Proceedings of the 2006 IEEE
Conference on Computer Aided Control Systems Design
Munich, Germany, October 4-6, 2006

Computational Experience with Robust Pole Assignment Algorithms

Vasile Sima André L. Tits Yaguang Yang

***Abstract*— Two algorithms for robust pole assignment by state feedback, proposed by Kautsky, Nichols and Van Dooren (1985) and by Tits and Yang (1996) are briefly reviewed. MATLAB code implementations of these algorithms, `place` (from the MATLAB Control System Toolbox) and `robpole` (from SLICOT), are then numerically compared on randomly generated test data sets, as well as on examples from two benchmark collections, in terms of the robustness (insensitivity of poles to variations in plant parameters) of the closed-loop systems they produce. The functions `place` and `robpole` are also compared with each other, as well as with the (non robust) pole assignment code `pass` (from SLICOT) in terms of CPU time and accuracy of the pole assignment.**

***Index Terms*— computer-aided control system design, numerical algorithms, numerical linear algebra, pole assignment, software library**

I. INTRODUCTION

It has been long known that controllability of a state space pair (A, B) is necessary and sufficient for arbitrary assignment of the closed-loop eigenvalues (poles of the closed-loop transfer function) by state feedback, i.e., for existence of a feedback matrix F such that $A + BF$ has the requisite eigenvalues. It also has been long known that, when there are more than one input, i.e., B has more than one column, such F is typically non-unique. Many authors have proposed approaches to make use of this freedom to achieve other properties, such as low sensitivity of the closed-loop eigenvalues to variations in the plant parameters, e.g., [1]–[8].

The work of Kautsky, Nichols and Van Dooren [9] in particular has received sustained attention in both the research community and the industrial community. A key reason for this success is probably the fact that the algorithms proposed in that paper require few arithmetic operations per iteration. "Method 0" from [9] forms the basis of the `place` MATLAB code, available in the Control Toolbox [10]. While no convergence analysis is included in [9], it was later shown in [11] that, in Method 0 of [9], the determinant of the closed-loop unit-length-eigenvector matrix X is increased at each iteration, and that all limit points of the constructed sequence X^k are stationary points for the problem of maximizing $\det(X)$ subject to X being such closed-loop unit-length-eigenvector matrix for the prescribed eigenvalue assignment.

V. Sima is with the National Institute for Research & Development in Informatics, 011455 Bucharest, Romania `vsima@ici.ro`

A.L. Tits is with the Electrical and Computer Engineering and the Institute for Systems Research, University of Maryland, College Park, U.S.A. `andre@umd.edu`

Y. Yang is with the Guidance, Navigation and Control Department, Orbital Sciences Corporation, Dulles, VA, U.S.A. `yaguang.yang@worldnet.att.net`

Tests reported in [11] however show that Method 0 sometimes falls far short of globally maximizing $\det(X)$ (see, e.g., Fig. 1 in that paper). Based on this observation, Method 0 was enhanced in [11] by updating two columns (of X) at a time rather than just one. This also allows assignment of (pairs of) complex (conjugate) eigenvalues, while Method 0 is limited to real eigenvalue assignment. (Function `place` handles complex conjugate prescribed eigenvalues in an ad hoc fashion.) Several (related) algorithms were proposed in [11]; "Algorithm 5.1a" (see top paragraph of page 1440 in [11]) was later implemented in MATLAB as `robpole`, available from the SLICOT Library [12]–[14] web site [15].

The purpose of this paper is to present a comparative numerical study of the most recent versions of `place` (latest version available to us in Control System Toolbox, Version 6.2) and `robpole` (Version 1.5) in terms of their accuracy (in assigning the prescribed eigenvalues), their speed, and the (in)sensitivity of the eigenvalues of the closed-loop matrices they produce to variations in the data. Their accuracy and speed are also compared to those of `pass`, another MATLAB script available from SLICOT for eigenvalue assignment by state feedback (but without "robustification"), which calls (via a MEX-file) the Fortran routine `SB01BD` from the SLICOT Library [13]. This Fortran routine is based on the algorithm in [16].

II. POLE ASSIGNMENT ALGORITHMS

In this section, we sketch Method 0 of [9] and Algorithm 5.1a of [11]. (The latter is specified in the top paragraph of the second column of page 1440 in the paper.) To emphasize their similarity, we use the formalism of [11] for both algorithms.

