Computer Physics Communications ••• (••••) •••–•••

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

A uniform object-oriented solution to the eigenvalue problem for real symmetric and Hermitian matrices[☆]

María Eugenia Castro, Javier Díaz, Camelia Muñoz-Caro, Alfonso Niño[*]

Grupo QCyCAR, Escuela Superior de Informática, Universidad de Castilla-La Mancha, Paseo de la Universidad 4, 13071 Ciudad Real, Spain

A R T I C L E   I N F O

Article history:
Received 31 July 2010
Received in revised form 21 October 2010
Accepted 9 November 2010
Available online xxxx

Keywords:
Eigenvalues
Eigenvectors
Object-orientation
Real symmetric matrix
Hermitian matrix

A B S T R A C T

We present a system of classes, SHMatrix, to deal in a unified way with the computation of eigenvalues and eigenvectors in real symmetric and Hermitian matrices. Thus, two descendant classes, one for the real symmetric and other for the Hermitian cases, override the abstract methods defined in a base class. The use of the inheritance relationship and polymorphism allows handling objects of any descendant class using a single reference of the base class. The system of classes is intended to be the core element of more sophisticated methods to deal with large eigenvalue problems, as those arising in the variational treatment of realistic quantum mechanical problems. The present system of classes allows computing a subset of all the possible eigenvalues and, optionally, the corresponding eigenvectors. Comparison with well established solutions for analogous eigenvalue problems, as those included in LAPACK, shows that the present solution is competitive against them.

Program summary

Program title: SHMatrix
Catalogue identifier: AEHZ_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEHZ_v1_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: Standard CPC licence, http://cpc.cs.qub.ac.uk/licence/licence.html
No. of lines in distributed program, including test data, etc.: 2616
No. of bytes in distributed program, including test data, etc.: 127 312
Distribution format: tar.gz
Programming language: Standard ANSI C++.
Computer: PCs and workstations.
Operating system: Linux, Windows.
Classification: 4.8.
Nature of problem: The treatment of problems involving eigensystems is a central topic in the quantum mechanical field. Here, the use of the variational approach leads to the computation of eigenvalues and eigenvectors of real symmetric and Hermitian Hamiltonian matrices. Realistic models with several degrees of freedom leads to large (sometimes very large) matrices. Different techniques, such as divide and conquer, can be used to factorize the matrices in order to apply a parallel computing approach. However, it is still interesting to have a core procedure able to tackle the computation of eigenvalues and eigenvectors once the matrix has been factorized to pieces of enough small size. Several available software packages, such as LAPACK, tackled this problem under the traditional imperative programming paradigm. In order to ease the modelling of complex quantum mechanical models it could be interesting to apply an object-oriented approach to the treatment of the eigenproblem. This approach offers the advantage of a single, uniform treatment for the real symmetric and Hermitian cases.
Solution method: To reach the above goals, we have developed a system of classes: SHMatrix. SHMatrix is composed by an abstract base class and two descendant classes, one for real symmetric matrices and the other for the Hermitian case. The object-oriented characteristics of inheritance and polymorphism allows handling both cases using a single reference of the base class. The basic computing strategy applied in SHMatrix allows computing subsets of eigenvalues and (optionally) eigenvectors. The tests performed

☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author. Tel.: +34 926295300; fax: +34 926295354.
E-mail address: alfonso.nino@uclm.es (A. Niño).

0010-4655/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2010.11.022

show that SHMatrix is competitive, and more efficient for large matrices, than the equivalent routines of the LAPACK package.

Running time: The examples included in the distribution take only a couple of seconds to run.

