Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Future Generation Computer Systems xxx (2004) xxx–xxx

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.com/locate/future

The design of a distributed MATLAB-based environment for computing pseudospectra

C. Bekas [a], E. Kokiopoulou [a], E. Gallopoulos [b,*]

[a] *Computer Science and Engineering Department, University of Minnesota, Minneapolis, MN USA*
[b] *Computer Engineering and Informatics Department, University of Patras, Patras, Greece*

Received 24 September 2003; received in revised form 30 October 2003

Abstract

It has been documented in the literature that the pseudospectrum of a matrix is a powerful concept that broadens our understanding of phenomena based on matrix computations. When the matrix A is non-normal, however, the computation of the pseudospectrum becomes a very expensive computational task. Thus, the use of high performance computing resources becomes key to obtaining useful answers in acceptable amounts of time. In this work we describe the design and implementation of an environment that integrates a suite of state-of-the-art algorithms running on a cluster of workstations to enable the matrix pseudospectrum become a practical tool for scientists and engineers. The user interacts with the environment via the graphical user interface `PPsGUI`. The environment is constructed on top of `CMTM`, an existing environment that enables distributed computation via an MPI API for `MATLAB`.

Keywords: Eigenvalues; Pseudospectrum; Problem solving environments; Parallel MATLAB; MPI; CMTM; Grid computing

1. Introduction and motivation

The ϵ-pseudospectrum, $\Lambda_\epsilon(A)$, of a matrix (pseudospectrum for short) describes the locus of eigenvalues of $A + E$ for all possible E such that $\|E\| \leq \epsilon$ for some matrix norm and given ϵ, that is

$$\Lambda_\epsilon(A) = \{z \in \mathbb{C} : z \in \Lambda(A + E), \|E\| \leq \epsilon\}, \qquad (1)$$

where $\Lambda(A)$ denotes the set of eigenvalues of matrix A. For the remainder of this paper we will assume that the underlying matrix norm bounding the perturbations is the two-norm. The ϵ-pseudospectra are regions of the complex plane that show where the eigenvalues of a matrix could go when the matrix is subjected to perturbations. They have many interesting properties and can provide useful information regarding the behavior of iterative methods for the major problems of numerical linear algebra as well for stability studies that frequently are more informative than the eigenvalues of the underlying matrix A (e.g. [15,37–39]). It is thus argued that pseudospectra calculations are a necessary component in any suite of tools that returns qualitative information about a matrix computation. As a result, it becomes of interest to build a tool that would compute pseudospectra, possibly together with other information about the matrix (e.g. condition numbers, indices of eigenvalue sensitivity, etc.) Unfortunately, it is well known that in many cases of interest the computation of the pseudospectrum

* Corresponding author.
E-mail addresses: bekas@cs.umn.edu (C. Bekas), kokiopou@cs.umn.edu (E. Kokiopoulou), stratis@ceid.upatras.gr (E. Gallopoulos).

0167-739X/$ – see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2003.12.017

becomes a difficult task because of its high computational cost. For example, the application of definition (1) would entail a cost that would be a multiple of the computation of all the eigenvalues of the matrix, a daunting task for large matrices. As a result, there have been several research efforts leading to a variety of methods that researchers actively deploy to approximate pseudospectra ranging from simple applications of (1) or its equivalents (2, 3 presented in the next section) (e.g. see [22] for some codes), to state-of-the-art polyalgorithms (cf. [8,38]).

In this work we outline the design and implementation of a system designed to provide all the computational facilities to compute matrix pseudospectra. The system is built to be flexible so as to permit the incorporation of novel algorithms and is deployed via `PPsGUI`, a powerful graphical user interface. The system integrates a suite of state-of-the-art parallel algorithms running in a distributed mode to facilitate scientists and engineers mine pseudospectral information from any matrix at a cost that is as little as possible without specific knowledge of the algorithms underlying its computation. Related work includes the graphical tool presented in [29] as well as the popular and continuously evolving `eigtool`; cf. [40–42]. These tools implement a limited combination of matrix-based and domain-based algorithms. To these, our environment contributes a flexible combination of parallel "domain" and "matrix-based" approaches (cf. Section 2) that permit the fast and convenient computation of the pseudospectrum of large matrices. Indeed, the work presented herein is the first effort to combine a wide variety of methods from both of the above categories in a single tool. Our programming platform is `MATLAB` that provides an open environment equipped with sophisticated visualization capabilities. Parallelism is supported by means of the Message Passing Interface paradigm (`MPI`). This choice is based on the specifications of `CMTM`, a software module that provides `MPI` functionality for `MATLAB` and thus allows the concurrent execution of `MATLAB` processes on clusters of workstations (COWs) [43].

The remainder of this paper is organized as follows. In Section 2 we present some examples of pseudospectra, `GRID`—a simple classical algorithm for their computation and a classification of methods. In Section 3 we outline three domain-based algorithms, a matrix-based methodology and their hybrids. In Section 4 we describe the cluster architecture and parallel programming with `CMTM` in `MATLAB`. Section 5 describes the GUI front-end of the environment and the applicability of the proposed tool in the Grid framework. Finally, Section 6 contains concluding remarks.

2. Computing pseudospectra

A first example of pseudospectra is presented in Fig. 1. We use definition (1) to estimate $\Lambda_\epsilon(A)$ for a matrix that is routinely used as a benchmark; this is matrix `kahan` (see [22] for the generating `MATLAB` code) of size $n = 100$ when the norm of E is bounded by different values of $\epsilon = 10^{-3}, 10^{-7}$ and 10^{-11}. To this end, we computed the eigenvalues of random perturbations $A + E$ where for each value of ϵ we conducted 100 experiments. Then, for each ϵ we created a plot where we superimpose the "pseudoeigenvalues" (eigenvalues of $A + E$) and the original eigenvalues of A. It readily follows from the definition that for

