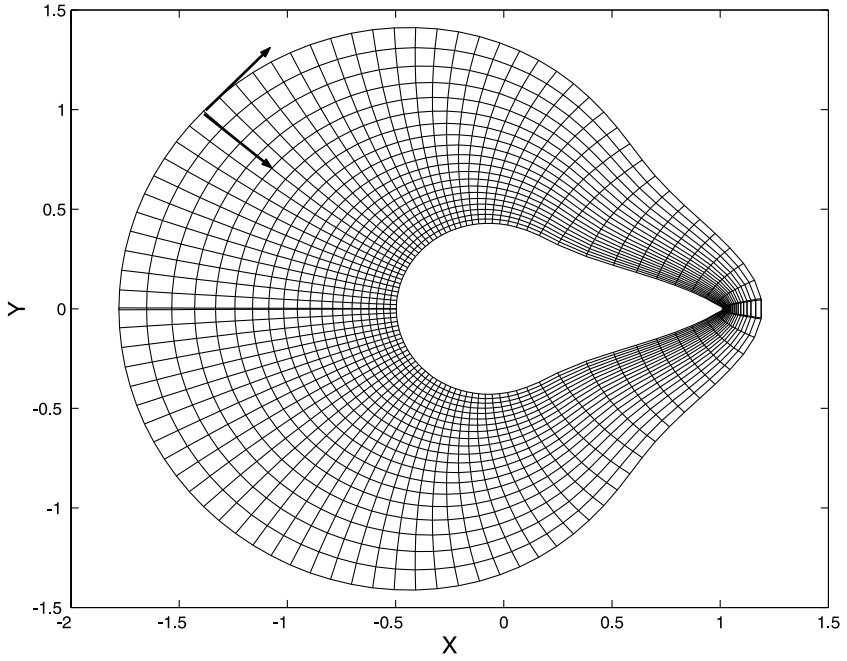

Fig. 1. Pseudospectrum contours and trajectories of points computed by PsDM for $\partial\Lambda_\epsilon(A)$, $\epsilon = 10^{-1}, \ldots, 10^{-3}$, for matrix kahan of order 100. Arrows show the directions used in preparing the outermost curve with path following and the directions used in marching from the outer to the inner curves with PsDM. See Section 4 for further results with this matrix.

2. Review of path following

Consider the function $G : \mathbb{C} \to \mathbb{R}$ where

$$G(z) = \sigma_{\min}(zI - A) - \epsilon. \tag{3}$$

According to the definition of the pseudospectrum, the zeros of $G(z)$ are points of the boundary $\partial\Lambda_\epsilon(A)$. Allgower and Georg in [1] describe a generic procedure to numerically trace solution curves of equations such as (3). In Table 1 we outline the algorithm and illustrate one step in Fig. 2. See [2,5] for more details.

In the sequel, as in [5], we would be identifying the complex plane $\mathbb{C}$ with $\mathbb{R}^2$ and frequently use the notation $G(z)$ for $G(x, y)$. Critical to the effective use of PF is the availability of the gradient $\nabla G(x, y)$; that this becomes available at little cost follows from the following important result (cf. [5,6]):

Theorem 2.1. *Let $z \in \mathbb{C} \setminus \Lambda(A)$. Then $G(x, y)$ is real analytic in a neighborhood of $z = x + \mathrm{i}y$ if $\sigma_{\min}(zI - A)$ is a simple singular value. Then the gradient of G at z is*

$$\nabla G(x, y) = (\Re(v_{\min}^* u_{\min}), \Im(v_{\min}^* u_{\min})), \tag{4}$$

where $u_{\min}$, $v_{\min}$ denote the left and right singular vectors corresponding to $\sigma_{\min}(zI - A)$.

Table 1
PF generic procedure for the computation of a single contour

(**Input**): Point z_0 on $\partial\Lambda_\epsilon(A)$.
for $k = 1, \ldots$
  (**Prediction phase**)
    1.1. Determine a prediction direction p_k.
    1.2. Choose a step length h and predict the point $\tilde{z}_k = z_{k-1} + hp_k$.
  (**Correction phase**)
    2.1. Determine a correction direction c_k.
    2.2. Correct $\tilde{z}_k$ to z_k using Newton iteration.
end

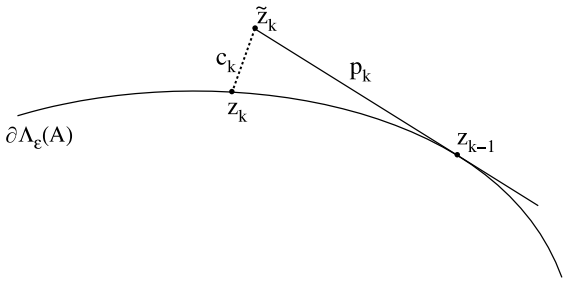

Fig. 2. A generic prediction–correction PF scheme.

Therefore, assuming that the minimum singular triplet $(\sigma_{\min}, u_{\min}, v_{\min})$ of $zI - A$ (in the sequel we would be referring to it simply as "the triplet at z") has been computed, the gradient is available with one inner product.

From the above it follows that the dominant effort in one step of the PF scheme proposed in [5] amounts to (i) the estimation of the prediction direction p_k and (ii) the Newton iteration of the correction procedure. Regarding (ii) we note that it is sufficient (cf. [2,5]) to use a single Newton step, requiring one triplet evaluation at $\tilde{z}_k$. Then, as proposed in [5], $\tilde{z}_k$ is corrected towards the direction of steepest descent c_k, i.e.

$$z_k = \tilde{z}_k - \frac{\sigma_{\min} - \epsilon}{u_{\min}^* v_{\min}}, \tag{5}$$

where the triplet $(\sigma_{\min}, u_{\min}, v_{\min})$ is associated with $\tilde{z}_k$. Regarding (i), note that selecting p_k to be tangential to the curve at z_{k-1} requires the triplet at z_{k-1}. Since this would double the cost, it has been shown acceptable to take p_k orthogonal to the previous correction direction c_{k-1}; cf. [2,5]. Therefore the overall cost of a single step of the original method of [5] is approximately equal to the cost for computing the triplet.

3. The pseudospectrum descent method

Let us now assume that an initial contour $\partial\Lambda_\epsilon(A)$ is available in the form of some approximation (e.g. piecewise linear) based on N points z_k previously computed using some version of PF. In order to obtain a new set of points that define an inner level curve we proceed in two steps:

Step 1: Start from z_k and compute an intermediate point $\tilde{w}_k$ by a single modified Newton step towards a steepest descent direction d_k obtained earlier.
Step 2: Correct $\tilde{w}_k$ to w_k using a Newton step along the direction l_k of steepest descent at $\tilde{w}_k$.