Let A be $n \times n$ and B be $n \times m$, both real. Let Λ be the $n \times n$ diagonal matrix with diagonal entries λ_i equal to the prescribed closed-loop eigenvalues. Both algorithms construct a sequence of valid unit-length-eigenvector matrices X for a closed-loop matrix $A+BF$ that is similar to Λ, for some real "feedback matrix" F; i.e., X and F are such that

$$(A + BF)X = X\Lambda. \tag{1}$$

For a given nonsingular X, whose columns follow the same complex-conjugacy pattern as the entries of Λ, there exists a real F such that (1) holds if and only if, for every i, the ith column x_i of X lies in the subspace

$$\mathcal{S}_i := \{x : (A - \lambda_i I)x \in \mathrm{range}(B)\}. \tag{2}$$

It is well known that the eigenvalues of *normal* matrices, i.e., matrices whose eigenvector matrix is unitary, are least

0-7803-9797-5/06/$20.00 ©2006 IEEE

sensitive (to first order) to variations in the matrix entries. Based on this observation (applied to $A + BF$), both algorithms attempt to maximize orthogonality of the columns of X subject to each of them lying within the corresponding $\mathcal{S}_i$. Specifically, they attempt to maximize $\det(X)$ while keeping each x_i on the Euclidean unit sphere of $\mathcal{S}_i$, i.e., all $x \in \mathcal{S}_i$ with $\|x\| = 1$. Method 0 of [9] does this by performing, at each iteration, a global maximization with respect to *one* of the columns, and cyclically iterating over the columns. Such iterations are performed at low computational cost of order $O(n^2)$. Algorithm 5.1a of [11] does this (still globally) *two* columns at a time. As shown in [11] this again can be done at low computational cost, indeed, at only slightly more than twice the cost of the one-column update. Intuitively, updating two columns at a time should result in more effective iterations and, on the average, in convergence of $\det(X)$ to a larger value—though neither approach guarantees that the global maximum will be achieved. Further, updating two columns at a time makes it possible to handle prescribed eigenvalue sets that include complex conjugate pairs, while Method 0 of [9] is restricted to real prescribed eigenvalues.

Specifically, given a nonsingular matrix X satisfying $x_i \in \mathcal{S}_i$, the updates in Method 0 of [9] and Algorithm 5.1a of [11] are effected as follows. In the former, the updated x_i is obtained by maximizing, over the unit sphere in $\mathcal{S}_i$, the inner product $x_i^T u_i$, where u_i is orthogonal to all other x_ℓ's. The bulk of the computation consists in constructing such u_i, which can be done by means of a rank one update of a QR factorization. In the latter, when λ_i and λ_j are real (and x_i and x_j are constrained to be real as well), the updated x_i and x_j are obtained by maximizing, over the unit spheres in $\mathcal{S}_i$ and $\mathcal{S}_j$, respectively, the bilinear form $x_i^T U_{ij} x_j$, where U_{ij} is the dyad $uv^T - vu^T$, where u and v form an orthonormal basis for the orthogonal complement of the subspace spanned by all other real x_ℓ's and the real and imaginary part of the complex x_ℓ. Here, the bulk of the computation consists in constructing the basis (u, v), which can be done by means of a rank *two* update of a QR factorization. The maximization of $x_i^T U_{ij} x_j$ merely involves computation of the top left-right singular vector pair of the rank-two matrix $S_i^T U_{ij} S_j$, where S_i and S_j are matrices whose columns form orthonormal bases for the subspaces $\mathcal{S}_i$ and $\mathcal{S}_j$. When λ_i and λ_j form a complex conjugate pair (and x_i and x_j are constrained to be complex conjugate as well), the computation is similar, but $S_i^T U_{ij} S_j$ is now replaced by $S_i^* U_i S_i$, with U_i being the dyad $u\bar{u}^T - \bar{u}u^T$, where the real and imaginary parts of u form an orthonormal basis for the orthogonal complement of the subspace spanned by the real and imaginary part of the other complex x_ℓ, and by all real x_ℓ. Finally (and this is what distinguishes Algorithm 5.1a of [11] from Algorithm 5.1 in the same paper), the iteration acts in turn on *all* $(r(r-1)/2$, where r is the number of prescribed real eigenvalues) pairs of real columns of X; complex conjugate column pairs are acted upon with a higher repetition frequency in such a way that every column is acted upon as often as any other.