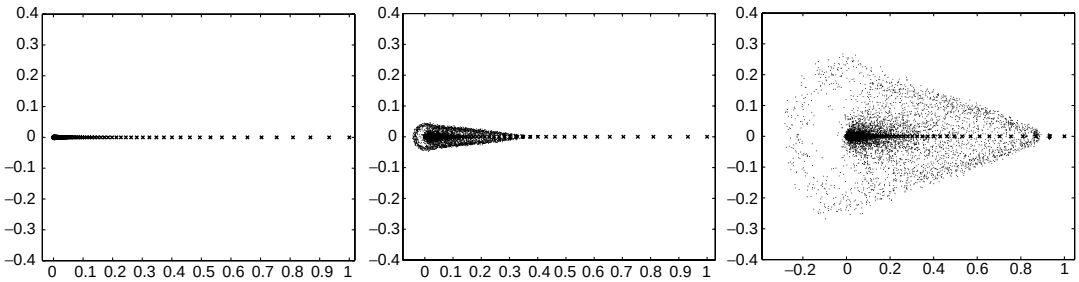

Fig. 1. Estimating $\Lambda_\epsilon(A)$ for matrix `kahan(100)` by random bounded perturbations $A + E$, where $\|E\| \leq \epsilon, \epsilon = 10^{-11}, 10^{-7}$ and 10^{-3} (left to right). Dots denote "pseudoeigenvalues" while crosses denote the eigenvalues of A.

different values of ϵ, the ϵ-pseudospectra form nested point sets that "collapse" to the spectrum of A as ϵ tends to zero. The figures indicate that the smaller eigenvalues of A are more sensitive to perturbations. When A is normal (i.e. satisfies the relation $AA^* = A^*A$), the pseudospectrum is readily computed from the eigenvalues since classical matrix theory predicts that it will be the union of the disks of radius ϵ surrounding each eigenvalue of A. The pseudospectrum becomes of interest on its own or as an alternative to standard eigenvalue analysis in cases such as the above, where A is non-normal (e.g. nonsymmetric). Even for medium size matrices, however, the computation of their pseudospectrum is a difficult task [38]. Regarding definition (1) for example, even if we perform a large number, say s, of random perturbations, we cannot guarantee that they will capture the extreme dislocations that are possible for the eigenvalues under all possible ϵ-bounded perturbations. Furthermore, as already mentioned earlier, the cost of a pseudospectrum approximation method based on definition (1) is at least equal to the cost of computing all the eigenvalues of s matrices of the form $A + E$. One way to get round this problem is by means of two alternative definitions of $\Lambda_\epsilon(A)$ that can be shown to be equivalent to (1) [38]:

$$\Lambda_\epsilon(A) = \{z \in \mathbb{C} : \|(A - zI)^{-1}\| \geq \epsilon^{-1}\}, \tag{2}$$

$$\Lambda_\epsilon(A) = \{z \in \mathbb{C} : \sigma_{\min}(A - zI) \leq \epsilon\}, \tag{3}$$

where $\sigma_{\min}$ denotes the smallest singular value of its matrix argument. The standard algorithm (`GRID`) for the computation of $\Lambda_\epsilon(A)$ consists of three phases that are presented in Algorithm (1).

Algorithm (`GRID`).

1. Let Ω be such that $\Omega \supseteq \Lambda_\epsilon(A)$ for the largest ϵ of interest and Ω_h be a discretization of Ω with gridpoints z_k.
2. Compute $s(z_k) := \sigma_{\min}(z_k I - A)\, \forall z_k \in \Omega_h$.
3. Plot the contours of $s(z_k)$ for all values of ϵ of interest to obtain $\partial\Lambda_\epsilon(A)$.

Therefore, if we know Ω (an interesting issue in its own right that can be addressed by special algorithms, e.g. see [8,10]), the cost to decide if z is a member of $\Lambda_\epsilon(A)$ or not amounts to the cost of computing $s(z)$. Fig. 2 depicts the pseudospectra boundaries $\partial\Lambda_\epsilon(A)$ of `kahan(100)` for $\epsilon = 10^{-k}, k = 3, \ldots, , 11$. Two important features of `GRID` are its straightforward simplicity, robustness and potential for embarrassingly parallel implementation. Its cost is typically modeled by

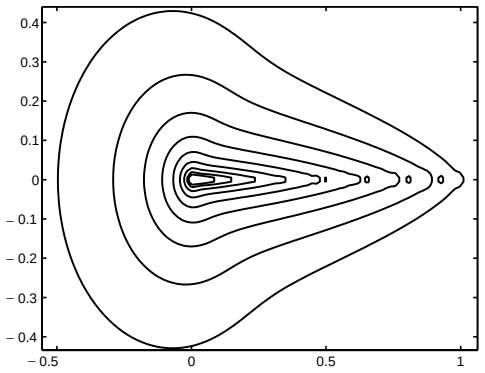

Fig. 2. The pseudospectra boundaries $\partial\Lambda_\epsilon(A)$ of matrix `kahan(100)` for $\epsilon = 10^{-k}, k = 3, \ldots, , 11$.

$$C_{\texttt{GRID}} = |\Omega_h| \times C_{\sigma_{\min}}, \tag{4}$$

where $|\Omega_h|$ denotes the number of nodes of Ω_h and $C_{\sigma_{\min}}$ is a measure of the average cost for the computation of $s(z)$. The total cost quickly becomes prohibitive as the size of A and/or the number of nodes increase. Given that the cost of computing $s(z)$ is at least $O(n^2)$ and that a typical mesh could easily contain $O(10^4)$ points, the cost can be very high even for matrices of moderate size. For example computing the pseudospectrum of a dense matrix of size $n = 1000$ on a Pentium III workstation using a 50×50 mesh and `MATLAB` 6.1 requires more than a day. Cost formula (4) readily indicates two major classes of methods for accelerating the computation, based on the mathematics and numerics of the problem: (a) domain-based methods, that aim to reduce the number of nodes where $s(z)$ is computed, and (b) matrix-based methods that focus on reducing the cost for the evaluation of $s(z)$. The above methods can be combined with system-level approaches, such as the exploitation of hierarchical memory and parallel processing. See [38,42] for informative surveys of recent efforts in the area and the *Pseudospectra Gateway* site [31] for a comprehensive repository of links to research efforts, references and software.

