Chapter 1
On the Semiclassical Jacobi Algorithm *

María T. Cámara and Domingo Giménez†

Abstract

In this paper a new Jacobi method to solve the symmetric eigenvalue problem is presented. This method is called "semiclassical", and uses some of the characteristics of both the cyclic and classical Jacobi methods. By combining this method with an adequate threshold strategy a reduction in the execution time is obtained.

1 Comparison of cyclic and classical Jacobi methods

A Jacobi method for the symmetric eigenvalue problem consists in the generation of a succession $\{A_s\}$ through:

$$A_{s+1} = Q_s A_s Q_s^t, \ \ s = 1, 2, ...$$

with $A_1 = A$, and Q_s represents Givens rotations in the plane (i, j), with $1 \leq i, j \leq n$, nullifying a_{ij} and a_{ji}.

Under certain conditions it is demonstrated that A_s converges to a diagonal matrix

$$D = Q_k Q_{k-1} ... Q_2 Q_1 A Q_1^t Q_2^t ... Q_{k-1}^t Q_k^t$$

whose diagonal elements are the eigenvealues of A [4].

The classical Jacobi method proceeds by choosing in each iteration as element to be nullified that of greater absolute value from among the nondiagonal elements. So, a scheme of this algorithm could be:

WHILE $Off(A) > BoundError$
  Compute a_{ij} with maximum absolute value from among the nondiagonal elements
  Compute Rotation $Q(i, j, \theta)$ such that $(QAQ^t)_{ij} = 0$
  $A = QAQ^t$

Because in each iteration of the algorithm the element of greater absolute value is chosen, the number of iterations is small but the execution time is very long, due to the additional work of the calculation of the maximum. Other Jacobi methods ([4], [10], [1], [8], [2], [6], [3], [7], etc...) proceed by nullifying in each sweep each nondiagonal element once (so, each sweep will consist of $n(n-1)/2$ steps), using a certain order to nullify the elements. In this way the calculation of the maximum is avoided and an order $O(n^3)$ is obtained per sweep, while the classical method has an order $O(n^4)$ in $n(n-1)/2$ steps. However, more steps are needed to reach convergence than in the classical method.

---

*The experimental results here shown have been obtained on i860s in the Argonne National Laboratory and in the Center for High Performance Computing of The University of Texas at Austin

†Departamento de Informática y Sistemas. Universidad de Murcia. Aptdo 4021. 30001 Murcia. Spain. (domingo@dif.um.es)

TABLE 1
Number of sweeps of the classical and cyclic-by-rows methods.

	128	192	256	320
$JCLA$	4.50	4.56	4.59	4.62
$JROW$	9	9	10	10

The convergence of the classical method is demonstrated [11], [9], and also the convergence of some cyclic methods [4], [2], [6].

In table 1 we compare the number of sweeps needed to reach the convergence with the classical and the cyclic-by-rows methods. In the classical method we count $n(n-1)/2$ steps as a sweep.

2 A new Jacobi method

If before each sweep the nondiagonal elements are sorted from greatest to least absolute value and are nullified in this order, we would need in each sweep a previous sorting of $n(n-1)/2$ data, which done with an optimum method as the Quicksort [5], represents an average order $O(n^2 \lg\ n)$. As the order of the Jacobi method in each sweep is $O(n^3)$, this previous sorting would not modify the order per sweep. On the other hand, the previous sorting means that in each sweep we nullify first the elements of greatest absolute value, by which the number of necessary sweeps for the convergence would be reduced.

Once the sorting is made, the first iterations are performed on the elements with greater absolute value, but in the nullification of these elements the nondiagonal elements are modified, and so these elements are not sorted in the order obtained before the sweep. For this reason, our method will not performe the sorting of the elements but, following the Quicksort plan, we take the first element and obtain the greater and the smaller elements in absolute value, taking a step further in this procedure of "semisorting" with those elements greater than the chosen element, and continuing until the greatest element is obtained. This "semisorting" has an average order of $O(n^2)$.

In the last iterations of each sweep, elements with a small absolute value are nullified, and so these nullifications should contribute little to the convergence of the method, therefore we consider it preferable to divide each sweep in d subsweeps, and each subsweep in $(n(n-1)/2)/d$ steps, $1/d$ indicating the portion of nondiagonal elements we nullify in each subsweep. When $d = n(n-1)/2$ the method is the classical Jacobi method where the maximum is obtained with a "semisorting".

We call this method JSCd. And the scheme would be:

WHILE $Off(A) > BoundError$
  "Semisorting" of nondiagonal elements from greatest to least absolute value
  FOR $k = 0$ TO $(n(n-1)/2)/d - 1$
    Take pair (i, j) that occupies the place k in the "semisorting"
    Compute Rotation $Q(i, j, \theta)$ such that $(QAQ^t)_{ij} = 0$
    $A = QAQ^t$

The order per sweep of JSCd (cost per $n(n-1)/2$ iterations) is $O(n^3)+dO(n^2)$, therefore the cost per sweep of JSCd is worse than that of JROW, but this can be compensated with an increase of the speed of convergence, because with JSCd elements with greater absolute value are nullified.

Table 2

Number of sweeps with the cyclic-by-rows and the "semiclassical" method with different values of d

	256	320	384	448	512
$JROW$	10	10	10	10	10
$JSC2$	7.50	8.00	9.50	8.00	9.00
$JSC4$	6.75	6.75	7.00	7.00	7.25
$JSC6$	6.17	6.50	6.67	6.50	6.50
$JSC8$	5.88	6.25	6.38	6.25	6.25
$JSC10$	5.60	5.90	6.00	6.00	6.10
$JSC12$	5.42	5.58	5.67	5.67	5.83
$JSC14$	5.43	5.36	5.57	5.64	5.78

Table 3

Execution time in seconds of the cyclic-by-rows and the "semiclassical" method with different values of d.