Fig. 3 illustrates the basic idea for a single initial point. Applying one Newton step at z_k would require $\nabla\sigma_{\min}((x_k + \mathrm{i}y_k)I - A)$, and, therefore, a triplet evaluation at z_k. To avoid this extra cost, we apply the same idea as PF and use instead $\nabla\sigma_{\min}((\tilde{x}_k + \mathrm{i}\tilde{y}_k)I - A) = (\Re g_{\min}^* q_{\min}, \Im g_{\min}^* q_{\min})$, which is already available from the original path following procedure. In essence we approximate the gradient based at z_k with the gradient based at $\tilde{z}_k$. Vectors $g_{\min}$, $q_{\min}$ are the right and left singular vectors associated with $\sigma_{\min}(\tilde{z}_k I - A)$. Applying correction as in (5) it follows that

$$\tilde{w}_k = z_k - \frac{\epsilon - \delta}{q_{\min}^* g_{\min}}, \tag{6}$$

where $\delta < \epsilon$ and $\partial\Lambda_\delta(A)$ is the new pseudospectrum boundary we seek. Once we have computed $\tilde{w}_k$, we perform a second Newton step that yields w_k:

$$w_k = \tilde{w}_k - \frac{\sigma_{\min}(\tilde{w}_k I - A) - \delta}{u_{\min}^* v_{\min}}, \tag{7}$$

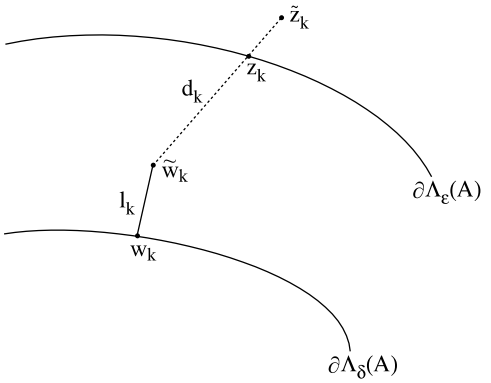

Fig. 3. Computing $\partial\Lambda_\delta(A)$, $\delta < \epsilon$.

where the triplet used is associated with $\tilde{w}_k$. These steps can be applied to all N points in what we call *one sweep of* PsDM; we denote it by σPsDM and outline it in Table 2. Starting from an initial contour $\partial\Lambda_\epsilon(A)$ we have shown how to compute points that approximate a nearby contour $\partial\Lambda_\delta(A)$, $\delta < \epsilon$.

Assume now that the new points computed with one sweep of PsDM define satisfactory approximations of $\partial\Lambda_\delta(A)$ (cf. end of section). We ask whether it would it be practical to use these points to march one further step to approximate another $\partial\Lambda_{\hat{\delta}}(A)$, $\hat{\delta} < \delta < \epsilon$. As noted in the previous discussion, the application of the sweep σPsDM uses $\nabla\sigma_{\min}((\tilde{x}_k + \mathrm{i}\tilde{y}_k)I - A)$, i.e. the triplet at $\tilde{z}_k$. This is readily available when the curve $\partial\Lambda_\epsilon(A)$ is obtained via PF.

Observe now that as the sweep proceeds to compute $\partial\Lambda_{\hat{\delta}}(A)$ from $\partial\Lambda_\delta(A)$, it also computes the triplet at $\tilde{w}_k$. Therefore enough derivative information is available for the sweep σPsDM to proceed one more step with starting points computed via the previous application of σPsDM. Therefore, it is not necessary to run PF again. Continuing with this repeated application of sweeps σPsDM, we obtain the promised pseudospectrum descent method, outlined in Table 3 and illustrated in Fig. 4.

It is worth noting that each sweep can be viewed as a map that takes as input N_{in} points approximating $\Lambda_{\delta_i}(A)$ and produces N_{out} points approximating $\Lambda_{\delta_{i+1}}(A)$ ($\delta_{i+1} < \delta_i$). In the description so far $N_{\text{in}} = N_{\text{out}}$, but as we show in Section 4.4, this is not necessarily an optimal strategy.

Table 2
σPsDM: one sweep of PsDM

(**Input**): N points z_k of the initial contour $\partial\Lambda_\epsilon(A)$.
(**Output**): N points w_k on the target contour $\partial\Lambda_\delta(A)$, $\delta < \epsilon$.
for $k = 1, \ldots, N$
1. Compute the intermediate point $\tilde{w}_k$ according to (6).
2. Compute the target point w_k using (7).
end

Table 3
The PsDM method

Method PsDM
(**Input**): N points z_k approximating a contour $\partial\Lambda_\epsilon(A)$.
(**Output**): $M \times N$ points approximating M contours $\partial\Lambda_{\delta_i}$, $\epsilon > \delta_1 > \cdots > \delta_M$.
$\delta_0 = \epsilon$.
for $i = 1, \ldots, M$
Compute N points of $\partial\Lambda_{\delta_i}$ by σPsDM on the N points of $\partial\Lambda_{\delta_{i-1}}$
end

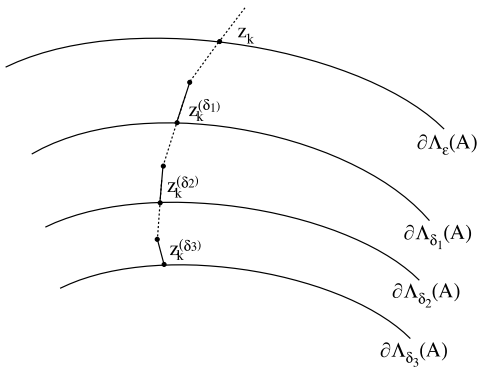

Fig. 4. Descent process for a single point.

We finally note that the two-step process described above was found to be necessary. By contrast, the more straightforward approach in which we compute the exact steepest descent directions from each point at the starting curve and use those values directly to approximate the next set of points in one step required a much smaller step size to be successful. We did not experiment further with this procedure.