1. Introduction

The eigenvalue problem plays an important role in the quantum mechanical field. This is so since on the grounds of the Rayleigh–Ritz and McDonald theorems [1,2], the variational solution of time independent systems can be transformed into an eigenvalue problem. Here, the matrix to deal with is the Hamiltonian matrix. The corresponding matrix elements are obtained by integrating the Hamiltonian operator with all possible couples of the considered basis functions [3]. Sometimes, this results in a symmetric, real, matrix. However, in the general case we obtain a Hermitian, complex one, see for instance [4]. Two characteristics of these kinds of problems are worth mentioning. First, the quadratic dependence of the matrix with the number of basis functions leads easily to large matrices. Second, frequently we are interested only in specific subsets of all possible eigenvalues and eigenvectors.

The eigenvalue problem for square matrices is a central topic in numerical linear algebra. The standard approach for the numerical solution of the eigenproblem is to reduce the matrix to some simpler form that yields the eigenvalues and eigenvectors directly [5]. The first method of this kind dates back to 1846 when Jacobi proposed to reduce a real symmetric matrix to diagonal form by a series of plane rotations [6]. From then, the field has experienced a huge development, especially since the availability of the modern computer in the early 1950s [5]. A milestone of the field was due to Givens in 1954 [7]. In this work, Givens proposed to use a finite number of orthogonal transformations to reduce a matrix to a form easier to handle, such as a tridiagonal form. In addition, in 1958 Householder showed how to zeroing the elements outside the tridiagonal part of a matrix row and column without spoiling any previous similar transformation [8]. The Householder method became the standard reduction method of matrices to tridiagonal form on serial computers [5]. The method is described in detail in any text dealing with the eigenvalue problem, see for instance [9–12]. From the 1960s the way to compute selected eigenvalues and eigenvectors of a tridiagonal matrix involves locating the eigenvalues using a Sturm sequence and obtaining the eigenvectors by inverse iteration [13]. This approach is well presented in the classical Wilkinson's book [9]. On the other hand, for computing the whole set of eigenvalues and eigenvectors the QR technique is more efficient [12]. A different standpoint is represented by the divide and conquer approach initially proposed by Cuppen in 1981 [14]. In this approach, the matrix is reduced to tridiagonal form, splitting this last in two blocks plus a rank-one update. The procedure can be recursively applied until the block matrices are small enough. Then, we can treat the resulting blocks by other method such as QR. The "modern", stable implementation of the method was proposed in 1995 by Gu and Eisenstat [15]. The divide and conquer approach is the fastest way to obtain all the eigenvalues and eigenvectors of a symmetric matrix [5]. Besides, the method is well suited for parallel implementation.

Different available software packages allow treating the eigenvalue problem. Almost all implement descendants of the algorithms presented in the classical Wilkinson and Reinsch book [16]. In particular, many of these algorithms were codified in Fortran, in the 1970s, in the LINPACK (for numerical linear algebra) and EISPACK (for eigenproblems) packages [17,18]. LINPACK and EISPACK gives rise to LAPACK (also in Fortran) in the 1990s [19]. The last descendant in this family is ScaLAPACK, which provides a parallel implementation of a subset of LAPACK using a distributed memory parallel programming approach [20]. These packages have been implemented under a traditional imperative programming model. However, to ease the modelling of complex problems it would be interesting to have object-oriented implementations.

Object-oriented programming (OOP) develops as a mainstream programming paradigm in the 1990s [21]. OOP works by defining classes, which are sets of data and their associated functions, subroutines or procedures (usually called collectively methods in the OOP world). This programming model is characterized by three properties: encapsulation, inheritance and polymorphism (see Chapters 6 and 14 of Ref. [21]). These properties allow for an easy extension, reuse and maintainability of programs.

To ease the application of the variational approach to quantum-mechanical problems, we present here an object-oriented design and C++ implementation of a uniform, general solution to the eigenvalue problem for real symmetric and Hermitian matrices. The goal is to have a core system able to obtain selected eigenvalues and eigenvectors that can be used on any general approach for the treatment of large problems. Here, appropriate use is made of polymorphism and method overriding to achieve user-transparency. The efficiency of the development is tested against the equivalent routines implemented in LAPACK.

