A Numerically Stable Algorithm for Pole Placement of Single/Input and Multi/Input Systems

François E. Cellier, Ph.D.
Associate Professor
Dept. of Electrical & Computer Engineering
University of Arizona
TUCSON, AZ 85721

Abstract

One of the standard problems in controller design of linear systems is the so-called pole placement problem. Although there are meanwhile a couple of algorithms for this problem on the market, the most common approach (at least for single-input systems) is the one suggested by Jürgen Ackermann[1]. As it was recently shown by Alan Laub *et alia* [2], this algorithm is numerically harmful. For systems of larger than 10^{th} order, double precision is required on a 32-bit machine (e.g. VAX); for systems larger than about 15^{th} order, the algorithm fails altogether. Newer algorithms making use of a numerically more stable transformation (e.g. Hessenberg form) are behaving better, but were shown to fail as well for systems of larger than approximately 20^{th} to 25^{th} order. The authors therefore suggested (without proof) that the pole placement problem is intrinsically ill-posed.

In this paper, we present an algorithm which shows numerically stable behavior for much larger systems, thus contradicting the above conjecture. This algorithm is closely related to the one suggested by Roppenecker[3]. It therefore lends itself equally well to single-input and multi-input systems. In case of several inputs, the additional freedom is used to optimize the numerical behavior of the algorithm. However even in the single-input case, the algorithm behaves numerically much better than any other pole placement algorithm that we are aware of.

Numerical Resolution of the Feedback Matrix

The numerical machine resolution of a computer is usually defined in the following sense:

```
eps = 1;
WHILE 1+eps>1,...
  eps = eps/2;...
END
eps = 2*eps;
```

It is now interesting to ask ourselves how well the poles of the closed-loop system can possibly be determined as a function of the previously defined machine resolution *eps*. The following algorithm may answer this question:

```
// [a,b,lambda,res,rrel]=resol(n)
// Calculate the Resolution of the Feedback Matrix
DEFF csrt
a = RAND(n);
b = RAND(n,1);
k = RAND(1,n);
aa = a - b*k;
lambda = EIG(aa);
lambda = CSRT(lambda);
bb = aa + 100*EPS*RAND(aa);
p = EIG(bb);
p = CSRT(p);
res = NORM(lambda-p,'INF')/100;
rrel = res/EPS;
RETURN
```

It is coded in CTRL_C [4], a command driven interactive program for control system design. CTRL_C is basically a superset of the well known program MATLAB [5] by Cleve Moler. We prefer to present our algorithms in CTRL_C rather than in MATLAB for its increased readability.

This algorithm first defines the system matrix *(a)*, the input vector *(b)*, and the feedback vector *(k)* as random matrices. Then, the closed-loop system matrix *(aa)* is computed, and its eigenvalues are evaluated which are the poles of the closed-loop system. These are then sorted in *CSRT* into ascending order of their real values whereby positive imaginary parts always precede negative imaginary parts. Then, the closed-loop matrix is perturbed *(bb)*. For that purpose, we add random values to each element of the matrix. These random elements are scaled by *100*eps*, as it makes little sense to multiply a number smaller than 1 by *eps* itself, and add it up to something lerge. Then, the new eigenvalues

are computed and sorted. Finally, we compute an infinity norm of the difference of the old and the new poles, and divide again by *100* for normalization purposes. Obviously, we cannot expect any poles to lie closer to the desired poles, independently of the algorithm we use. Thus, *res* is really the absolute numerical resolution, and *rrel* is the relative numerical resolution of the feedback matrix.

We calculated this resolution on a VAX for different system orders, and obtained the following results:

System Order	RES	RREL
5	$5 \cdot 10^{-17}$	3.62
10	$6.88 \cdot 10^{-17}$	4.96
15	$1.02 \cdot 10^{-16}$	7.36
20	$1.51 \cdot 10^{-16}$	10.9
25	$1.89 \cdot 10^{-16}$	13.62
30	$2.30 \cdot 10^{-16}$	16.64
35	$2.52 \cdot 10^{-16}$	18.16

Of course, these numbers depend on the random number generator but it is obvious that, with increased order, the resolution of the feedback matrix decreases only slightly. Thus, the concept of state feedback itself is numerically sound.