3. Parallel algorithms for the pseudospectrum

The algorithms driving our environment draw from a number of domain as well as matrix-based methods that were developed in recent years that lend themselves to parallel computation.

3.1. Domain-based methods

The three major domain-based components of our environment are the following (the numbers in bold brackets denote the reference where extended discussion about the method can be found.)

- `Cobra` [4]: Parallel path following method that numerically traces a single boundary of $\Lambda_\epsilon(A)$ by concurrent estimation, using Newton iteration to solve $\sigma_{\min}(A-zI)-\epsilon = 0$, of m points z_k^i, $i = 1, \ldots, m \in \partial\Lambda_\epsilon(A)$. The method inherits the flexibility of numerical continuation methods (see [1,11]), while remedying a number of weaknesses such as danger of breakdown at steep turns of $\partial\Lambda_\epsilon(A)$ and lack of large grain parallelism.
- `PsDM` [5]: Using the points $z_k \in \partial\Lambda_\epsilon(A), k = 1, \ldots, N$ that are already available (i.e. computed by `Cobra`), the method proceeds to estimate an inner boundary $\partial\Lambda_\delta(A)$, $\delta < \epsilon$, by concurrent Newton corrections towards the points $y_k \in \partial\Lambda_\delta(A), k = 1, \ldots, N$. The method combines the flexibility of path following methods with the robustness of `GRID`, while preserving the opportunities for large grain parallelism of the latter.
- `PMoG` [8]: Parallel variant of `GRID` that drastically reduces the cost $C_{\texttt{GRID}}$ by quickly excluding points z_h of the mesh Ω_h that are not part of $\Lambda_\epsilon(A)$. The method can be used to quickly estimate the region of the complex plane that contains the pseudospectrum. If needed, it can also provide approximations to $\partial\Lambda_\epsilon(A)$, at very low cost for coarse resolutions.

Fig. 3 depicts instances of the above methods. Each of the aforementioned parallel domain-based methods has a different intuitive geometrical interpretation. `Cobra` generates a single (or more) $\partial\Lambda_\epsilon(A)$ at a time. `PsDM` generates nested $\partial\Lambda_\epsilon(A)$. `PMoG` is a generalization of `GRID` that is based on fast exclusions. Therefore, it is natural to combine the above approaches in order to exploit all of their features. `PMoG` provides a very effective scheme for deciding where on the complex plane to concentrate our efforts. The data resulting from `PMoG` can be used as input points for `PsDM`. In particular, one can quickly come close to the pseudospectrum boundary $\partial\Lambda_\epsilon(A)$ by means of

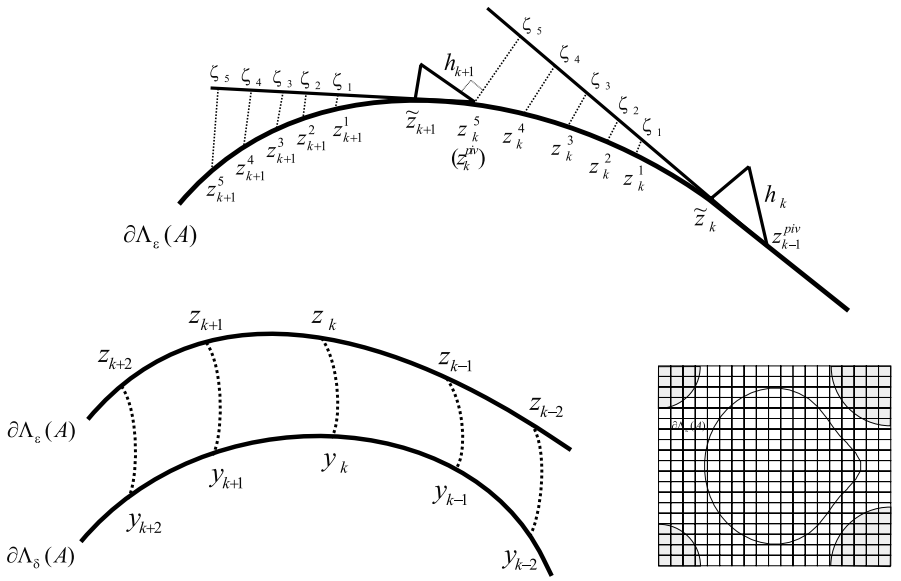

Fig. 3. Instances of parallel domain-based methods for computing matrix pseudospectra. Top: `Cobra`. Bottom: `PsDM` (left) and `PMoG` (right).

PMoG and then use PsDM to obtain it at greater detail. Frequently, one is interested in observing how do the $\partial\Lambda_\epsilon(A)$ pseudospectral boundaries vary with ϵ. To do this, one can first apply Cobra to compute a pseudospectral boundary for a single ϵ and then estimate additional boundaries corresponding to smaller ϵ using PsDM (see Section 5. As already mentioned, all three methods lend themselves to parallel computation. Cobra is the most limiting of the three because it allows the concurrent computation of a small number of points defining a given pseudospectral boundary at a time. Thus we have a fixed amount of parallelism per step and the algorithm cannot scale on geometrical considerations alone; cf. [4]. PsDM allows the concurrent computation of all points of a new pseudospectral curve from a current one in a "level set"-like manner. Finally, PMoG is closely related to GRID and allows the concurrent computation at any points of Ω_h, except that the algorithm also leads to the fast exclusion of points of Ω_h.