	256	320	384	448	512
$JROW$	134.80	279.87	493.09	825.54	1230.12
$JSC2$	114.30	245.80	491.58	711.74	1158.88
$JSC4$	109.87	221.75	393.70	649.02	987.96
$JSC6$	106.86	223.42	393.15	629.75	927.83
$JSC8$	106.29	226.44	393.96	623.41	919.83
$JSC10$	107.95	220.81	379.42	616.63	927.87
$JSC12$	108.96	216.58	375.63	597.92	902.46
$JSC14$	113.82	216.31	376.57	611.24	917.39

3 Experimental results

In this Section the experimental results obtained are shown. These results have been obtained on i860s with programs in C, and the matrices used are real, symmetric and dense with random values between -10 and 10.

In table 2 the number of necessary sweeps to reach the convergence with JROW and JSCd for different values of d is shown. The number of sweeps decreases as d increases. This could produce a reduction in the execution time but, due to the "semisorting", this reduction in the number of sweeps does not always produce such a reduction. In table 3 the execution times are shown.

Until now the results have been obtained nullifying a pair of nondiagonal elements in each iteration. If the nullification is avoided when the nondiagonal elements to nullify are bellow a certain threshold, better execution times are obtained [11], [9]. The sweeps obtained using variable threshold strategy [9] and the "semiclassical" method combined with a fixed thershold strategy are shown in table 4. In this case the number of sweeps needed to reach the convergence is less than those need not using threshold, and so better execution times are obtained. These times are shown in table 5.

In table 6 the reduction obtained with the "semiclassical" method, the variable threshold strategy and a fixed threshold strategy combined with the "semiclassical" method is shown.

TABLE 4
Number of sweeps with the cyclic-by-rows and the "semiclassical" method with different values of d, using threshold

	256	320	384	448	512
$JROW$	5.93	6.03	6.08	6.10	6.14
$JSC2$	7.00	8.00	7.50	8.00	7.50
$JSC4$	6.25	6.50	6.50	6.75	6.75
$JSC6$	5.50	5.67	5.83	5.83	6.00
$JSC8$	5.38	5.25	5.38	5.62	5.62
$JSC10$	5.00	5.10	5.20	5.30	5.40
$JSC12$	4.83	4.83	5.08	5.08	5.17
$JSC14$	4.57	4.78	4.86	5.07	5.00

TABLE 5
Execution time in seconds of the cyclic-by-rows and the "semiclassical" method with different values of d, using threshold

	256	320	384	448	512
$JROW$	114.39	224.88	383.82	618.49	907.56
$JSC2$	105.16	241.51	397.00	704.53	992.96
$JSC4$	98.14	210.33	358.91	620.58	905.24
$JSC6$	89.66	190.22	335.84	551.90	835.38
$JSC8$	91.31	179.54	318.52	541.14	794.47
$JSC10$	87.07	178.66	311.35	523.40	781.80
$JSC12$	85.46	169.51	309.13	504.02	754.32
$JSC14$	83.94	170.74	307.15	481.00	764.11

The best results are obtained using the threshold/"semiclassical" method.

4 Conclusions

We have shown a Jacobi method that, using ideas from the classical and the cyclic methods, reduces the execution time of the previous methods.

With the method presented the execution time of the cyclic methods is improved using and not using threshold. In the experiments realized a reduction of 40% in the execution time is obtained when an adequate threshold strategy is combined with the "semiclassical" method.

With the experimental results obtained we cannot deduce the optimum value of d.

At present we are working on the application of these ideas to other eigenvalue problems.

References

[1] R. P. Brent and F. T. Luk, *A Systolic Architecture for Almost Linear-Time Solution of the Symmetric Eigenvalue Problem*, Tech. Report 82-525, Department of Computer Science, Australian National University, Camberra, 1982.

TABLE 6

Reduction in the execution time with respect to the cyclic-by-rows method

	256	320	384	448	512
variable threshold	15.14	19.64	22.16	25.08	26.22
semiclassical	21.14	22.71	23.82	27.57	26.63
fixed threshold + semiclassical	37.72	39.43	37.70	41.73	38.67

[2] P. J. Eberlein and M. Mantharam, *Jacobi Sets for the Eigenproblem and Their Effect On Convergence studied by Graphic Representation*, Proc. Fourth SIAM Conf. on Parallel Processing for Scientific Computing, (1989), pp. 15-22.

[3] P. J. Eberlein and H. Park, *Efficient Implementation of Jacobi Algorithms and Jacobi Sets on Distributed Memory Architectures*, Jour. Parallel and Distrib. Comput., 8 (1990), pp. 358-367.

[4] G. E. Forsythe and R. Henrici, *The cyclic Jacobi method for computing the principal values of a complex matrix*, Trans. Amer. Math. Soc., 94 (1960), pp. 1–23.

[5] D. E. Knuth, *The Art of Computer Programming. Vol 3: Sorting and Searching*, Addison-Wesley, Massachusetts, 1973.

[6] F. T. Luk and H. Park, *A Proof of Convergence for Two Parallel Jacobi SVD Algorithms*, IEEE Trans. on Computers, 28, N 6 (1989), pp. 806-811.

[7] M. Mantharam and P. J. Eberlein, *New Jacobi-sets for parallel computations*, Parallel Computing, 19 (1993), pp. 437-454.

[8] J. J. Modi, *Parallel Algorithms and Matrix Computation*, Clarendon Press, Oxford, 1988.

[9] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[10] A. H. Sameh, *On Jacobi and Jacobi-like algorithms for a parallel computer*, Math. Comput., 25 (1971), pp. 579–590.

[11] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.