Pole Placement According to Ackermann

The pole placement algorithm can be expressed in the following way:

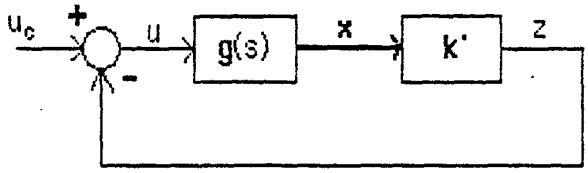
$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{b}u$$

$$g(s) = \frac{x(s)}{d(s)} = (s\mathbf{I} - \mathbf{A})^{-1}\mathbf{b} = \frac{\mathbf{l}(s)}{d(s)}$$

$$= \frac{1}{s^n + d_1 s^{n-1} + \ldots + d_n} \cdot \begin{bmatrix} l_{11}s^{n-1} + \ldots + l_{1n} \\ \ldots \\ l_{n1}s^{n-1} + \ldots + l_{nn} \end{bmatrix}$$

$$d(s) = |s\mathbf{I} - \mathbf{A}| = s^n + d_1 s^{n-1} + \ldots + d_n$$

The feedback loop can be expressed in the following way:



$$z(s) = k'x(s)$$

$$u = u_c - k'x$$

$$\dot{x} = (A - bk')x + bu_c = A_c x + b u_c$$

The desired pole locations can be expressed as:

$$d_c(s) = |sI - A_c| = s^n + a_1 s^{n-1} + \ldots + a_n$$

Thus, we find:

$$\frac{z(s)}{u(s)} = k'g(s) = \frac{k'l(s)}{d(s)}$$

$$\frac{z(s)}{u_c(s)} = \frac{k'l(s)}{d(s) + k'l(s)} = \frac{l_c(s)}{d_c(s)}$$

that is:

$$d_c(s) = d(s) + k'l(s)$$

By comparing the coefficients, we find:

$$L'k = a - d$$

where:

$$L = \begin{bmatrix} l_{11} & \cdots & l_{1n} \\ \vdots & & \vdots \\ l_{n1} & \cdots & l_{nn} \end{bmatrix}, \quad a = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}, \quad d = \begin{bmatrix} d_1 \\ \vdots \\ d_n \end{bmatrix}$$

and therefore:

$$k = (L')^{-1}(a-d)$$

The exact notation of this algorithm was taken from Mansour[6]. The coefficients of the transfer function vector can e.g. be found by transforming the system into controller-canonical form. A CTRL_C function which implements this algorithm is the following:

```
// [k]=poll(a,b,lambda)
// Calculates State Feedback of SI-System by Pole Placement
DEFF ctran
flag = 'FALSE';
GLOB(flag);
[n,m] = SIZE(lambda);
IF m<>1, ...
 IF n<>1, ...
  DISPLAY('Poles must form a vector'), ...
  flag = 'TRUE'; ...
  RETURN, ...
 END, ...
END
nn = n*m;
[n,n] = SIZE(a);
IF nn<>n, ...
 DISPLAY('Number of poles inconsistent with system order'), ...
 flag = 'TRUE'; ...
 RETURN, ...
END
caux = EYE(a);
[anew,bnew,cnew] = CTRAN(a,b,caux);
IF flag='TRUE', ...
 RETURN, ...
END
dd = -anew(n,n:-1:1)';
aa = REAL(POLY(lambda));
aa = aa(2:n+1);
l = cnew(:,n:-1:1);
f = l'\(aa - dd);
k = REAL(f');
RETURN
```