3.1. Cost considerations

It is evident that the cost for the computations of the intermediate points $\tilde{w}_k$, $k = 1, \ldots, N$, is relatively small since the derivatives at $\tilde{z}_k$, $k = 1, \ldots, N$, have already been computed by PF or a previous sweep of PsDM. Furthermore, we have assumed that $\sigma_{\min}(z_k I - A) = \epsilon$ for all k, since the points z_k approximate $\partial\Lambda_\epsilon(A)$. On the other hand, computing the final points w_k, $k = 1, \ldots, N$, requires N triplet evaluations. To avoid proliferation of symbols, we use $C_{\sigma_{\min}}$ to now denote the average cost for computing the triplet. Therefore, we approximate the cost of a single sweep of PsDM by $C_{\sigma\text{PsDM}} = NC_{\sigma_{\min}}$. The computation of each target point w_k ($k = 1, \ldots, N$) is decoupled from the computation of all other target points w_j, $j \neq k$. On a system with P processors, we can assign the computation of at most $\lceil N/P \rceil$ target points to each processor; one sweep will then proceed with no need for synchronization and communication and its total cost is approximated by $C_{\sigma\text{PsDM}} = \lceil N/P \rceil C_{\sigma_{\min}}$. We discuss

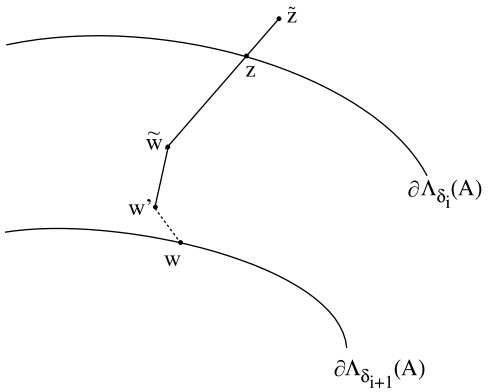

Fig. 5. The transformation of z to w' via σPsDM.

this issue in more detail in Section 4.3. We emphasize that, typically, the number of points N on each curve is expected to be large, and therefore the algorithm is scalable. This is better than the PF methods described in Section 1: Cobra allows only a moderate number of independent calculations of $\sigma_{\min}$ per step while in the original version of PF, coarse-grain parallelism becomes available only if we attempt to compute different curves in parallel.

3.2. Error analysis for a single sweep of PsDM

In order to gauge the quality of the approximation of the curve obtained in a single sweep of PsDM, it is natural to use as a measure the value $|\delta_i - \delta_{i+1}|$, which we call "step size of the sweep". Let $w \in \partial\Lambda_{\delta_{i+1}}(A)$ be the point approximated starting from a point $z \in \partial\Lambda_{\delta_i}(A)$. Define the function $G_0(z) = \sigma_{\min}(zI - A)$, and therefore $G_0(\tilde{z}) = \tilde{\delta} < \delta_i$, and let w' be the point obtained using a single Newton step at $\tilde{w}$, as depicted in Fig. 5. [1] Function G_0 is real analytic in a domain $\mathscr{D}$ if the minimum singular value $\sigma_{\min}(wI - A)$ is simple for all $w \in \mathscr{D}$. Let all five points $\tilde{z}$, z, $\tilde{w}$, w' and w (see Fig. 5) lie in the interior of such a domain. From the relations

$$z = \tilde{z} - \frac{\sigma_{\min}(\tilde{z}I - A) - \delta_i}{(u^*_{\min} v_{\min})_{\tilde{z}}} \quad \text{and} \quad \tilde{w} = z - \frac{\delta_i - \delta_{i+1}}{(u^*_{\min} v_{\min})_{\tilde{z}}}, \tag{8}$$

it follows that w' can also be considered as the outcome of two exact Newton steps originating from $\tilde{z}$. From the identity $(w'I - A) = (wI - A) + (w' - w)I$ and standard singular value inequalities, it follows that

$$\sigma_{\min}(w'I - A) \leqslant \sigma_{\min}(wI - A) + \sigma_{\max}((w' - w)I), \tag{9}$$

[1] This figure can be considered as an enlargement of Fig. 3 that reveals that the point computed with a single Newton step in the sweep produces only an approximation of w_k.

and therefore $|\delta' - \delta_{i+1}| \leqslant |w' - w|$, where $\delta' = \sigma_{\min}(w'I - A)$. From results of Levin and Ben-Israel (see [12]) regarding Newton's method for underdetermined systems, under standard assumptions there will be a region in which the Newton iteration used to produce w' from $\tilde{w}$ will converge quadratically; we next assume that we are within this region. Then $|w' - w| \leqslant \kappa_1 |w' - \tilde{w}|^2$, where $\kappa_1 = \mathrm{O}(1/\|\nabla G_0(\tilde{w}_x, \tilde{w}_y)\|)$; note that $1/\|\nabla G_0(\tilde{w}_x, \tilde{w}_y)\|$ is at least 1, since $G_0(\tilde{w}_x, \tilde{w}_y)$ is the inner product of singular vectors (cf. Theorem 2.1). Therefore, using relations (7) and (9) it follows that

$$|\delta' - \delta_{i+1}| \leqslant \kappa_1 |w' - \tilde{w}|^2 \leqslant \kappa_1 \frac{|\tilde{\delta} - \delta_{i+1}|^2}{\|\nabla G_0(\tilde{w}_x, \tilde{w}_y)\|^2} \leqslant \kappa |\tilde{\delta} - \delta_{i+1}|^2 \leqslant \kappa |\delta_i - \delta_{i+1}|^2,$$

where $\kappa = \mathrm{O}(1/\|\nabla G_0(\zeta_x, \zeta_y)\|^3)$ for $\zeta \in \mathscr{D}$ and $\tilde{\delta} = \sigma_{\min}(\tilde{w}I - A)$. It follows that if this minimum is not much smaller than 1 and the assumptions made above hold, then the error $|\delta' - \delta_{i+1}|$ induced by one sweep of `PsDM` will be bounded by a moderate multiple of the square of the step size of the sweep. A full-scale analysis of the global error, for multiple sweeps of `PsDM`, is the subject of current work.

4. Refinements and numerical experiments

We conducted experiments with matrices that have been used in the literature to benchmark pseudospectra algorithms. Our results, presented below, indicate that `PsDM` returns the same levels of accuracy as `GRID` at a fraction of the cost. We then describe a parallel implementation of `PsDM`. The speedups obtained underscore the parallel nature of the algorithm. We also show that the application of `PsDM` to large matrices for which the triplets have to be computed via some iterative method is likely to suffer from load imbalance and speedups that are inferior than what one would expect from an embarrassingly parallel algorithm; we describe a simple heuristic to address this problem.

Finally we discuss some other properties that underline the flexibility of `PsDM`. We show in particular that: (i) adaptation of the number of points computed in each sweep of `PsDM` can lead to significant cost savings, and (ii) `PsDM` can capture disconnected components of the pseudospectrum lying inside the initial boundary computed via `PF`.