2. Methodology

2.1. Basic algorithmic strategy

The basic strategy used is represented in Fig. 1 as a UML activity diagram [22]. An activity diagram allows representing sequential or concurrent actions occurring in a process. Simple symbols are used to represent typical operations. Thus, diamonds identify optional execution lines and asterisks identify iterative tasks. Fig. 1 shows that computation of eigenvalues and eigenvectors is organized in three main steps: tridiagonalization of the original matrix, computation of selected eigenvalues and computation of selected eigenvectors.

The first step is to reduce the matrix to tridiagonal form. We start by normalizing the matrix using the Frobenius (Euclidean) norm [10]. As explained in the following point, we can reduce both, real symmetric and Hermitian matrices to a real tridiagonal form. The next step is to compute the desired eigenvalues, of the real tridiagonal matrix, using a Sturm sequence and the bisection method [9–11]. Next, we compute the eigenvectors for each desired eigenvalue using inverse iteration [9–11]. For the initial random guess eigenvector, in the inverse iteration procedure, we implement the Park and Miller MINSTD random number generator [23]. Each eigenvector is approached iteratively by solving a linear set of equations for each eigenvalue using a LU decomposition [12,24] of the tridiagonal matrix. When degenerate eigenvalues do exist the corresponding eigenvectors are orthogonormalized using the modified Gram–Schmidt (MGS) algorithm [10,25]. Each tridiagonal matrix eigenvector is rotated back to the original matrix by applying inversely the performed Householder transformations as described in the next section.

To save storage space, the real symmetric or Hermitian matrix is stored in packed upper triangular form in row-major order. Thus for a $n \times n$ matrix only $n(n+1)/2$ elements are needed.

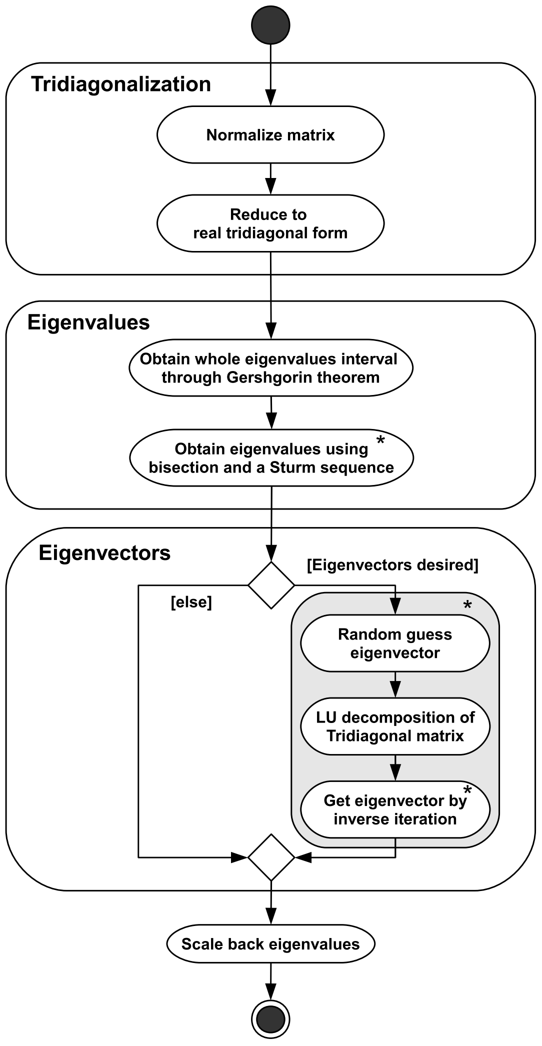
Tridiagonalization
Normalize matrix
Reduce to real tridiagonal form
Eigenvalues
Obtain whole eigenvalues interval through Gershgorin theorem
Obtain eigenvalues using bisection and a Sturm sequence
*
Eigenvectors
[Eigenvectors desired]
[else]
*
Random guess eigenvector
LU decomposition of Tridiagonal matrix
Get eigenvector by inverse iteration
*
Scale back eigenvalues

Fig. 1. UML activity diagram for the basic algorithmic strategy implemented in the SHMatrix system of classes.