III. NUMERICAL RESULTS

This section presents some comparative results on the performance of the functions `robpole` and `place` (MATLAB) and `pass` (SLICOT). The numerical results have been obtained on an Intel Pentium 4 computer at 3 GHz, with 1 GB RAM, with the relative machine precision $\epsilon \approx 2.22 \times 10^{-16}$, using Windows XP (Service Pack 2) operating system, Compaq Visual Fortran V6.5 compiler and MATLAB 7.0.4.365 (R14) Service Pack 2. The SLICOT-based MATLAB executable MEX-function called by `pass` has been built using MATLAB-provided optimized LAPACK and BLAS subroutines.

Robustness (insensitivity of eigenvalues to variations in the entries of $A + BF$) of the closed-loop systems constructed by `place` and `robpole` are compared, as are the speed and accuracy of the three solvers. The use by `pass` of an executable MEX-function, and the fact that `pass` does not ensure robust results, make the timing results less comparable. Still, the timing/accuracy results obtained with `pass` are included to serve as a reference for the fastest available algorithm, which often works quite well concerning accuracy, and sometimes, robustness.

While the algorithm in `pass` is non-iterative, in `place`, execution stops after five "sweeps" have been completed, where one sweep corresponds to updating once every column of the X matrix. As for `robpole` a "maximum number of sweeps" input argument is at the user's disposal. However, as per the process outlined at the end of section II above, each `robpole` sweep corresponds to $r - 1$ (or 1 when $r = 1$), rather than one, updates of each column of X, where r is the number of real prescribed eigenvalues. Accordingly, except as otherwise specified, when attempting to compare the performances of `place` and `robpole`, we set the `numsweep` input parameter of `robpole` to $\max\{1, \mathrm{round}(5/\max\{r-1, 1\})\}$. (Note that this is a rough approximation; accordingly, our results should be taken with a grain of salt.)

The `toler` input argument in robpole (which allows the user to request that execution be terminated when little progress is made) was set to zero, i.e., that feature was turned off. A "tolerance" input argument in `pass` is used to decide whether or not the given matrix pair (A, B) is to be deemed controllable. In our tests, were used the default value of $n\epsilon \max(\|A\|_1, \|B\|_1)$; all our test matrix pairs were deemed controllable. Finally, in order to be comparable with the other two solvers, `pass` was forced to assign *all* controllable poles, by setting the optional scalar input argument `alpha` to `-Inf`. (We used the "continuous-time" mode.) Indeed, the function `pass` can assign part of the poles: `alpha` specifies the maximum admissible value for real parts of the eigenvalues of A which will not be modified by the eigenvalue assignment algorithm.

To assess the robustness of the closed-loop system, $\mathrm{cond}(X)$ (in the 2-norm) was computed and plotted for `place` and `robpole`, obtained via the MATLAB commands `[X,Lr] = eig(A+B*F)` for `robpole`, and `[X,Lm]`

= eig(A-B*F) for place. The plots display "relative" values, i.e., cond(X) divided by the maximum of its values obtained for place and robpole, for each system. Smaller values show better robustness.

The accuracy of the results is measured by the relative errors of the computed poles L?, with respect to the desired poles L, obtained using formulas of the form

```
err? = norm(L - cmpoles(L,L?)) ...
       /max(1,norm(L));
```

where ? stands for s, m, and r, denoting SLICOT pass, MATLAB place, and robpole, respectively, and cmpoles sorts the complex entries of the vector L? to be in the order of the entries of L.

Test sets 1 and 2 below use A and B with entries distributed according to $N(0,1)$. Test set 1 includes examples of small size, while Test set 2 includes examples of larger size. The poles to be assigned are also $N(0,1)$-distributed. Specifically, the system matrices and the desired poles (L) were obtained using the following MATLAB commands

```
A = randn(n);  B = randn(n,m);
L = randn(n,1);
nc = 2*floor(floor((n+2)/2)*rand);
for ii = 1 : 2 : nc,
    L(ii)   = L(ii) + i*L(ii+1);
    L(ii+1) = conj( L(ii) );
end
```

Note that L contains nc complex poles and n − nc real poles, and the nc values are also chosen randomly, uniformly over even integers from 0 to n. For A, B and L, we also tried using the MATLAB function rand, which generates numbers uniformly distributed in the (0,1) interval, and obtained similar results.

The tests were carried out with various values of n and m, always satisfying $1 < m < n$. Indeed, when $m = n$, perfect robustness ($X = I$) is achieved by both place and robpole, and when $m = 1$, there is only one F that makes the requisite assignment, so no robustification is possible.