First, we make sure that the input *lambda* (containing the desired poles) is really a vector. Then, we check that there are as many poles specified as there are states in the system. Function *ctran* transforms the SIMO-system:

$$\left|\begin{array}{l}\dot{x} = Ax + bu \\ y = x\end{array}\right|$$

into controller-canonical form. In this new form, the matrix *cnew* contains the coefficients of the $\mathcal{L}$ matrix (columns in reverse order), and the last row of the matrix *anew* contains the coefficients of the open-loop

denominator polynomial *d(s)* (also in reverse order). The rest of the algorithm is self-explanatory. Function *POLY* calculates the coefficients of the polynomial whose roots were stored in *lambda*. The backslash operator denotes "left-division" (that is: multiplication from left by the inverse, calculated by use of Gaussian elimination).

The transformation is done by first calculating the controllability matrix:

$$Q_c = [\, b \,,\, Ab \,,\, A^2b \,,\, \ldots \,,\, A^{n-1}b \,]$$

then using the last row of its inverse for the construction of an "observability matrix" of the modified system (A,q):

$$T = [\, q \,;\, qA \,;\, qA^2 \,;\, \ldots \,;\, qA^{n-1} \,]$$

"," being used for concatenation from the right whereas ";" being used for concatenation from below. T is then used for the similarity transformation:

$$A_{new} = TAT^{-1} \quad , \quad b_{new} = Tb \quad , \quad c_{new} = cT^{-1}$$

A CTRL_C function for this purpose would be:

```
// [anew,bnew,cnew]=ctran(a,b,c)
// Transforms System into Controller-Canonical Form
DEFF contr
DEFF obser
flag = 'FALSE';
GLOB(flag);
qc = CONTR(a,b);
IF flag='TRUE', ...
 RETURN, ...
END
[n,m] = SIZE(b);
IF m<>1, ...
 DISPLAY('Algorithm only for single-input systems'), ...
 flag = 'TRUE'; ...
 RETURN, ...
END
[p,n1] = SIZE(c);
IF n1<>n, ...
 DISPLAY('C must have n columns'), ...
 flag = 'TRUE'; ...
 RETURN, ...
END
IF RANK(qc)<>n, ...
 DISPLAY('System is not controllable'), ...
```

```
 flag = 'TRUE'; ...
 RETURN, ...
END
qcin = INV(qc);
q = qcin(n,:);
t = OBSER(a,q);
tin = INV(t);
anew = t*a*tin;
bnew = t*b;
cnew = c*tin;
RETURN
```

which is pretty self-explanatory. The two functions *contr* and *obser* calculate the controllability matrix and observability matrix, resp.:

```
// [qc]=contr(a,b)
// Calculates Controllability Matrix
flag = 'FALSE';
qc = b;
aux = b;
[n1,n2] = SIZE(a);
IF n1<>n2, ...
 DISPLAY('A must be a square matrix'), ...
 flag = 'TRUE'; ...
 RETURN, ...
END
n = n1;
[n1,m] = SIZE(b);
IF n1<>n, ...
 DISPLAY('B must have n rows'), ...
 flag = 'TRUE'; ...
 RETURN, ...
END
FOR i=2:n, ...
 aux = a*aux; ...
 qc = [qc,aux]; ...
END
RETURN

// [qo]=obser(a,c)
// Calculates Observability Matrix
flag = 'FALSE';
qo = c;
aux = c;
[n1,n2] = SIZE(a);
IF n1<>n2, ...
 DISPLAY('A must be a square matrix'), ...
 flag = 'TRUE'; ...
 RETURN, ...
END
n = n1;
[p,n1] = SIZE(c);
```

```
IF n1<>n, ...
 DISPLAY('C must have n columns'), ...
 flag = 'TRUE'; ...
 RETURN, ...
END
FOR i=2:n, ...
 aux = aux*a; ...
 qo = [qo;aux]; ...
END
RETURN
```

which are easily understandable.