2.2. Reduction to real tridiagonal form

The first step in both the real symmetric and complex Hermitian cases is to reduce the matrix to tridiagonal form. For the real case, an efficient and well-known method resorts to the use of Householder reflections [9–12]. A similar approach can be adopted for complex Hermitian matrices, generating a complex tridiagonal matrix. However, as shown by Shukuzawa et al. [26], any complex Hermitian matrix can be transformed to a real tridiagonal matrix using modified Houselholder reflections. Therefore, we can devise a single computing schema for the real and Hermitian cases. In addition, since the tridiagonal matrix is always real, this approach has the advantage that the algorithms for the computation of the eigenvalues and eigenvectors do not depend on the kind of problem, real or Hermitian matrix. The formalism we propose is the following.

Annihilation of the non-tridiagonal elements in the m column of a $n \times n$ real symmetric or Hermitian matrix, $\mathbf{A}$, is obtained by an orthogonal transformation:

$$\mathbf{A}' = \mathbf{P}' \cdot \mathbf{A} \cdot \mathbf{P}^{+} \tag{1}$$

where the superscript + represents transpose in the real case and conjugate transpose in the complex case. The matrix $\mathbf{P}$ is defined as,

$$\mathbf{P} = \mathbf{I} - \beta \cdot \mathbf{u} \cdot \mathbf{u}^{+} \tag{2}$$

with $\mathbf{I}$ being the unitary matrix and $\mathbf{u}$ a unit column vector being obtained as $\mathbf{x}/|\mathbf{x}|$, where the $\mathbf{x}$ vector elements are:

$$\begin{aligned} &x_i = 0, \quad \forall i \leqslant m \\ &x_m = a_{m,m+1} \pm s \\ &x_i = a_{m,i} \quad \forall i > m+1 \end{aligned} \tag{3}$$

and s is defined as,

$$s = \left(\sum_{i=m+1}^{n-1} |a_{m,i}|^2 \right) \tag{4}$$

In addition, β is obtained as

$$\begin{aligned} &\beta = 2 \quad \text{in the real case} \\ &\beta = 1 - \kappa \quad \text{in the complex case with} \\ &\kappa = -\left(s + a^{*}_{m,m+1}\right)/(s + a_{m,m+1}) \end{aligned} \tag{5}$$

The sign of s in Eq. (3) equals the sign of $a_{m,m+1}$ in the real case and is negative for the complex Hermitian case. The transformation reducing column m of matrix $\mathbf{A}$ to tridiagonal form ($\mathbf{A}'$) is obtained as,

$$\begin{aligned} &\mathbf{A}'\mathbf{A} - \mathbf{q} \cdot \mathbf{u}^{+} - \mathbf{u} \cdot \mathbf{q}^{+} \quad \text{with} \\ &\mathbf{q} = \mathbf{p} - K \cdot \mathbf{u} \quad \text{and} \\ &\mathbf{p} = \beta^{+} \cdot \mathbf{A} \cdot \mathbf{u} \end{aligned} \tag{6}$$

The K symbol in Eq. (6) stands for:

$$\begin{aligned} &K = 2\mathbf{u}^{\mathbf{T}} \cdot \mathbf{A} \cdot \mathbf{u} \quad \text{in the real case} \\ &K = \left[1 - \text{real}(\kappa)\right]\mathbf{u}^{\mathbf{T}} \cdot \mathbf{A} \cdot \mathbf{u} \quad \text{in the complex case} \end{aligned} \tag{7}$$

The previous procedure must be repeated from column 0 to column $n-3$ for a $n \times n$ matrix.

Finally, in the case that all or some of the eigenvectors of the tridiagonal matrix have been computed, they can be rotated back to the original matrix by applying inversely the Householder transformations in the following form:

$$\mathbf{V}^{i}_{j} = \mathbf{V}^{i+1}_{j} - \beta_j \cdot \mathbf{u}^{+}_{j} \cdot \left[\mathbf{u} \cdot \mathbf{V}^{i+1}_{j}\right] \quad \text{for } i = n = 3 \text{ to } 0 \tag{8}$$

where $\mathbf{V}$ represents the eigenvectors matrix and the i superindex corresponds to the rotation to apply.