3.2. Matrix-based methods and hybrids

In spite of the significant improvements in computational savings that have been demonstrated in the literature via domain-based methods, they are not sufficient for the fast computation of pseudospectra of large matrices. Even in GRID, the cost is linear in the number of gridpoints but at least quadratic in the size of the matrix. In fact, the computation of $s(z)$, even by means of advanced methods (e.g. those presented in [2,23,25,26]), is very expensive, especially when the smallest singular values are clustered. Moreover, we need to compute $s(z)$ for many values of z. Unfortunately, however, the singular values of $A - z_k I$ are not readily obtained by shifting the singular values of A; that is, in general, $\sigma_{\min}(A - z_k I) \neq \sigma_{\min}(A) - z_k$ even though the set of eigenvalues $\Lambda(A - z_k I) = \Lambda(A) - z_k$. This is the root of the domain-based complexity of pseudospectra: Standard dense or iterative methods for computing $s(z)$ require a different run for each z_k. This is also the reason that GRID is embarrassingly parallel and apparently well suited for a distributed or even Grid-like implementation. What we would like to derive is a method that concurrently provides accurate approximations of $s(z_k)$ for a large number of values z_k. This would enable the creation of powerful hybrid algorithms based on the aforementioned domain approaches. One early method in that direction was described in [36]. This method contains two major phases: The first is an (expensive) computation that is common to all values z_k while the second consists of cheaper computations that are independent for each z_k. An alternative approach that appears to return more accurate approximations is based on a "transfer function" framework [7,34]. This framework is based on definition (2) and an approximation of $\Lambda_\epsilon(A)$ by $\Lambda_\epsilon(G_{z,m}(A))$, where

$$G_{z,m}(A) = V_{m+1}^*(A - zI)^{-1}V_{m+1}. \tag{5}$$

Here, V_{m+1} denotes an orthogonal basis for the Krylov subspace $\mathcal{K}_m(A, v_1)$ computed by the Arnoldi method [33]. We call this "transfer function" framework inspired by the term used in Control for functions like $G_{z,m}(A)$. In [3,7,34] it was shown that $\|G_{z_h,m}(A)\|$ can be efficiently computed iteratively for a large number of shifts z_k using Krylov linear solvers such as GMRES and restarted FOM (see [33]) at an additional cost of $O(m^3)$ for each shift compared to a standard singular value solver that computes $s(z_k)$. Since typically m is much smaller than the size of the matrix we achieve substantial computational savings.

As a natural next step we developed hybrid algorithms that combine the domain methods of the previous section with the transfer function framework. The resulting algorithms permit us to address a variety of problems with very large matrices. In particular, GRID-like hybrids with PMoG extensions are ideal when we are interested in the complete pseudospectrum $\Lambda_\epsilon(A)$. On the other hand, hybrids based on Cobra and PsDM are particularly suitable when we are interested only in local behavior of the pseudospectrum, for example near the imaginary axis in applications concerning stability. In Table 1 we illustrate the expected performance of the transfer function approach on a Pentium III workstation with 1GB of RAM as matrix sizes scale.

Table 1
Typical runtimes on a Pentium III workstation with 1GB RAM using state-of-the-art hybrid pseudospectra computing algorithms

	Matrix size			
	10^3	10^4	10^5	10^6
Runtimes	1–2 min	5–10 min	2–3 h	8–16 h

4. Computational platforms and issues

Given the high computational complexity of pseudospectra computations for large matrix sizes and resolutions, it becomes necessary to use high performance computing (HPC) resources. At the level of the hardware infrastructure, to render pseudospectra practical for the common user, it would be preferable to use off-the-shelf systems such as clusters or networks of workstations that have recently become a particularly appealing, low-cost solution to supercomputing as well as a component element to more distributed, Grid-type configurations. In fact, network-based computing is rapidly becoming a dominant paradigm for Computational Science and Engineering. All algorithms described in Section 3 and incorporated in our environment were implemented to run on a COW used as an integrated computing resource. The actual configuration used for most of our experiments consisted of eight nodes of Pentium III PCs running Windows 2000 connected via a 100 Mb fast Ethernet.

On the algorithmic side, it is expected that a successful approach would combine several of the above methods into a polyalgorithm [32]. In particular, we seek an open software environment integrating and interconnecting available algorithms, providing transparent access to parallel architectures and at the same time offering visualization features. Moreover, in addition to being fast and accurate, a practical system would also need to be user-friendly. The algorithms described in Section 3 demonstrate parallelism both at the level of the independent computations that occur in the domain as well as at the level of the parallelism that is possible within the computation of the singular triplets (meaning the triplets $(\sigma_{\min}, u_{\min}, v_{\min})$ of minimum singular value and left and right singular vectors); cf. Section 3 as well as the original references for details.

An important issue in the design of parallel and distributed algorithms is load balancing. For the algorithms outlined above, we were mostly concerned with the specific issue of distributing the computational load equally amongst the processors of a dedicated COW in order to achieve high parallel efficiency and speedup. Consider for example the GRID algorithm which requires one to compute $s(z)$ (and possibly the corresponding left and right singular vectors) for every mesh point $z_k \in \Omega_h$. If we were to use the intrinsic MATLAB's svd function, the runtime for $s(z)$ would be almost independent of the value of z (except for the few occasions that z is on the real line), so load balancing is not an issue. Frequently, however, the problem is large and sparse so to solve it we need to apply sparse iterative techniques, e.g. the intrinsic MATLAB svds function that is based on ARPACK [27]. In that case, performance becomes sensitive to the value of z and runtimes can vary significantly. Therefore, a load balancing scheme becomes necessary if we are to achieve high efficiency on a dedicated COW. One possible approach when applying iterative methods in the context of GRID is to use system-level information obtained during execution to distribute the computational load according to resource availability. Using a queue, for instance, could ensure fully dynamic load distribution at the cost of increased network traffic. An alternative approach is to attempt to exploit the numerics of the problem. It has been observed, for example, that nearby points, say z and z', typically require similar times to compute $s(z)$ and $s(z')$. Therefore, a careful interleaved distribution of the points in Ω_h across the processors could lead to better load balance. Similar load balancing issues arise in PsDM and PMoG; cf. [8] for a detailed description of some load balancing policies for the algorithms described in Section 3.

Parallelism and communication for our system running on the COW are based on the message passing paradigm, and in particular MPI [30]. MPI addresses concepts and constructs such as point-to-point message passing, collective communications, process groups and topologies. There exist several commercial and free implementations of MPI. We used MPI/Pro [35], as this is currently the only system supporting the MPI API for CMTM that is a major enabling component of our environment.