4.1. System configuration

We performed our experiments on an `SGI Origin 2000` system with eight `MIPS R10000` processors. The system had a total of 768 MB RAM and 1 MB cache per processor, running `IRIX 6.5`. The codes were written in Fortran-90 using `F90/77 MIPSpro` version 7.2.1 compilers. For the parallelization we used the `MPI` programming paradigm, implemented by `SGI`'s `MPT 1.2.1.0`. We used `ARPACK` [11] to approximate the triplets and `SPARSKIT` [14], suitably modified to handle

double-precision complex arithmetic, for sparse matrix-vector multiplies. All our experiments were conducted in single-user mode.

4.2. Numerical experiments with PsDM

We remind the reader that, for matrices with real elements, the pseudospectrum curves are symmetric with respect to the real axis, and it has become standard in the pseudospectrum literature to measure the success of methods for pseudospectra by direct comparison of the one half of the figure computed with a new method and the other half with GRID. Therefore, in subsequent experiments with any method we only compute points and curves lying on the upper or lower half of the complex plane. We start with the same matrix used to obtain Fig. 1, that is kahan of order 100. We used the parallel version of PF, namely Cobra, to obtain $N = 46$ points to approximate (one half of) the pseudospectrum curve corresponding to $\epsilon = 10^{-1}$; we then asked PsDM to compute 60 contours corresponding to values of ϵ ranging from $\delta_1 = 0.9 \times 10^{-1}$ to $\delta_{60} = 10^{-7}$ using a step size that remained equal to 10^{-s-1} for values between $\delta_s = 0.9 \times 10^{-s}$ and $\delta_{s+1} = 10^{-s-1}$ for $s = 1, \ldots, 6$. The upper half of Fig. 6 illustrates the contours corresponding to $\epsilon = 10^{-1}, 10^{-2}, \ldots, 10^{-7}$ while the lower half illustrates the same contours computed using GRID on a 100×100 mesh of equidistant points. We chose this resolution in order to allow GRID to offer a level of detail that is between the minimum and maximum resolution offered by PsDM.

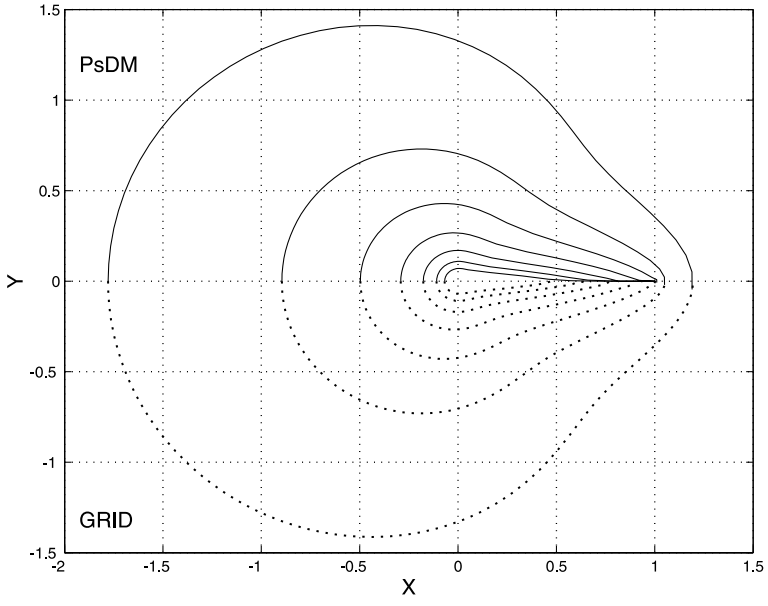

Fig. 6. Selected pseudospectrum contours $\partial\Lambda_\epsilon(A)$, $\epsilon = 10^{-1}, \ldots, 10^{-7}$, for matrix kahan of order 100 computed by PsDM (top) and GRID (bottom).

In particular, the distance between neighboring points of GRID was selected near the median of the smallest and largest distance of neighboring points computed in the course of PsDM. The pictures are virtually indistinguishable and indicate that PsDM achieves an accuracy comparable to GRID. On the other hand, the cost is much smaller; in particular PsDM approximates the contours using 2700 points while GRID used 10 000 mesh points. It is also worth noting that despite the fact that GRID does not require the computation of singular vectors, in the context of iterative methods such as ARPACK the extra cost is not significant and PsDM is expected to be far less expensive.

In order to further analyze the accuracy of PsDM, we computed the relative error $|\sigma_{\min}(A - zI) - \delta_j|/\delta_j$ at each computed point z of the (approximate) curves $\partial\Lambda_{\delta_j}(A)$ produced by the algorithm; from these values, we obtained the maximum and mean relative errors for each contour point and show the results in Fig. 7. Note that the best way to read this figure is from right to left, as this shows how the maximum and mean errors per curve develop as we consider the curves from the outermost to the innermost. The maximum relative error appears to be satisfactory except in a few cases, where the maximum error approaches 10^{-1}; even then, however, the mean error remains two orders of magnitude smaller. This highlights one observation we made in our experiments, namely that only a very limited number of points suffer from increased relative error. We also note that there is an apparent increase in the error as we approach very small values of ϵ. This does not reflect a weakness of PsDM, but the difficulty of the underlying SVD method, ARPACK in this case, to compute approximations of the minimum singular value with very small relative error.

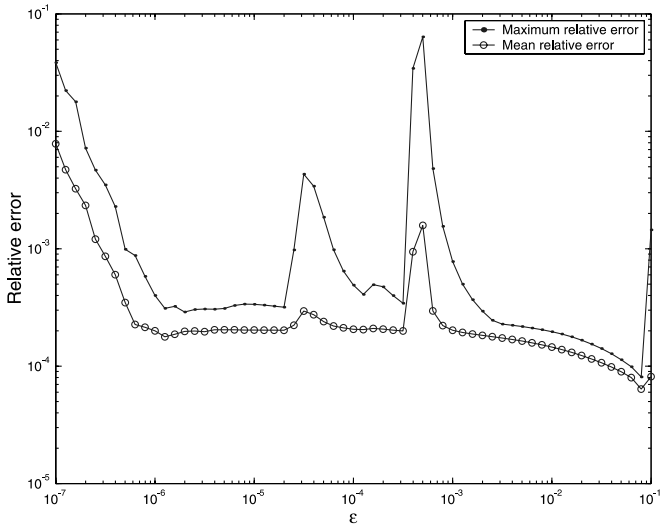

Fig. 7. Maximum and mean relative errors for each curve (60 total) computed via PsDM for $\epsilon = 10^{-1}, \ldots, 10^{-7}$.