2.3. Software design

For allowing a uniform use of real symmetric and Hermitian matrices we resort to the polymorphism characteristic of object-orientation, see Section 14.2 in Ref. [21]. Thus, a hierarchy of classes is build using the inheritance relationship. This is shown in Fig. 2 in a UML class diagram [22]. Fig. 2 shows that we define a base, parent class, SHMatrix, which contains all the data and procedures, methods, common to the treatment of real symmetric and Hermitian matrices. This is an abstract class (see Section 14.5 of Ref. [21]) since the implementation of the data dependent methods, such as the tridiagonalization of the matrix and the computation of eigenvectors, is deferred to the descendant classes. The method for computing selected eigenvalues from a real tridiagonal matrix, however, is implemented here. From SHMatrix we have two descendant classes: SMatrix and HMatrix. The first one deals with the symmetric case, whereas the second corresponds to the Hermitian one. In these classes, the specific implementations of the tridiagonalization and computation of eigenvectors methods are defined. In other words, the methods are overridden. Finally,

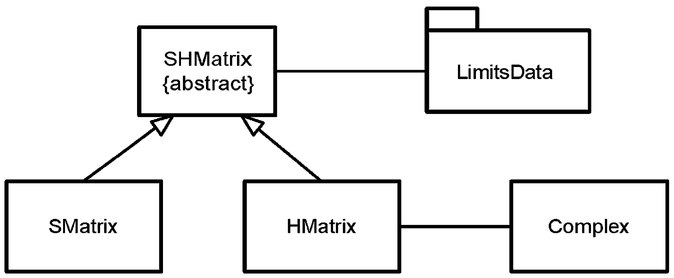

Fig. 2. UML class diagram for the SHMatrix system of classes.

the ability to request computation of eigenvalues and eigenvectors or just eigenvalues is provided by overloading the constructor methods in the classes.

Fig. 2 shows that HMatrix has an association relationship with a “Complex” class. This class allows for the use of a complex number data type and the application of the complex arithmetic operators. Finally, we observe in Fig. 2 that SHMatrix incorporates a package, LimitsData, which defines the numerical parameters used, such as the machine epsilon or the tolerance in eigenvalues and eigenvectors.

With the previous design it is possible to use polymorphism to define data structures formed by parent class references, where each reference can refer to SMatrix or HMatrix objects. Analogously, it is possible to define methods accepting SHMatrix references as formal parameters, independently of the reference to be tied to SMatrix or HMatrix objects. In this form, we can handle in a user-transparent way real symmetric or Hermitian matrices in a program.

2.4. Implementation

The design and algorithms previously described are implemented in ANSII C++. For the class Complex shown in Fig. 2, we use the standard C++ complex library. To increase the efficiency of our code, the inner loops have been unrolled four times. In addition, all variables and arrays are declared in the outer code block of each method. Data structures are dynamically allocated. So, for instance, there is no need to change matrices dimensions and recompile. In addition, all auxiliary data structures are used on the fly. Thus, they are defined and allocated when they are needed, and the associated memory is released when they are no longer necessary.

2.5. Using the system of classes

The system of classes can be easily used in a program by following a few simple steps:

(a) The program must include the definition (header) file of the class to use. Thus, for the real symmetric case we have:

```
#include "SMatrix.h"
```

`SMatrix.h` must be replaced by `HMatrix.h` when using a Hermitian matrix.

(b) Within the program, a pointer to the base class must be declared

```
SHMatrix *miMatriz;
```

(c) On the other hand, it is necessary to create the matrix to work with (matrix `a` with size `n`), and the eigenvalues and eigenvectors matrices (`eValues` and `V`).