4.1. MATLAB and the Cornell multitask toolbox (CMTM)

MATLAB has evolved into a powerful domain-specific Problem Solving Environment (PSE) [24] for the development and rapid prototyping of numerical applications. The key ingredient to its success is that interaction with MATLAB is in linear algebra terms and Linear Algebra is admittedly a principal tool for scientific computing. MATLAB's programming language is relatively simple and frees the application

developers from a host of programming details, allowing them to focus on the problem to solve. MATLAB was originally conceived and designed as a user-friendly interface to the LINPACK and EISPACK libraries but today incorporates and provides access to sophisticated implementations of a large number of state-of-the-art numerical methods that address a wide range of problems. MATLAB offers sophisticated visualization routines that are necessary for an application like pseudospectra. Because of its open environment, its large user community contributes new codes and other software to public repositories like Netlib.

A common criticism of high-level rapid prototyping environments, like MATLAB, is that ease of use comes at the cost of computational performance. As stated in [13], for example, "Environments like MATLAB and Mathematica have proven to be especially attractive for rapid prototyping of new algorithms and systems that may subsequently be implemented in a more customized manner for higher performance". This has motivated several efforts to enhance the performance of these environments. In the case of MATLAB these efforts include the incorporation of "cache aware" linear algebra primitives based on BLAS-3 and LAPACK as well as kernels based on novel architecture-dependent tuning such as ATLAS [14] and FFTW [19]; use of JIT compiler technology for the rapid execution of MATLAB code [28]; enabling the linking of subroutines produced from C or Fortran source code; compilers to convert MATLAB applications into stand-alone C or C++ code. Source-to-source compilation from MATLAB to a high-level language such as C or Fortran with mature compiler technology on parallel systems is one of several possible approaches for enabling MATLAB for parallel processing. There exist several others, nicely surveyed in [12]. One of the most popular approaches, judging by the number of efforts, is to engage COW processors in concurrent MATLAB sessions and provide access to a message passing library. This approach is followed in the Cornell Multitask Toolbox (CMTM) [43]. CMTM was developed at the Cornell Theory Center and offers an MPI API for MATLAB consisting of approximately 40 commands. Once the master MATLAB process has been initialized, additional MATLAB processes are invoked by issuing the command where np is the total number of processes. Variable mm_er returns any error codes. The master process can kill all others via the command CMTM provides facilities to write programs under the SIMD as well as the MIMD paradigms. In particular, by invoking on the master, process dest executes the MATLAB string command. In case dest is omitted, command is executed by all processes. As in MATLAB, CMTM functions take arrays as arguments. A nice feature of CMTM is that it also provides a very simplified interface to several MPI commands. For example, to broadcast a matrix, say A, from node zero to all other nodes, all nodes execute A=MMPI_Bcast(A). The system automatically distinguishes whether a node is a sender or a receiver and performs the communication. Similar calls exist to perform other procedures requiring communication such as blocking send–receive and reductions. For example, in order to compute the dot product of vectors x and y that are already distributed across the processors, all nodes execute an instruction d= MMPI_Reduce(x′∗ y,MPI_ADD) that performs a local dot product in each node and subsequently accumulates the partial results in variable d of the master node. CMTM currently runs only on Windows 2000 systems. It would not be difficult, however, to port the environment onto many of the parallel MATLAB systems reviewed in [12] that belong to the message passing category and run Linux, some of which offer more MPI-like commands than CMTM; MPITB, for example, offers more than 150 [17].

5. PPsGUI

The next concern in building the environment was to design and build a user-friendly interface for the algorithms described in Section 3. We called this PPsGUI and depict one instance of it in Fig. 4 An important design decision was to adopt a hierarchical presentation of the parameters of the pseudospectra computation. The first level that is immediately accessible to the user provides information regarding (i) the matrix at hand; (ii) the active domain and matrix-based methods (PsDM and svd in the snapshot of Fig. 4) and (iii) the current number of processors. The user can set the specific configuration parameters for the components of the first level at a second level. For example, pushing the DETAILS button next to the active Domain method selection field (see Fig. 4) triggers the activation of a dialog box that contains all the parameters of

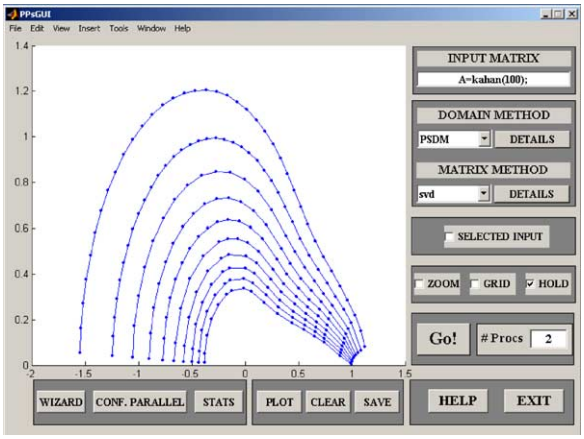

Fig. 4. Graphical tool.

the specific Domain method, such as step length and total number of steps for Cobra. This strategy ensures that the GUI is not overloaded with an overwhelming number of edit dialogs and drop lists at any level. On the other hand, we limited the depth of the parameter selection hierarchy to two levels, thus facilitating a simple and swift switch between different configurations. The results of the computations are graphically illustrated in an axis box (see Fig. 4). MATLAB treats graphics as objects, having a number of system predefined as well as user-defined attributes. In PPsGUI we exploit this property and assign to each graphically illustrated result a number of attributes such as (i) the domain and matrix-based methods that have been used for its computation; (ii) performance indicators such as runtimes and speedups; and (iii) object handles. The latter are used to serve as identifiers for the graphical manipulation of the computed results. In particular, the user can select all or part of a pseudospectrum boundary that was computed by Cobra and mark that it is to be used as input for subsequent computations, i.e. in PsDM. PPsGUI provides several graphical manipulation capabilities. The user can select a specific area and restart computations zooming therein. For example, suppose that we obtain a rough estimate of the pseudospectrum via PMoG and that we wish to compute it at a specific area of the complex plane in greater detail. To do this with PPsGUI we first select the specific area using the mouse and let PsDM take over to compute the contours of interest.