A. Test set 1

Test set 1 includes examples of small size, with n = 3 : 10, and m = 2 : n-1, for a total of 36 data sets. Fig. 1, 2, and 3 show results obtained by taking the means for 25 different generated systems for each pair of values (n, m).

Fig. 1 shows the relative values of cond(X) for place and robpole, revealing that robpole often produces designs of better robustness than place (in slightly less time, as per Fig. 2). Fig. 2 depicts the ratios of the execution times for place and robpole to the execution times for pass. (Zero timing values for pass have been set to 0.01 seconds.) The algorithm implemented in pass does not perform any optimization. Consequently, it is the fastest algorithm, and sometimes it can be significantly faster than the other two algorithms. Note that the execution times of place and robpole are quite close to each other, as should be expected since, as explained above, we set the

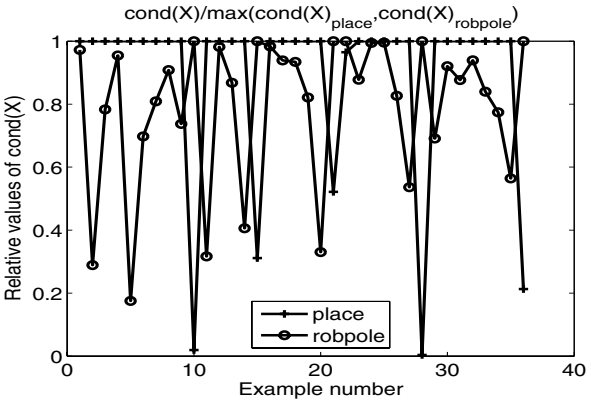

Fig. 1. Test set 1: Relative mean values of the 2-norm condition numbers of X.

numsweep input parameter to robpole in such a way that place and robpole always roughly effect the same number of column updates.

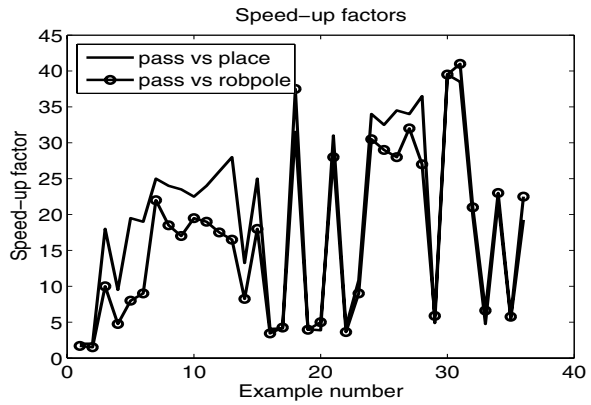

Fig. 2. Test set 1: Mean values of the speed-up factors.

Fig. 3 shows the decimal logarithms of the relative errors of the assigned poles for the three functions, pass, place, and robpole. For these random examples, pass and robpole are always highly accurate, with robpole typically even more accurate than pass. As for place, it is usually as accurate as robpole but is surprisingly inaccurate in a small number of cases.

While Fig. 2 adequately compares place and robpole, it is unfair to both of them as far as comparison with pass is concerned, since pass makes no attempt at robustification of the design. A more meaningful timing comparison can be made between pass and robpole, by making use of the robpole option to stop computation as soon as a nonsingular X has been achieved, i.e., as soon as some valid feedback matrix F is obtained. The results are shown in Fig. 4. The speed-up factor is rather small, and robpole is actually faster than pass for 15 examples.

Fig. 5 shows the relative accuracies of the three codes when robpole execution is terminated as soon as a nonsingular X is obtained. The place data points are the same as in Fig. 3. Note that robpole displays very poor accuracy

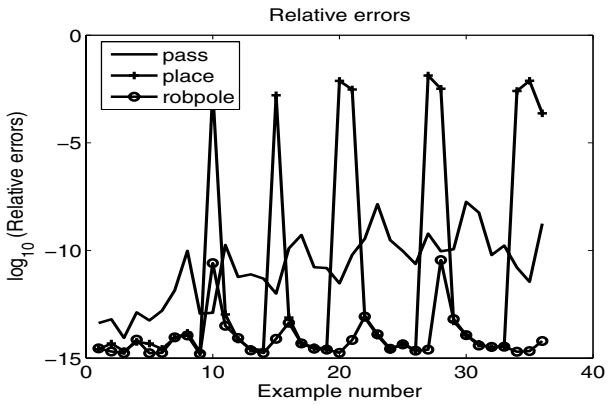

Fig. 3. Test set 1: Mean values of the decimal logarithms of the relative errors of the assigned poles.

on five of the nine problems on which `place` (at the end of five sweeps) was somewhat inaccurate, as well as on one problem on which `place` was highly accurate (problem 6). This is somewhat surprising, since `robpole` (like `place`) is supposed to reach full accuracy as soon as a nonsingular X is obtained. (We will look into resolving this `robpole` issue by the time of the conference.)