(d) When needed, the base class reference can be used to instantiate an object of the desired class, for instance

```
miMatriz=new SMatrix (a,eValues,n,n,V);
```

`SMatrix` must be replaced by `HMatrix` if we are working with a Hermitian matrix.

(e) When appropriate, a polymorphic call to the `eigen()` method can be made to compute eigenvalues and, optionally, eigenvectors:

```
miMatriz->eigen();
```

Observe that the call is unique independently of the actual nature of the object (that is, if it belongs to the `SMatrix` or `HMatrix` class). The results will be accessible through the `eValues` and, optionally, the `V` matrices.

3. Performance

When a new solution based on a new programming paradigm is proposed for a previous problem two conditions must be fulfilled. First, we must have the advantages of the new paradigm, object-orientation in this case. Second, the performance of the new solution must be comparable to prior accepted and well established solutions. Here, we compare the performance of the methods implementing the algorithms described in Section 2 against equivalent routines of the LAPACK package [19]. Thus, evaluation of eigenvalues and eigenvectors in the real case are tested against the DSPEVX LAPACK routine [19]. For the Hermitian case, we use the ZHPEVX LAPACK routine [19]. The LAPACK routines, as well as our methods, use a triangular packed array in row-major order and allow selecting a specific number of eigenvalues and eigenvectors. The LAPACK version used is the latest one: 3.2.2.

In all the test cases, we use $n \times n$ dense real symmetric or Hermitian matrices. The matrices are filled with real numbers generated randomly in the $[0, 99]$ interval using the C language rand( ) function. Initial tests are performed to determine the equivalence with the LAPACK results. Thus, we have considered real and Hermitian matrices of size $n = 10$, 50 and 100. The results show that the maximum difference in eigenvalues between the present approach and LAPACK is as low as 1.2×10^{-5}% and 1.9×10^{-6}% in the real and Hermitian cases, respectively.

On the other hand, to test the performance of the object-oriented and the LAPACK approaches, we use real symmetric and Hermitian matrices of sizes representative of realistic problems. Thus, we use $n \times n$ matrices with n ranging from 1000 to 10 000 in steps of 1000. In the examples, we compute different proportions of eigenvalues and eigenvectors, from 10% to 100% of the total in steps of 10%. For the classes developed in this work, see Fig. 2, we write a driver program using a polymorphic call. Thus, we define a reference of base class, SHMatrix, but instantiate objects of SMatrix or HMatrix classes as needed. In this form, we account for the overhead associated to the dynamic binding (*i.e.*, the run-time resolved call to the methods) of the SHMatrix class, which are overridden in SMatrix and HMatrix. For our classes, and the LAPACK routines, compilation is carried out under the Linux operating system using the gcc compiler with the -O3 compiling option, *i.e.*, the higher level of code optimization. The system we use is a computer cluster running under Rocks 5.3 [27]. The compute nodes are Quad-Core AMD Opteron™ 2354 (2.2 GHz) with 8 GB of main memory.

Figs. 3 and 4 compare the speedup of the present approach against LAPACK, in the real symmetric and Hermitian cases, respectively. The speedup is calculated as the quotient between the time needed to get the LAPACK results and those with our approach. The computing time ranges between a couple of seconds (for all the cases with a matrix of size 1000) and 20 351 seconds (LAPACK with a Hermitian matrix of size 10 000 and 100% of eigenvalues and eigenvectors). Since the time needed to process the $n = 1000$ cases is so small, the first point in the curves of Figs. 3

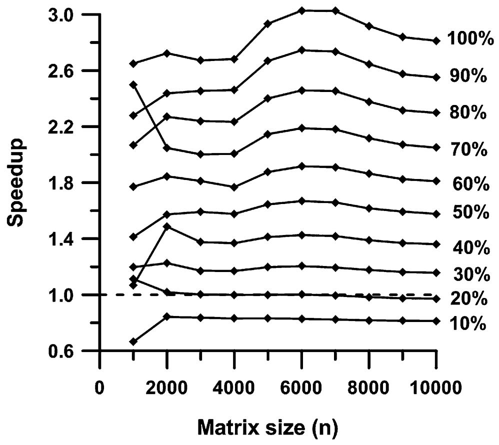

Fig. 3. Speedup (LAPACK over SHMatrix results) as a function of the matrix size, *n*, for the calculation of different subsets of eigenvalues and eigenvectors (in percent of the total) in real symmetric matrices. The dashed line marks a baseline of speedup = 1.0.