An important goal for the system is to provide transparent use of the COW. All that is required is that the software resides on a file system shared by all nodes participating in the computation. The tool automatically decides which and how many of the available processors to use. PPsGUI, however, also allows an expert user to manually define the parallel configuration of his choice. The most important feature of PPsGUI is that the user can specify that results computed by one method (e.g. Cobra) will serve as inputs of another (e.g. PsDM). Furthermore, it is possible to use data computed elsewhere and likewise store its results for future use. The environment also incorporates and lets one use, via appropriate selections in PPsGUI, many state-of-the-art algorithms for computing singular values. So, in addition to the transfer function framework, one can use MATLAB's own direct and iterative methods svd, svds as well as methods described in [2,23,25,26]. PPsGUI also allows the superposition of plots as well as three-dimensional plotting. These allow, for example, the visual comparison or detection of the evolution of pseudospectra of different matrices. A "recommender" system to help the non-expert select the best combination of algorithms is currently under development.

5.1. Automatic and manual configuration

The automatic detection of the characteristics and configuration management of the COW on which PPsGUI is deployed is a helpful component in any automatic or manual decision-making regarding the setup of the problem solving process. For this purpose we developed the Parallelism Configuration GUI (PCGUI), a software tool available within the environment. Our goals were for the software to enable transparent and user-friendly access to computational resources while also permitting expert users greater control. In particular, the user can either manually choose a particular configuration, based on the information that the tool provides, or let the tool automatically decide the configuration at runtime. For each of the available workstations in the COW, the master workstation, on which PPsGUI was initiated, gathers information which includes the number of available processors on the workstation, their type and the clock frequency, the total amount of RAM memory and the availability of MATLAB. All that is needed is that the user has an account on all the COW nodes and access to a common file system. The communication

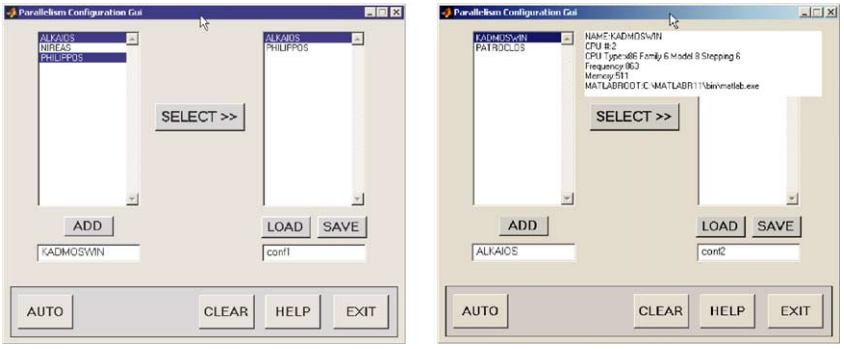

Fig. 5. The PCGUI.

mechanism is implemented as an MPI program called find_cluster. The central node executes find_cluster using a standard call to mpirun. The workstations on which find_cluster will run are typically defined in a special machines file, every line of which contains the name or IP address of the workstations. Our environment currently runs on Windows 2000 and PCGUI utilizes the registry of Windows from which it reads information regarding the present hardware and software resources as well as the command net view which returns all available workstations in the Windows 2000 domain. It is worth noting, however, that above methodology is general and virtually independent of the particular OS.

Fig. 5 illustrates PCGUI and an instance of its use. Its functionality is summarized as follows:

- The user can request the automatic detection of the features of the COW (AUTO key). On the other hand, it is possible to manually add a certain workstation (ADD key). All workstations that are eventually available for computation appear in a list above the ADD key.
- For each workstation we can illustrate information about its hardware and software configuration.
- The user can make a selection of the available workstations and form a configuration using the SELECT key. Furthermore, the user can save a configuration or load a previously saved one.
- The CLEAR key resets the tool and the HELP key triggers a help dialog.

PCGUI is a general tool, which is useful on its own, and can be used in applications other than pseudospectra.

5.2. Numerical experiments

We next present a number of experiments with PPsGUI. In the first example we used Cobra to compute an initial boundary curve $\partial\Lambda_\epsilon(A)$ with $\epsilon = 0.1$ for a very small matrix, namely kahan(100). The total runtime for two processors was 5.5 s. Then we used the points of this curve as initial points for PsDM to compute five inner curves $\partial\Lambda_{\epsilon_i}(A), \log_{10}(\epsilon_i) = -3 - 0.05i, i = 1, \ldots, , 5$. Using again two processors it required 8 s. Therefore, if we were to use just Cobra to compute the inner curves we would need four processors to top the performance of PPsGUI.

The next test matrices were gre_1107 (the numeral indicates its dimension) from Matrix Market [9] and a random sparse matrix of size $n = 2000$. We used TRGRID, that is a hybrid algorithm based on GRID and transfer functions [7]. Table 2 illustrates runtimes and corresponding speedups. We witness impressive results that exhibit superlinear speedups due to the amortization of memory load across the workstations of the cluster. We next used TRCOBRA, that

Table 2
Performance of parallel TRGRID

	P			
	1	2	4	8
Matrix: gre_1107				
Time (s)	185	80.3	42.4	20.5
Speedup	–	2.3	4.4	9
Random sparse matrix ($n = 2000$)				
Time (s)	77.2	36.2	18.6	10.6
Speedup	–	2.1	4.2	7.32

Table 3
Runtimes (s) and speedups (in parenthesis) for increasing number of correction points of parallel TRCOBRA for matrix gre_1107 and a bidiagonal matrix with random sparse elements and size $n = 40\,000$

P	Points		
	2	4	8
gre_1107			
1	63	104	194
2	40 (1.58)	–	–
4	38.5 (1.65)	39.5 (2.65)	–
8	38.2 (1.65)	38.8 (2.7)	40 (4.85)
Random matrix			
1	402	633	1135
2	244 (1.65)	–	–
4	182 (2.21)	185 (3.42)	–
8	138 (2.91)	139 (4.56)	142 (8)

is a hybrid built from Cobra and the transfer function framework [3,6]. We used the cluster of the previous example with test matrices gre_1107 and a bidiagonal matrix of size $n = 40\,000$ with random sparse entries elsewhere [41]. Table 3 lists runtimes and speedups (in parentheses) for various number of correction points at each step of Cobra.