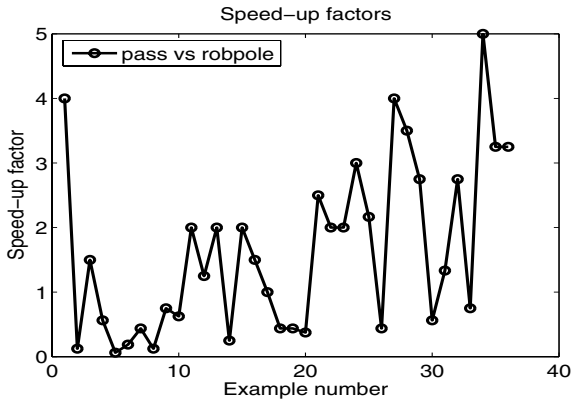

Fig. 4. Test set 1: Mean values of the speed-up factors, with `robpole`'s execution interrupted as soon as a nonsingular X is obtained.

B. Test set 2

The second set of tests also uses randomly generated (A, B) pairs, and poles L, using an $N(0, 1)$ distribution, but for larger values of n and m, namely $n = 20 : 10 : 100$, and $m = 10 : 10 : n - 10$. Therefore, 45 systems are generated. For each pair (n, m), only one system is generated.

Fig. 6 shows the relative values of cond(X) for `place` and `robpole`. It appears that, in most cases, `robpole` yields feedbacks that are more robust than those produced by `place`.

We do not report timing comparisons for this second data set. The reason is that `robpole` is always at a disadvantage in this respect, for problems of such size, because even a single `robpole` sweep entails a much larger number of column updates than five `place` sweep. A comparison of relative accuracies is of possible interest; such comparison is

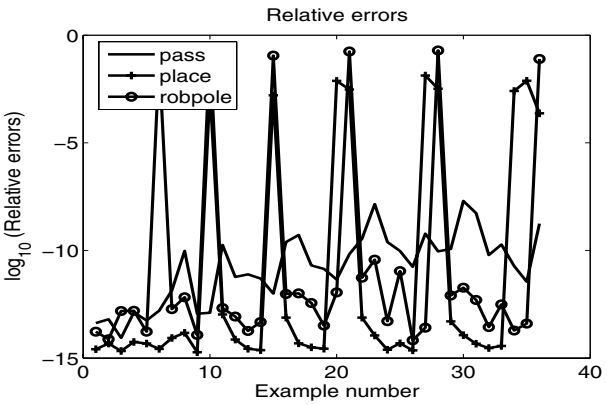

Fig. 5. Test set 1: Mean values of the decimal logarithms of the relative errors of the assigned poles, with `robpole`'s execution interrupted as soon as a nonsingular X is obtained.

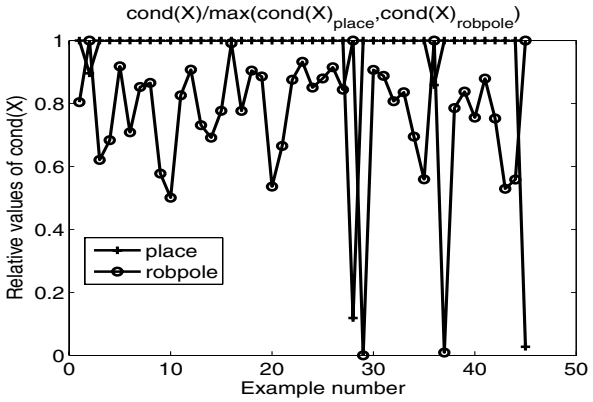

Fig. 6. Test set 2: Relative values of the 2-norm condition numbers of X.

reported in Fig. 7, which shows, for `place` and `robpole`, qualitatively similar results to those of Fig. 3. On the other hand, rather surprisingly, `pass` shows rather poor accuracy on this data set. The reason could be the increase of the number of eigenvalue interchanges involved in the underlying algorithm of `pass`; although this process is backward stable, the accuracy of the results could diminish for large problems. This topic deserves further investigation.