4.3. Parallel performance

One important advantage of PsDM is that, by construction, it is embarrassingly parallel: each sweep can be split into a number of tasks equal to the number of points it handles and each task can proceed independently with its work, most of it being triplet computations. Furthermore, assuming that no adaptation is used, no communication is needed between sweeps. Let us assume that this is the case and that the number of points computed in each sweep and therefore the number of tasks is constant, say N. It is thus natural to use static partitioning of tasks, allocating to each of the P processor those tasks handling the sweep for approximately $\lceil N/P \rceil$ points. Table 4 outlines the method. The next question is how to allocate tasks to processors. One natural idea is to use "static block partitioning" and allocate to each processor the computations corresponding to $\lceil N/P \rceil$ consecutive points, say $z_{iL+1}, \ldots, z_{\min((i+1)L,N)}$ for $i = 0, \ldots, P-1$ where $L = \lceil N/P \rceil$.

We applied PsDM on matrix kahan(100) from Section 4.2, starting from $N = 40$ points on the initial curve $\partial\Lambda_{0.1}(A)$, and computed 30 pseudospectrum curves. Therefore, the total number of points that are computed by the end of the run was $1200 = 30 \times 40$. In these and subsequent performance results we did not take into account the time taken by Cobra to approximate the initial curve. Table 5 depicts the corresponding execution times and speedups. The improvements are substantial and the speedups reflect the parallel nature of the algorithm. In order to better understand the effect of the static task allocation policy, we marked which processor would finish first and then examined how much work had been accomplished by then in each of the remaining processors. The results are shown in Table 6, where the number of points that have been completed by the processor that finished first are shown in boldface. Line 1, for instance, shows that processor 0 finished first (the number 600 is in boldface) while, by that time, processor 1 had accomplished the computation of 591 points. Since each processor had to deal with $600 = 1200/2$ points, this meant that processor 1 still had to accomplish work for nine points.

Table 4
Parallel PsDM with static partitioning

(*Input*): N points z_k on the starting curve; P: number of processors.
1. For each processor, select and assign the sweep computations for $\lceil N/P \rceil$ points.
2. Each processor proceeds with PsDM on its own set of points.
3. A marked processor gathers the results of all processors.

Table 5
Performance of parallel implementation of PsDM for kahan(100)

No. of processors	1	2	4	8
Time (s)	264	135	70	36
Speedup	1	1.96	3.77	7.33

Table 6
Number of points computed by each processor by the time the first processor finishes its share of the workload under static block partitioning

Processor(s)	0	1	2	3	4	5	6	7
2	**600**	591	–	–	–	–	–	–
4	299	**300**	296	286	–	–	–	–
8	137	147	**150**	148	148	148	141	145

$P = 2$, 4, 8 processors are used to compute 30 pseudospectra curves for `kahan(100)`. Numbers in boldface denote the number of points computed by the processor that finished first.

As each row shows, the work is reasonably well balanced, thus justifying the good speedups reported in Table 5.

We next applied `PsDM` on matrix `gre_1107` (1107×1107 sparse, real and unsymmetric) from the Harwell-Boeing collection. The initial curve $\partial\Lambda_{0.1}(A)$ was approximated by 64 points computed by `Cobra`. We computed 20 pseudospectrum curves corresponding to $\log_{10}\epsilon = -1.1 : -0.1 : -3$; therefore each processor was allocated $\lceil 64 \times 20/P \rceil$ consecutive points. Rows 3 and 4 of Table 7 depict the times and speedups. Even though computing time is reduced, the speedups are far inferior than those reported for `kahan` in Table 5. To explain this phenomenon, we prepared for `gre_1107` a table similar to Table 6. Results are tabulated in rows 4–6 of Table 8 and reveal severe load imbalance. For example, notice that when using two processors in `PsDM`, (fourth row of Table 8), by the time processor 1 has finished with all its allocated points, processor 0 has finished with only 75% them. Similar patterns hold when using 4 and 8 processors. This load imbalance is due to the varying level of difficulty that an iterative method, such as `ARPACK`, has when computing the triplet corresponding to $\sigma_{\min}(zI - A)$ as z moves from point to point.

Two ways to resolve this problem are (a) a system-level approach to dispatch tasks to the processors from a queue as processors are freed, and/or (b) a problem-level approach in which we estimate the work involved in each task and partition the tasks so as to achieve acceptable load balance. The former approach has the potential for better load balance, at additional system-level overhead; the latter approach has little overhead and is simpler to implement, but appears to require a priori estimates for the workload of each task. Even though this is difficult to know beforehand when using iterative methods, it helps to note that when interested in

Table 7
Performance of parallel implementation of `PsDM` using static block and cyclic partitionings for `gre_1107`

No. of processors	1	2	4	8
Static block partitioning				
Time (s)	6000	3750	2500	1390
Speedup	1	1.6	2.4	4.3
Static cyclic partitioning				
Time (s)	6000	3060	1560	840
Speedup	1	1.96	3.9	7.14

Table 8
As Table 6 for matrix gre_1107 using static block and cyclic partitionings to compute 20 pseudospectrum curves

Processor	Processor id.							
	0	1	2	3	4	5	6	7
Static block partitioning								
2	487	**640**	–	–	–	–	–	–
4	198	293	**320**	319	–	–	–	–
8	83	77	98	159	109	**160**	155	144
Static cyclic partitioning								
2	622	**640**	–	–	–	–	–	–
4	**320**	309	313	309	–	–	–	–
8	**160**	157	152	147	148	157	159	159

load balancing, what is important is not information regarding the amount of time taken by each task, but information regarding work differential between different tasks. Based on this idea, we tried the heuristic that the number of iterations required for a triplet based at z is likely to be similar for neighboring values of z. It is therefore reasonable to allocate points in an interleaved fashion. This leads to "static cyclic partitioning", in which processors $i = 0, \ldots, P-1$ are initialized with the points $z_i, z_{i+P}, z_{i+2P}, \ldots$ Therefore, by interleaving the initial points we aim to shuffle the "difficult" points along the descent of PsDM and distribute them to all the processors. Rows 6 and 7 of Table 7 and rows 8–10 of Table 8 depict the speedups and load distribution obtained from static cyclic partitioning. The improvement in load balance is evident and leads to much better speedups.

We also experimented with "static block cyclic partitioning" in which points were partitioned in $\lceil N/b \rceil$ blocks of b consecutive points each and then the blocks were allocated in a cyclic fashion. We experimented with blocks of size 2 and 4 and found that performance was worse with increasing block size, and was all inferior than the static cyclic case.

We finally show, in Fig. 8, the contours obtained for gre_1107.