The above experiments clearly illustrate that there is much to be gained by hybrid methods that are based on domain and matrix-based algorithms in combination with efficient parallel implementations running on clusters of workstations. To appreciate these results, we note that using a parallel implementation of the classical GRID algorithm together with a state-of-the-art general purpose SVD solver [26] for the large ($n = 40\,000$) sparse matrix of the previous example on a 50×50 mesh and an eight-node COW, the pseudospectrum took in excess of 4 h of runtime.

5.3. Towards a Grid PSE

The key motive behind our efforts was to provide access to state-of-the-art algorithms that would bring down the high cost of computing detailed pseudospectra of large matrices. The environment presented in this paper is a powerful and user-friendly system based on off-the-shelf software and hardware components and accomplishes this goal. As mathematical models become more realistic and numerical simulations more detailed, however, matrices become larger, resolutions finer and even our algorithms running on COWs will have difficulty producing answers in acceptable amounts of time. In view of recent advances in middleware and network enabled server systems designed for desktop scientific computing (see, e.g. [20]), we consider the Grid (cf. [18]) as a next framework for our environment. The fact that several of the algorithms described in Section 3 are amenable to decomposition at large granularities renders the Grid a viable framework for computing pseudospectra. Therefore, our ongoing effort is to evolve the environment into a Grid-enabled PSE whose components would be selected and triggered as a function of the properties of the problem and the availability of the Grid's resources.

6. Conclusions

We have described the design of a software environment for the effective computation and presentation of matrix pseudospectra on clusters of workstations. The computational kernel of the environment consists of several state-of-the-art domain- and matrix-based methods. The described environment permits the user to combine algorithms into hybrid methods that appear to be much more effective than algorithms that rely solely on the geometry or matrix numerics. The environment is built over MATLAB and the Cornell Multitask Toolbox that enables MPI-based message passing between concurrent instances of MATLAB. It must be noted that in spite of the fact that our implementations were necessarily based on specific off-the-shelf components, one could build a similar tool selecting alternative methods to enable MATLAB for parallel processing or even replace MATLAB altogether with tools such as SciLab [16,21]. We have also provided motivation behind the use of Grid computing for the computation of pseudospectra. The tools described in this paper are expected to provide important building blocks for the design of a Grid-enabled Pseudospectrum PSE that we consider to be the next goal in our efforts.

References

[1] E.L. Allgower, K. Georg, Continuation and path following, in: Acta Numerica 1993, vol. 2, Cambridge University Press, Cambridge, 1993, pp. 1–64.

[2] J. Baglama, D. Calvetti, L. Reichel, IRBL: an implicitly restarted block Lanczos method for large-scale Hermitian eigenproblems, SIAM J. Sci. Comput. 24 (5) (2003) 1650–1677.
[3] C. Bekas, Efficient computation of matrix pseudospectra: algorithms and tools, PhD Thesis, Computer Engineering and Informatics Department, University of Patras, June 2003 (in Greek).
[4] C. Bekas, E. Gallopoulos, Cobra: parallel path following for computing the matrix pseudospectrum, Parallel Comput. 27 (14) (2001) 1879–1896.
[5] C. Bekas, E. Gallopoulos, Parallel computation of pseudospectra by fast descent, Parallel Comput. 28 (2) (2002) 223–242.
[6] C. Bekas, E. Gallopoulos, V. Simoncini, Transfer functions and path following for computing pseudospectra, in: Proceedings of the SIAM Conference in Applied Linear Algebra, Extended Abstract, Williamsburg, VA, July 2003.
[7] C. Bekas, E. Kokiopoulou, E. Gallopoulos, E. Simoncini, Parallel computation of pseudospectra using transfer functions on a MATLAB–MPI cluster platform, in: Proceedings of the Ninth European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, vol. 2474, Springer-Verlag, Berlin, 2002.
[8] C. Bekas, E. Kokiopoulou, I. Koutis, E. Gallopoulos, Towards the effective parallel computation of matrix pseudospectra, in: Proceedings of the 15th ACM International Conference on Supercomputing (ICS'01), Sorrento, Italy, June 2001, pp. 260–269.
[9] R.F. Boisvert, R. Pozo, K. Remington, R. Barrett, J. Dongarra, The matrix market: a Web repository for test matrix data, in: R.F. Boisvert (Ed.), The Quality of Numerical Software, Assessment and Enhancement, Chapman & Hall, London, 1997, pp. 125–137.
[10] 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.
[11] M. Brühl, A curve tracing algorithm for computing the pseudospectrum, BIT 33 (3) (1996) 441–445.
[12] L.Y. Choy, Matlab∗p 2.0: interactive supercomputing made practical, Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA, September 2002. http://theory.lcs.mit.edu/∼cly/survey.html.
[13] J.J. Dongarra, V. Eijkhout, Numerical linear algebra algorithms and software, J. Comput. Appl. Math. 123 (2000) 489–514.
[14] J.J. Dongarra, V. Eijkhout, Self-adapting numerical software for next generation applications, Technical Report Lapack Working Note 157, ICL-UT-02-07, University of Tennessee, Knoxville, TN, August 2002.
[15] M. Embree, Convergence of Krylov subspace methods for non-normal matrices, PhD Thesis, Balliol College, University of Oxford, April 2000.
[16] E. Caron, et al., SCILAB to SCILAB//: the OURAGAN project, Parallel Comput. 27 (2001) 1497–1519
[17] J. Fernandez, A. Canas, A.F. Diaz, J. Gonzalez, J. Ortega, A. Prieto, Performance of message-passing MATLAB toolboxes, in: Proceedings of the Fifth International Conference on High Performance Computing for Computational Science (VECPAR 2002), Porto, Portugal, Lecture Notes in Computer Science, vol. 2565, Springer, Heidelberg, January 2003, pp. 228–241.
[18] I. Foster, C. Kesselman (Eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, San Mateo, 1998.
[19] M. Frigo, FFTW: an adaptive software architecture for the FFT, in: Proceedings of the ICASSP Conference, vol. 3, 1998, p. 1381.
[20] NetSolve/GridSolve Web site. http://icl.cs.utk.edu/netsolve.
[21] Scilab Group, Introduction to Scilab: User's Guide, Technical Report, INRIA-Unité de recherche de Rocquencourt-Project Méta2, 1997. http://www-rocq.inria.fr/scilab.
[22] N.J. Higham, The matrix computation toolbox, Technical Report, Manchester Centre for Computational Mathematics, 2002. http://www.ma.man.uc.uk/∼higham/mctoolbox.
[23] M.E. Hochstenbach, Subspace methods for eigenvalue problems, PhD Thesis, Department of Mathematics, Utrecht University, May 2003.
[24] E.N. Houstis, J.R. Rice, E. Gallopoulos, R. Bramley (Eds.), Enabling Technologies For Computational Science: Frameworks, Middleware, and Enviroments, Kluwer Academic Publishers, Dordrecht, 2000.
[25] Z. Jia, D. Niu, An implicitly restarted refined bidiagonalization Lanczos method for computing a partial singular value decomposition, SIAM J. Matrix Anal. Appl. 25 (1) 2003.
[26] E. Kokiopoulou, C. Bekas, E. Gallopoulos, Computing smallest singular triplets with implicitly restarted Lanczos bidiagonalization, Appl. Numer. Math. 49 (1) (2004).
[27] 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.
[28] The MATLAB JIT-Accelerator, Mathworks Technology Backgrounder, September 2002. http://www.mathworks.com/company/digest/sept02/accel_matlab.pdf.
[29] D. Mezher, A graphical tool for driving the parallel computation of pseudosprectra, in: Proceedings of the 15th ACM International Conference on Supercomputing (ICS'01), Sorrento, Italy, June 2001, pp. 270-276.
[30] Message Passing Interface Forum. http://www.mpi-forum.org.
[31] Pseudospectra gateway, At the Oxford University site http://web.comlab.ox.ac.uk/projects/pseudospectra.
[32] J.R. Rice, The algorithm selection problem, in: Advances in Computers, vol. 15, Academic Press, New York, 1976, pp. 65–118.
[33] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, Philadelphia, 2003.
[34] V. Simoncini, E. Gallopoulos, Transfer functions and resolvent norm approximation of large matrices, Electron. Trans. Numer. Anal. 7 (1998) 190–201.
[35] MPI Software Technology, MPI/Pro. http://www.mpi-softtech.com.