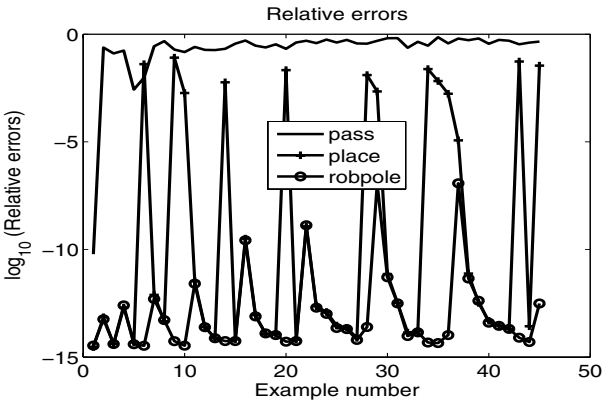

Fig. 7. Test set 2: Decimal logarithms of the relative errors of the assigned poles.

C. Test set 3

A third set of tests used the examples for algebraic Riccati equations (ARE) from the SLICOT continuous- and discrete-time ARE (CARE/DARE) benchmark collections [17], [18]. Most of these examples have small order, but could be very ill-conditioned. There are 21 CARE and 19 DARE examples in these collections. CARE example 4.4 ($n = 421$, $m = 211$) has been preliminarily scaled. With different parameter values, chosen as described in [17], [18], 35 and 25 experiments have been performed with CARE and DARE examples, respectively. In order to have reasonable locations, the desired poles have been set equal to the corresponding optimal closed-loop poles, found by solving the associated AREs using MATLAB function `care`. For this test set, `robpole` was run for five of its (long) sweeps. (This was due to the impending CACSD paper submission deadline.)

Fig. 8 shows the relative values of cond(X) for the 10 CARE examples with $1 < m < n$. Note that on problems 4 and 5, the `place` cond(X) is very large (more than 100 times larger than `robpole`'s). Similarly, Fig. 9

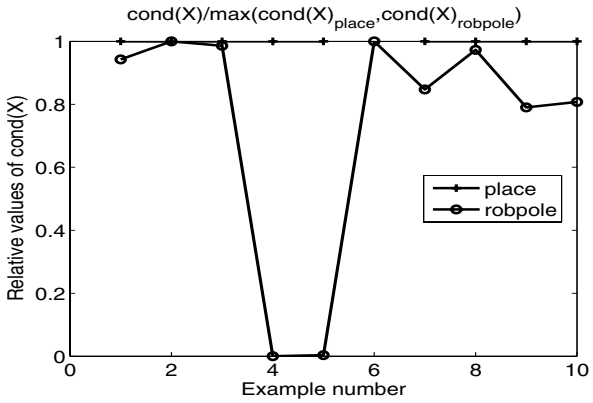

Fig. 8. The relative values of the 2-norm condition numbers of X for the 10 problems from CARE benchmark collection with $1 < m < n$.

depicts the ratios of the execution times for `place` and `robpole` to the execution times for `pass` for those 10 CARE examples. (The large `robpole` values should be discounted, as they are due to our running five full sweeps of `robpole` on this test set. This is also the case with the last DARE problem on Fig. 12.) Fig. 10 shows the decimal logarithms of the relative errors of the assigned poles.

Similarly, Fig. 11 shows the relative values of cond(X) for the 9 problems from DARE benchmark collection with $1 < m < n$, and Figs. 12 and 13 show the speed-up factors, and decimal logarithms of the relative errors of the assigned poles for `pass`, `place`, and `robpole` for those 9 DARE problems.

IV. CONCLUSIONS

Results of a numerical comparison of robust pole assignment codes `place` and `robpole`, and of (non-robust) pole assignment code `pass` were reported. As expected, a dramatic improvement of the robustness of the closed-loop system results when one of the two robust pole assignment

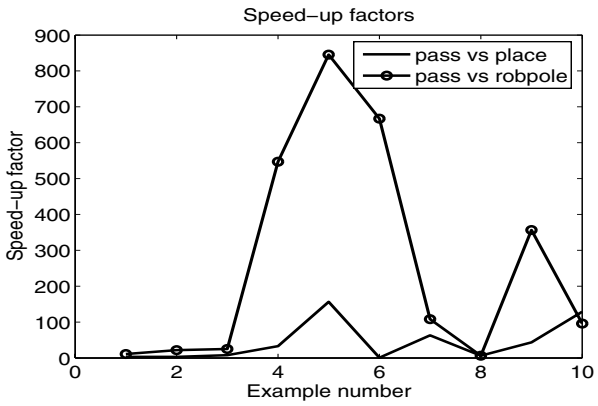

Fig. 9. The speed-up factors for the 10 problems from CARE benchmark collection with $1 < m < n$.