Numerical Behavior of Ackermann's Algorithm

As suggested by Alan Laub[2], we now analyze the numerical behavior of this algorithm. For that purpose, we execute in a sequence the previously described function *RESOL* and the function *CON1* for different values of the system order *n*.

```
// [err,erl]=con1(a,b,lambda,res)
// Calculates the Numerical Condition of POL1
DEFF csrt
DEFF pol1
flag = 'FALSE';
GLOB(flag);
k = POL1(a,b,lambda);
IF flag='TRUE', ...
 RETURN, ...
END
aa = a - b*k;
p = EIG(aa);
p = CSRT(p);
err = NORM(lambda-p,'INF');
erl = err/res;
RETURN
```

in which we calculate the closed-loop system matrix *(aa)* for the feedback found by function *POL1*. Then we compare its eigenvalues *(p)* with the desired eigenvalues *(lambda)*. *Err* is thus the absolute numerical resolution of the *POL1* algorithm, and *erl* is the numerical resolution relative to the best possible case *(res)*. As suggested by Alan Laub, the results were deplorable:

System Order	ERR	ERL
5	$8.35 \cdot 10^{-15}$	166
10	$1.22 \cdot 10^{-7}$	$1.77 \cdot 10^{9}$
15	0.0023	$2.20 \cdot 10^{13}$
20	Rank(Q_c) < n	

For larger system orders, the accuracy of this algorithm decays rapidly. At order 15, hardly any accuracy is left, and at order 20, the algorithm is unable to invert the controllability matrix, as it has (for numerical reasons) no longer full rank. Even the 10^{th} order system could only be treated due to the double precision arithmetics used by CTRL_C.

The pole placement algorithm built into CTRL_C *(PLACE)* failed on this problem altogether, issuing some lines of "trap" – whatever this may mean.

What went wrong in this algorithm? First, we had to defactorize our polynomials, and lateron factorize them again. This operation is known to be potentially harmful on larger order systems. However, even worse in our case was the computation of the controllability and modified observability matrices. If the original matrix A had the eigenvalues $\{s_1,...,s_n\}$, then the matrix A^{n-1} has the eigenvalues $\{s_1^{n-1},...,s_n^{n-1}\}$. That is: small eigenvalues have become even smaller, while large eigenvalues are getting yet larger. Also the condition of A^{n-1} has been worsened by roughly the power (n-1), and Q_c is hardly any better. We therefore must seek for an algorithm which does not commit either of these two sins.

Eigenstructure Approach

A number of authors[3,7,8,9,10] suggested to use the additional freedom in the pole placement of multi-input systems for (partial) eigenstructure selection. We shall show that this approach is fruitful, and amazingly even leads to a numerically much better behaving algorithm in the single-input case.

Given the MIMO-system:

$$\dot{x} = Ax + Bu$$

together with the feedback equation:

$$u = u_c + Kx$$

The closed-loop system is supposed to have the following properties:

$$\text{eigen}(A+BK) = [s_1, s_2, \ldots, s_n]$$

$$\text{modal}(A+BK) = [v_1, v_2, \ldots, v_n]$$

thus:

$$(A+BK)v_i = s_i v_i$$

or:

$$[\,(A-s_i I)\,,\, B\,] * [\,v_i\,;\,q_i\,] = 0$$

where:

$$q_i = Kv_i$$

that is, the vector $[\,v_i\,;\,q_i\,]$ must lie in the nullspace of the matrix:

$$S(s_i) = [\,(A-s_i I)\,,\, B\,]$$

After repeating this operation on all n poles, we can calculate:

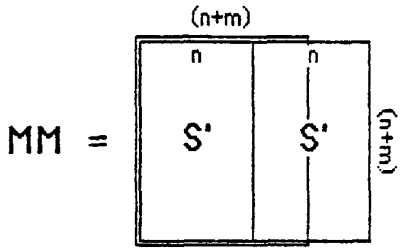
$$K = [\,q_1, q_2, \ldots, q_n\,]\,[v_1, v_2, \ldots, v_n]^{-1} = QV^{-1}$$

This algorithm works as long as the system is fully controllable, B has the maximum rank (m), and all desired poles are distinct (as otherwise V may be singular). Ackermann's algorithm does not require the poles to be distinct, but the numerical behavior of that algorithm gets even worse if the poles are placed on top of each other. We tried that algorithm on a 5^{th} order random system with poles located at [-1 ; -2 ; -3 ; -4 ; -5], and with all poles located at -1. In the first case, the resolution was roughly 10^{-11}, in the second case it was 10^{-3} only.