4.4. Adapting to decreasing contour lengths

It is known that for a given matrix and varying values of ϵ, the pseudospectrum forms a family of nested sets on the complex plane. Consequently, for any given ϵ and smaller δ, the area of $\Lambda_\delta(A)$ is likely to be smaller than the area of $\Lambda_\epsilon(A)$; similarly, the length of the boundary is likely to change; it would typically become smaller, unless there is separation and creation of disconnected components whose total perimeter exceeds that of the original curve, in which case it might increase. Let us assume now that any contour is approximated by the polygonal path defined by the points computed by PsDM. Then we can readily compute the lengths of the approximating paths. See for example the lengths corresponding to different values of ϵ for matrix kahan in Fig. 6 and the lengths of the corresponding polygonal paths in Table 9 (row 2). It is clear, in this case, that the polygonal path lengths become

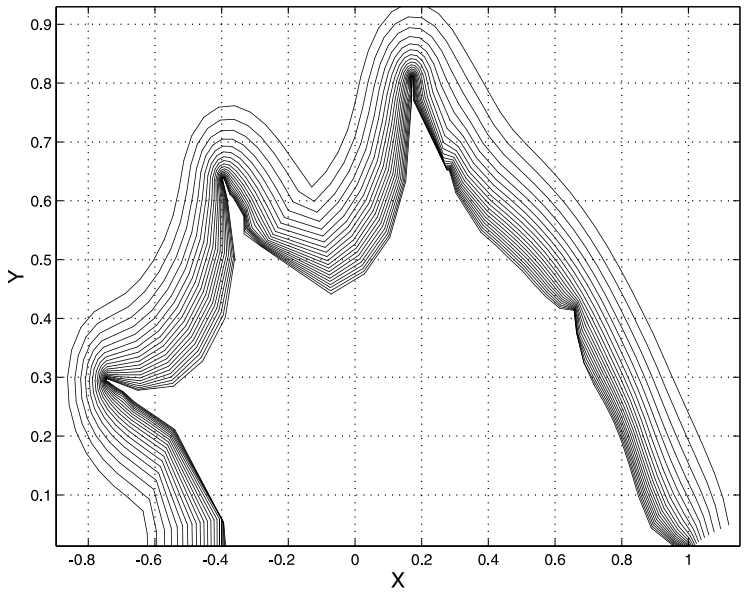

Fig. 8. gre_1107. $\partial\Lambda_\epsilon(A)$ contours for $\log_{10}\epsilon = -1 : -0.1 : -3$.

smaller as we move inwards. Nevertheless, in the PsDM algorithm we have described so far, the same number of points (46) was used to approximate both curves.

It would seem appropriate, then, to monitor any significant length reduction or increase, and adapt accordingly the number of points computed by σPsDM. In the remainder of this section we examine the case of reduced path lengths, noting that similar policies could also be adapted if we needed to increase, rather than reduce, the number of points. The following general scheme describes the transition from $\partial\Lambda_{\delta_m}(A)$ to $\partial\Lambda_{\delta_{m+1}}(A)$. We denote by N_{δ_m} the points defining the contour $\partial\Lambda_{\delta_m}(A)$.

1. Exclude K points of $\partial\Lambda_{\delta_m}(A)$ according to a curve length criterion.
2. Proceed to compute the next contour $\partial\Lambda_{\delta_{m+1}}(A)$, $\delta_{m+1} < \delta_m$, starting from the $N_{\delta_m} - K$ remaining points of $\partial\Lambda_{\delta_m}(A)$.

The scheme allows one to use a variety of curve length criteria, possibly chosen dynamically as PsDM proceeds. In Table 10, we show a single step of σPsDM together

Table 9
Approximate contour lengths of the pseudospectra of kahan(100) and number of points used to approximate each curve following the point reduction policy described in Section 4.3

$\log_{10}\epsilon$	−1	−2	−3	−4	−5	−6	−7
Approx. lengths	4.48	2.63	1.85	1.48	1.29	1.18	1.1
No. of points	46	29	24	22	22	21	20

Table 10
Single sweep of PsDM with adaptive point reduction

(**Input**):
  l_m: Min distance between consecutive points on $\partial\Lambda_{\delta_{m-1}}(A)$;
  points $\{z_k \in \partial\Lambda_{\delta_m}(A),\ k = 1,\ldots,N_{\delta_m}\}$.
(**Output**):
  l_{m+1}: Min distance between consecutive points on $\partial\Lambda_{\delta_m}(A)$;
  points $\{w_k \in \partial\Lambda_{\delta_{m+1}}(A),\ k = 1,\ldots,N_{\delta_{m+1}}\}$, where $N_{\delta_{m+1}} \leqslant N_{\delta_m}$.
1. **for** $k = 1 : N_{\delta_m} - 1$
  $d_k = |z_k - z_{k+1}|$.
  end
2. **for** $k = 1 : N_{\delta_m} - 2$
  $\mu_k = (d_k + d_{k+1}) < 2l_m$.
  end
3. $i = 1$, $\tilde{z}_1 = z_1$, $\mu_0 = 0$, $\mu_{N_{\delta_{m-1}}} = 0$, $\pi_1 = 0$.
4. **for** $k = 2 : N_{\delta_m} - 2$
  4.1 $\pi_k = (\mathrm{NOT}\pi_{k-1}) \cdot \mu_{k-1}$.
  4.2 **if** $(\pi_k = 0)$ **then** $\tilde{z}_i = z_k$ **else** $\tilde{z}_i = z_{k+1}$ **end**
  4.3 $i = i + 1$.
  end
5. $N_{\delta_{m+1}} = i$, $\tilde{z}_{N_{\delta_{m+1}}} = z_{N_{\delta_m}}$.
6. $l_{m+1} = \min\{|\tilde{z}_i - \tilde{z}_{i+1}| \text{ for } 1 \leqslant i \leqslant N_{\delta_m} - 1\}$.
7. Apply single sweep of σPsDM on $\{\tilde{z}_i \in \partial\Lambda_{\delta_m}(A),\ i = 1,\ldots,N_{\delta_{m+1}}\}$
  to compute $\{w_i \in \partial\Lambda_{\delta_{m+1}}(A),\ i = 1,\ldots,N_{\delta_{m+1}}\}$.

with the implementation of such a strategy, based on the variation of the minimum distance between consecutive points on subsequent contours. Steps 1–6 implement the point reduction and also compute the next minimum distance, l_{m+1}, between consecutive points of $\partial\Lambda_{\delta_m}(A)$, while step 7 is the standard sweep described in Table 2. First, the algorithm computes the distances $d_k = |z_k - z_{k+1}|$ between consecutive points on $\partial\Lambda_{\delta_m}(A)$, and then drops any point z_k from the curve if it finds that $d_k + d_{k+1} < 2l_m$, unless the previous point z_{k-1} had already been dropped. In the algorithm presented in Table 10, this test is implemented by means of the boolean variables μ_k, π_k and a boolean recurrence specified in line 4.1. Our implementation keeps fixed the first and last points on $\partial\Lambda_{\delta_m}(A)$, though this is easily modified. We also take advantage of the fact that A is real so that we need to compute only one half of the curve.