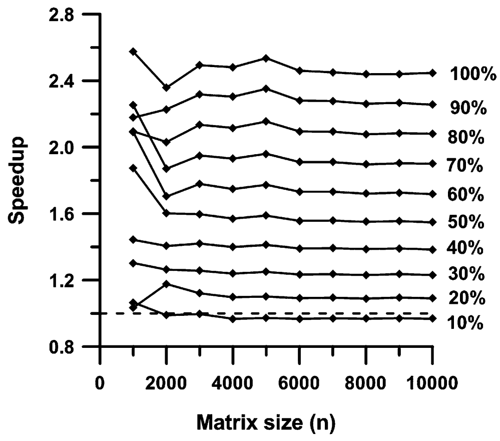

Fig. 4. Speedup (LAPACK over SHMatrix results) as a function of the matrix size, *n*, for the calculation of different subsets of eigenvalues and eigenvectors (in percent of the total) in Hermitian matrices. The dashed line marks a baseline of speedup = 1.0.

and 4 is little reliable. Both figures show that, when the proportion of eigenvalues and eigenvectors is kept constant, the speedup is relatively independent of the matrix size. Fig. 3 shows that in the real symmetric and Hermitian cases the present approach performs in general better than LAPACK. The only exception is when few eigenvalues and eigenvectors are requested (the 10% curve in the real and Hermitian cases, and the last three points of the 20% curve in the real case). In all the other cases, the speedup of the present approach is greater than one. The speedup reaches 2.0–3.0 and 2.0–2.6, in the real and Hermitian cases, when a large number of eigenvalues and eigenvectors (80%–100%) are requested. The speedup data show that for large real symmetric and Hermitian matrices, and a large number of requested eigenvalues and eigenvectors, our object-oriented approach is more than twice as fast as LAPACK.

The origin of the difference can be determining by considering the most time consuming case, *i.e.*, the 100% eigenvalues and eigenvectors case. Using the speedup, we consider the contribution to the total time of the two main tasks in the procedure: tridiagonalization + eigenvalues computation, the first one, and eigenvectors determination, the second. Fig. 5 collects the results for the symmetric and Hermitian cases. For the eigenvalues computation, we observe that in the real and Hermitian cases LAPACK is more efficient. However, the eigenvectors computation is much more efficient with the SHMatrix system. In fact, the difference

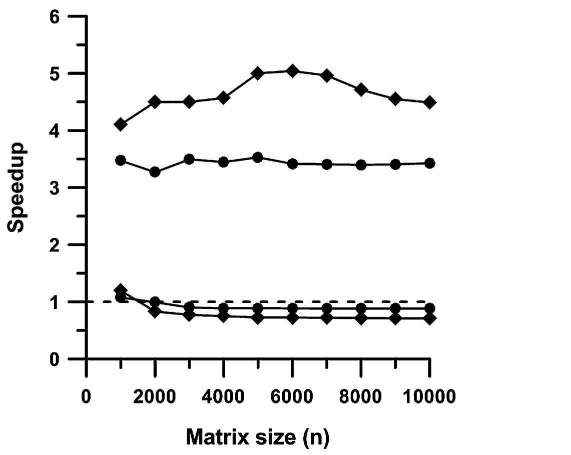

Fig. 5. Speedup (LAPACK over SHMatrix results) as a function of the matrix size, *n*, for the computation of all the eigenvalues (lower curves) and eigenvectors (upper curves) in $n \times n$ matrices. Diamonds mark real symmetric matrices data and circles correspond to Hermitian matrices. The dashed line marks a baseline of speedup = 1.0.

in efficiency is large enough to compensate for the less efficient eigenvalues computation. Therefore, the better overall results obtained by our approach are due to a higher efficiency of the eigenvectors computation procedure.