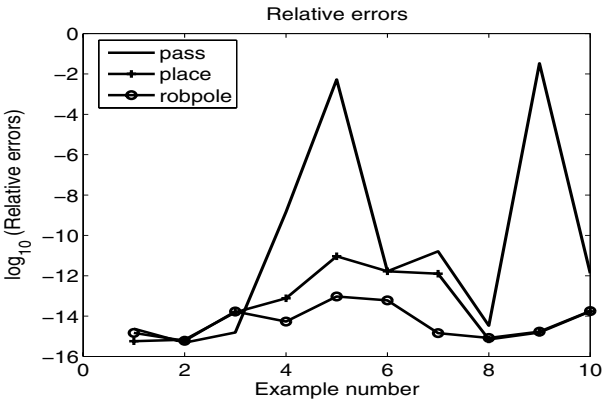

Fig. 10. Decimal logarithms of the relative errors of the assigned poles for the 10 problems from CARE benchmark collection with $1 < m < n$.

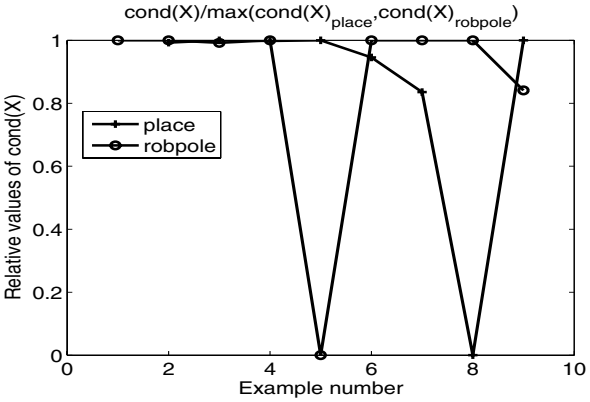

Fig. 11. The relative values of the 2-norm condition numbers of X for the 9 problems from DARE benchmark collection with $1 < m < n$.

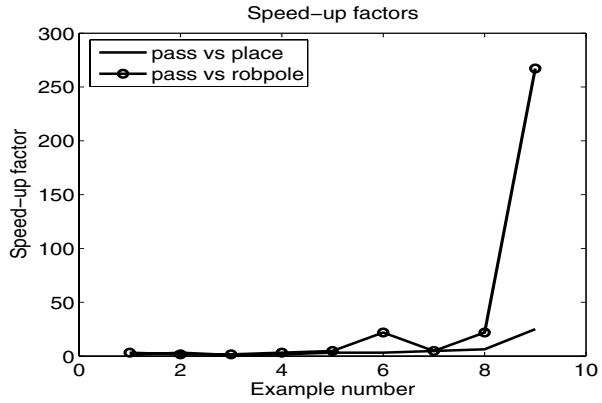

Fig. 12. The speed-up factors for the 9 problems from DARE benchmark collection with $1 < m < n$.

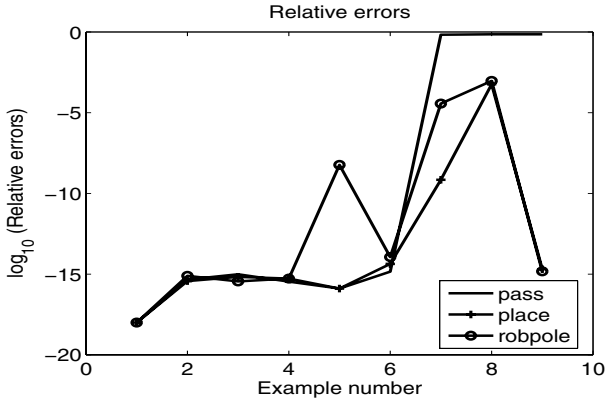

Fig. 13. Decimal logarithms of the relative errors of the assigned poles for the 9 problems from DARE benchmark collection with $1 < m < n$.

codes, rather than `pass`, is used. Further, the design produced by `robpole` is often significantly more robust than that produced by `place`. Timing-wise, `pass` is typically faster than `place` and `robpole`, even when `robpole`'s execution is stopped as soon as a valid (non-robust) feedback is obtained. Finally, interestingly, both `pass` and `place` are often rather inaccurate in that the poles of the closed-loop systems they produce are often noticeably different from the prescribed values.