Fig. 9 illustrates the underlying idea. In the top curve, all points, z_k, inside a box or circle are candidates for dropping, because they satisfy $|z_{k-1} - z_k| + |z_k - z_{k+1}| < 2l_m$. However, only z_k, $k = 4$, 7, 11, 13 (inside circle), are dropped because they are the only ones for which the preceding point, z_{k-1}, was not dropped. The remaining points are shown in the bottom curve, renumbered to indicate their relative position within the curve.

We implemented PsDM with the point reduction policy described in Table 10 on our parallel platform. We specifically required the parallel version to produce the same points that would have been produced, had the above policy run serially. To achieve this, processors synchronize so that the sweep outlined in Table 10 starts

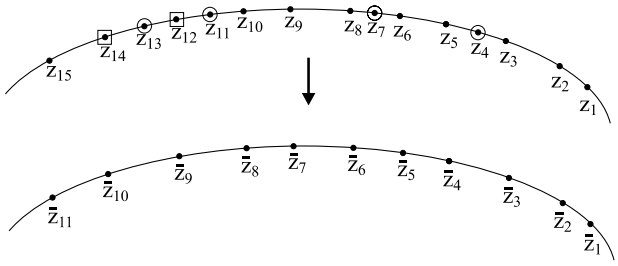

Fig. 9. Adaptive point reduction: inscribed points are candidates for dropping but only the encircled ones drop successfully.

only when all input data are available to all. Due to the low complexity of this procedure relative to the remaining work, we decided to use a simple approach in which steps 1–6 are executed by a single, master, processor instead of a more complicated parallel reduction policy. In particular, processors send back points to the master processor, who applies point reduction. The new points are then ready for allocation to the processors and the application of σ`PsDM` (step 7 of Table 10). The question arises then as to how to allocate these points. As before, we could use a queue or a static approach. Given the success of the latter in the previous experiments, and its lack of overhead, we experimented with the static approaches described earlier, assigning approximately $\lceil N_{\delta_{m+1}}/p \rceil$ points to each processor.

We first applied this adaptive version of `PsDM` on `kahan(100)`. This almost halved the number of triplet evaluations, reducing them from 2760 to 1466. Table 9 (third row) depicts how the number of points varied for selected values of ϵ. The contours computed using this strategy are depicted in Fig. 10 vs. the same contours computed using `PsDM` but holding the number of points per contour constant. It is clear that adaptation is very effective without visibly affecting the quality of the output. Columns 2 and 3 of Table 11 depict the execution times and speedups (in parentheses). Once again, static cyclic is superior to block assignment. Furthermore, the corresponding speedups are similar, though, naturally, not as high as `PsDM` without point reduction (cf. Table 5).

We also performed the same experiment on matrix `gre_1107` and show the times and corresponding speedups in columns 4 and 5 of Table 11. We started with 74 points on $\partial\Lambda_{0.1}(A)$ and computed 20 curves as deep as $\partial\Lambda_{0.001}(A)$, where we concluded with 60 points. The total number of triplet evaluations was reduced to 1300 compared to $20 \times 74 = 1480$ that would have been required if no point reduction had been performed. In this case too, cyclic allocation performs better than block. Furthermore, the achieved speedups are satisfactory, as they are very close to those reported for the no-drop policy in rows 8–10 of Table 8.

The above discussion shows that `PsDM` can be enhanced to adapt to the geometric features of the contours, in which case there is a corresponding reduction in cost as we move inwards. It is clear that this idea can serve as a springboard for the design of alternative adaptation strategies.

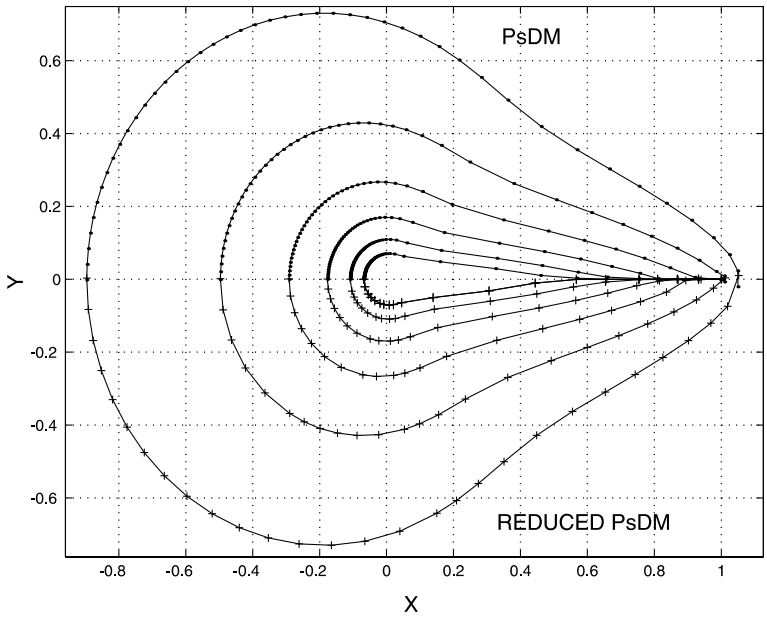

Fig. 10. Pseudospectra contours, $\log_{10}\epsilon = -2 : -1 : -7$, for `kahan(100)`. Top: normal `PsDM`; bottom: `PsDM` with point reduction.