Acknowledgements

This work has been co-financed by FEDER funds and the Consejería de Educación y Ciencia de la Junta de Comunidades de Castilla-La Mancha (grant #PBI08-0008). The Ministerio de Educación y Ciencia (grant #AYA 2008-00446) is also acknowledged.

References

[1] E.C. Kemble, Rev. Mod. Phys. 1 (1929) 157.
[2] J.K.L. MacDonald, Phys. Rev. 43 (10) (1933) 830.
[3] I.N. Levine, Quantum Chemistry, fifth edition, Prentice–Hall, 1999.
[4] M.E. Castro, A. Niño, C. Muñoz-Caro, WSEAS Trans. Inform. Sci. Appl. 7 (2) (2010) 263–272.
[5] G.H. Golub, H.A. van der Vorst, J. Comput. Appl. Math. 123 (2000) 35.
[6] C.G.J. Jacobi, J. Reine, Angew. Math. 30 (1846) 51.
[7] W. Givens, Numerical computation of the characteristic values of a real symmetric matrix, Oak Ridge Report Number ORNL 1574 (physics), 1954.
[8] A.S. Householder, J. ACM 5 (1958) 339.
[9] J.H. Wilkinson, The Algebraic Eigenvalue Problem, Clarendon Press, Oxford, 1965.
[10] B.N. Parlett, The Symmetric Eigenvalue Problem, SIAM, Philadelphia, 1998; Republication of the original work published by Prentice–Hall, 1980.
[11] G.H. Golub, C.F. van Loan, Matrix Computations, third edition, Johns Hopkins Univ. Press, 1996.
[12] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, Numerical Recipes, The Art of Scientific Computing, Cambridge Univ. Press, Cambridge, 2007.
[13] J.M. Ortega, in: A. Ralston, H.S. Wilf (Eds.), Mathematics for Digital Computers, vol. 2, John Wiley & Sons, 1967, p. 94.
[14] J.J.M. Cuppen, Numer. Math. 36 (1981) 177.
[15] M. Gu, S.C. Eisenstat, SIAM J. Matrix Anal. Appl. 16 (1995) 172.
[16] J.H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, Linear Algebra, vol. 2, Springer-Verlag, 1971.
[17] J.J. Dongarra, C.B. Moler, J.R. Bunch, G.W. Stewart, LINPACK Users' Guide, 1979, LINPACK: http://www.netlib.org/lapack/; last access July 22, 2010.
[18] EISPACK: http://www.netlib.org/eispack/; last access July 22, 2010.
[19] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J.J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK Users' Guide, third edition, SIAM, 1999, LAPACK: http://www.netlib.org/lapack/; last access October 9, 2010.
[20] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J.J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R.C. Whaley, ScaLAPACK Users' Guide, SIAM, 1997, ScaLAPACK: http://www.netlib.org/scalapack/; last access July 22, 2010.
[21] B. Meyer, Object-Oriented Software Construction, second edition, Prentice–Hall, 2000.

[22] M. Fowler, UML Distilled, third edition, Addison–Wesley, 2003.
[23] S.K. Park, K.W. Miller, Comm. ACM 31 (10) (1988) 1192.
[24] G. Engeln-Müllges, F. Uhlig, Numerical Algorithms with Fortran, Algorithm 4.32 Springer, 1996, Chap. 4, p. 91.
[25] C.D. Meyer, Matrix Analysis and Applied Linear Algebra, SIAM, 2000.
[26] O. Shukuzawa, T. Suzuki, I. Yokota, Proc. Japan Acad. Ser. A 72 (1996) 102.
[27] Rocks 5.3.: http://www.rocksclusters.org/; last access July 27, 2010.