REFERENCES

[1] V. Gourishanker and K. Ramer, "Pole assignment with minimum eigenvalue sensitivity to plant parameter variations," *Int. J. Control*, vol. 23, pp. 493–504, 1976.

[2] K. Ramer and V. Gourishanker, "Utilization for the design freedom of pole assignment feedback controllers of unrestricted rank," *Int. J. Control*, vol. 24, pp. 424–430, 1976.

[3] K. R. Cavin and S. P. Bhattacharyya, "Robust and well-conditioned eigenstructure assignment via Sylvester's equation." American Control Conference, 1982, pp. 1053–1057.

[4] A. Dickman, "On the robustness of multivariable linear feedback system in state space representation," *IEEE Trans. Automat. Contr.*, vol. 32, pp. 407–410, 1987.

[5] J. G. Sun, "On numerical methods for robust pole assignment in control system design," *J. Computational Math.*, vol. 5, pp. 119–134, 1987.

[6] L. Keel, S. Bhattacharyya, and J. Howze, "Robust control with structured perturbations," *IEEE Trans. Automat. Contr.*, vol. 33, pp. 68–78, 1988.

[7] R. Byers and S. Nash, "Approaches to robust pole assignment," *Int. J. Control*, vol. 49, pp. 97–117, 1989.

[8] L. H. Keel and S. P. Bhattacharyya, "State space design of low-order stabilizers," *IEEE Trans. Automat. Contr.*, vol. 35, pp. 82–86, 1990.

[9] J. Kautsky, N. K. Nichols, and P. Van Dooren, "Robust pole assignment in linear state feedback," *Int. J. Control*, vol. 41, pp. 1129–1155, 1985.

[10] *The* MATLAB *Control Toolbox, Version 5*, The MathWorks, Inc., 3 Apple Hill Drive, Natick, MA, 01760–2098, 2000.

[11] A. L. Tits and Y. Yang, "Globally convergent algorithms for robust pole placement by state feedback," *IEEE Trans. Automat. Contr.*, vol. AC–41, no. 10, pp. 1432–1452, 1996.

[12] P. Benner, V. Mehrmann, V. Sima, S. Van Huffel, and A. Varga, "SLICOT — A subroutine library in systems and control theory," in *Applied and Computational Control, Signals, and Circuits*, B. N. Datta, Ed. Birkhäuser, Boston, 1999, vol. 1, chapter 10, pp. 499–539.

[13] S. Van Huffel, V. Sima, A. Varga, S. Hammarling, and F. Delebecque, "High-performance numerical software for control," *IEEE Control Syst. Mag.*, vol. 24, no. 1, pp. 60–76, Feb. 2004.

[14] V. Sima, "SLICOT-based advanced automatic control computations," in *Advances in Automatic Control*, The Kluwer International Series in Engineering and Computer Science, M. Voicu, Ed. Boston/Dordrecht/New York/London: Kluwer Academic Publishers, 2004, pp. 337–349.

[15] SLICOT, "The Control and Systems Library SLICOT", 2005. [Online]. Available: `http://www.slicot.org`

[16] A. Varga, "A Schur method for pole assignment", *IEEE Trans. Automat. Contr.*, vol. AC–26, no. 2, pp. 517–519, 1981.

[17] J. Abels and P. Benner, "CAREX—A collection of benchmark examples for continuous-time algebraic Riccati equations (Version 2.0)," Katholieke Universiteit Leuven, ESAT/SISTA, Leuven, Belgium, SLICOT Working Note 1999-14, Nov. 1999. [Online]. Available: `http://www.slicot.de/REPORTS/SLWN1999-14.ps.gz`.

[18] ——, "DAREX—A collection of benchmark examples for discrete-time algebraic Riccati equations (Version 2.0)," Katholieke Universiteit Leuven, ESAT/SISTA, Leuven, Belgium, SLICOT Working Note 1999-16, Dec. 1999. [Online]. Available: `http://www.slicot.de/REPORTS/SLWN1999-16.ps.gz`.