4.5. Capturing disconnected components

One documented weakness of path following methods is that they cannot readily capture disconnected components of the pseudospectrum. One question is how does `PsDM` handle this difficulty? In this section we show that `PsDM` manages to trace disconnected curves, as long as they lie inside the initial curve $\partial\Lambda_\epsilon(A)$. In that respect, the performance is the same as that of `GRID`, where the level curves traced are only those that lie within the area discretized by Ω_h. Consider matrix `grcar(50)` from the Test Matrix Toolbox. We begin from 192 points on $\partial\Lambda_{10^{-3}}(A)$ and proceed with a fixed step 0.1 to compute 30 curves. Thus the inner curve is $\partial\Lambda_{10^{-6}}(A)$. The pseudo-

Table 11
Execution times in seconds and speedups (in parentheses) for parallel `PsDM` with point reduction for matrices `kahan(100)` (left) and `gre_1107` (right)

	`kahan`		`gre_1107`	
	Static block	Static cyclic	Static block	Static cyclic
1	364	–	6437	–
2	189 (1.93)	181 (2.01)	4113 (1.81)	3258 (1.96)
4	115 (3.17)	94 (3.87)	2564 (3.02)	1685 (3.82)
8	60 (6.06)	58 (6.28)	1526 (5.3)	910 (7.1)

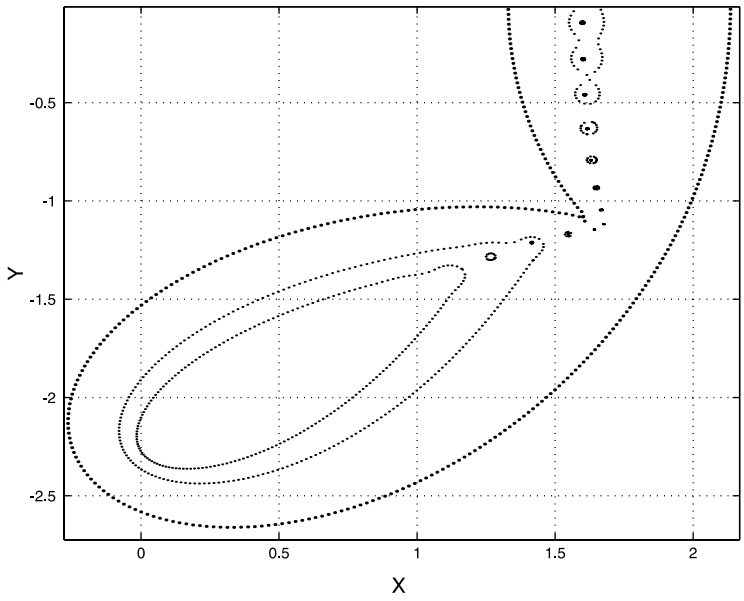

Fig. 11. Capturing disconnected components for `grcar(50)`. Curves from outer to inner: $\epsilon = 10^{-3},\ 10^{-5},\ 10^{-6}$.

spectra for $\epsilon = 10^{-5},\ 10^{-6}$ have disconnected components. Fig. 11 demonstrates that `PsDM` manages to retrieve this intricate structure. Of course this procedure does not return components that were already outside the initial $\partial\Lambda_\epsilon(A)$ curve. If we want to assure that all components are captured, we can start with an initial curve that is large enough using techniques such as those presented in [4].

5. Conclusions

We presented `PsDM`, a new method for the computation of the pseudospectrum of a matrix A that combines the appealing characteristics of the traditional method `GRID` and the versatility of path following methods. We saw that `PsDM` automatically generates new curves starting, for instance, from one application of `PF`. The method is such that the curves adapt to the geometric properties of the pseudospectrum and is able to capture disconnected components. Experimental results showed that the method achieves a significant reduction of the number of necessary triplet evaluations vs. `GRID` even though, like `GRID`, it also computes the pseudospectrum for several values of ϵ. Given the parallel nature of `PsDM`, we implemented it using `MPI`. These experiments also revealed the advantages of a simple heuristic designed to achieve load balance. An OpenMP implementation of `PsDM` is currently underway and is expected to serve as a platform for the investigation of a variety of alternative policies. It is worth noting that `PsDM` computes successively, using path following

but in the direction of steepest descent, points defining the nested curves $\partial\Lambda_\epsilon(A)$. This idea holds great promise for pseudospectra and eigenvalues as described in [3,9,10]. Overall, we believe that the approach used in PsDM will be useful in the construction of an adaptive algorithm for the computation of pseudospectra that will be based on path following. We note that the codes used in this paper are available from URL http://www.hpclab.ceid.upatras.gr/scgroup/pseudospectra.html.

Acknowledgements

We thank Bernard Philippe for his astute comments regarding the paper. We also thank the referees for their valuable suggestions that helped improve the paper, as well as our colleague, Efi Kokiopoulou, for her comments and support.

References

[1] E.L. Allgower, K. Georg, in: Numerical Continuation Methods: An Introduction, Computational Mathematics, vol. 13, Springer, Berlin, 1990.
[2] C. Bekas, E. Gallopoulos, Cobra: Parallel path following for computing the matrix pseudospectrum, Parallel Computing 27 (14) (2001) 1879–1896. Available from www.hpclab.ceid.upatras.gr in/faculty/stratis/Papers.
[3] C. Bekas, I. Koutis, Parallel algorithms for the computation of pseudospectra, Master's Thesis, Computer Engineering and Informatics Department, University of Patras, June 1998 (in Greek).
[4] T. Braconnier, R.A. McCoy, V. Toumazou, Using the field of values for pseudospectra generation, Technical Report TR/PA/97/28, CERFACS, Toulouse, September 1997.
[5] M. Brühl, A curve tracing algorithm for computing the pseudospectrum, BIT 33 (3) (1996) 441–445.
[6] L. Elsner, C. He, An algorithm for computing the distance to uncontrollability, Systems & Control Letters 17 (1991) 453–464.
[7] N.J. Higham, The Test Matrix Toolbox for MATLAB (version 3.0), Technical Report 276, Manchester Centre for Computational Mathematics, September 1995.
[8] V.I. Kostin, On definition of matrices' spectra, in: M. DurandF. El Dabaghi (Eds.), High Performance Computing II, North-Holland, Amsterdam, 1991, pp. 407–414.
[9] I. Koutis, E. Gallopoulos, Iterations on domains for computing the matrix (pseudo) spectrum, in: Foundation of Computational Mathematics Conference (FOCM), University of Oxford, July 1999.
[10] I. Koutis, E. Gallopoulos, Hermitian methods for computing eigenvalues, 2000 (in preparation).
[11] R. Lehoucq, D.C. Sorensen, C. Yang, Arpack User's Guide: Solution of Large-scale Eigenvalue Problems With Implicitly Restarted Arnoldi Methods, SIAM, Philadelphia, 1998.
[12] Y. Levin, A. Ben-Israel, Directional Newton methods in n variables, Technical Report 2000-13, DIMACS, May 2000.
[13] D. Mezher, B. Philippe, PAT – a reliable path following algorithm, August 2000 (submitted).
[14] Y. Saad, SPARSKIT: A basic tool-kit for sparse matrix computations (version 2), Technical Report, University of Minnesota, 1994.
[15] L.N. Trefethen, Computation of pseudospectra, in: Acta Numerica 1999, vol. 8, Cambridge University Press, Cambridge, 1999, pp. 247–295.