[36] K.-C. Toh, L.N. Trefethen, Calculation of pseudospectra by the Arnoldi iteration, SIAM J. Sci. Comput. 17 (1) (1996) 1–15.
[37] L.N. Trefethen, Pseudospectra of matrices, in: D.F. Griffiths, G.A. Watson (Eds.), Proceedings of the 14th Dundee Conference on Numerical Analysis 1991, Essex, UK, Longman Science and Technical, London, 1991, pp. 234–266.
[38] L.N. Trefethen, Computation of pseudospectra, in: Acta Numerica 1999, vol. 8, Cambridge University Press, Cambridge, 1999, pp. 247–295.
[39] L.N. Trefethen, A.E. Trefethen, S.C. Reddy, T.A. Driscoll, Hydrodynamic stability without eigenvalues, Science 261 (1993) 578–584.
[40] T. Wright, Eigtool: a graphical tool for nonsymmetric eigenproblems, December 2002. At the Oxford University Computing Laboratory site http://web.comlab.ox.ac.uk/pseudospectra/eigtool.
[41] T. Wright, L.N. Trefethen, Large-scale computation of pseudospectra using ARPACK and Eigs, SIAM J. Sci. Comput. 23 (2) (2001) 591–605.
[42] T.G. Wright, Algorithms and software for pseudospectra, PhD Thesis, University of Oxford, 2002.
[43] J.A. Zollweg, A. Verma, The Cornell Multitask Toolbox, Directory `Services/Software/CMTM`. http://www.tc.cornell.edu.

Constantine Bekas is a Post Doctoral associate at the Computer Science and Engineering Department of the University of Minnesota. His research interests include computation of pseudospectra of very large matrices, eigenvalue and large singular value problems and high performance and parallel computing. He was a recipient of a Bodossaki Foundation Doctoral Scholarship and a Technical Chamber of Greece award for excellent academic performance. He received a Computer Engineering Diploma in 1998 (with distinction), Master's Diploma in 2001 and PhD in 2003, all from the Computer Engineering and Informatics Department of the University of Patras, Greece.

Effrosyni Kokiopoulou is a graduate student and research associate at the Computer Science and Engineering Department of the University of Minnesota. Her research interests include large singular value problems, data mining, information retrieval and parallel computing. She was a recipient of a Bodossaki Foundation scholarship for graduate studies and numerous awards from the Hellenic Scholarship Foundation for excellent performance in undergraduate studies. She received an Engineering Diploma (class valedictorian) in 2002, from the Computer Engineering and Informatics Department of the University of Patras, Greece.

Efstratios Gallopoulos has been professor at the Department of Computer Engineering and Informatics since 1996. Before that he held positions of Senior Computer Scientist at the University of Illinois at Urbana-Champaign; assistant professor at the University of California Santa Barbara; visiting researcher at NASA Goddard Space Flight Center. As senior computer scientist he participated in research and development of the Cedar vector multiprocessor at the University of Illinois Center for Supercomputing Research and Development and in the software development of the Goodyear Aerospace Massively Parallel Processor (MPP). He has been member of scientific committees of many international conferences; editor of Computing in Science and Engineering (1994–1999) and the International Journal of High Speed Computing; co-organizer in 1992 of the NSF Workshop in Future Directions PSEs for Computational Science and member of the Steering Committee of the European Research Foundation EURESCO Conference on Advanced Environments and Tools for High Performance Computing. He received his PhD (computer science) at the University of Illinois at Urbana-Champaign in 1984 and his BSc (first class honors) in Mathematics from the Imperial College of Science and Technology in 1979.