How can we implement our new algorithm in terms of a CTRL_C function? First, we have to evaluate the nullspace of the rectangular matrix $S(s_i)$. The dimensions of that matrix are $n \times (n+m)$. If none of the

desired poles coincides with a pole of the open-loop system, the rank of that matrix is certainly n, but even if such a coincidence exists, the rank of S shall usually still remain n due to the concatenation with the B-matrix; thus, the new algorithm does not require the desired poles to be different from the poles of the open-loop system as some of the previously described algorithms do.

We now build the following matrix:

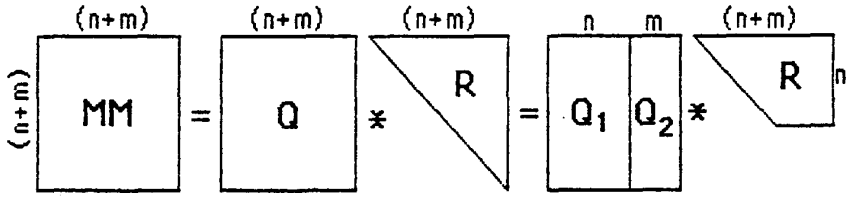


$$MM = [\, S' \,,\, S' \,]$$
$$MM = MM(:, 1{:}n{+}m)$$

This square matrix MM has certainly still the same Rank(MM)=n, and it spans the same space as S. Now we perform a QR decomposition on that matrix MM:

$$[\, Q \,,\, R \,] = QR(MM)$$

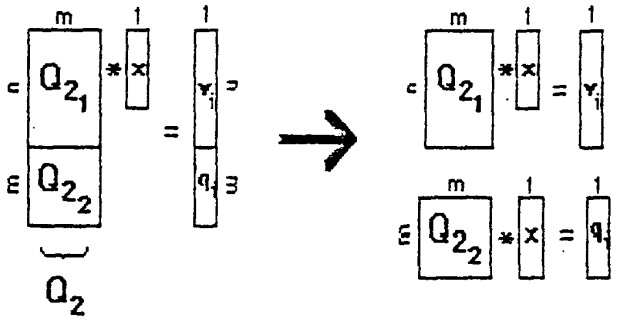
which creates two matrices Q and R such that MM=Q*R, where Q is a unitary matrix, and R is upper triangular with diagonal elements in decreasing order. As Q has Rank(Q)=(n+m), obviously the Rank(R)=n, that is, R has m rows equal to zero.



$$MM = Q*R = Q_1*R$$

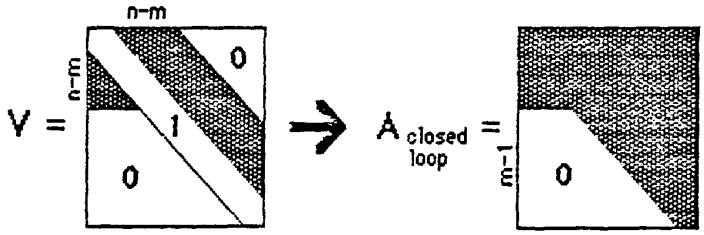
that is, Q_1 spans the same space as MM and thus as S, whereas Q_2 is an orthogonal extensions, and thus the nullspace of S. We could have achieved the same also by use of a singular value decomposition, but the much "cheaper" QR algorithm is very well suited for our purpose, and it is numerically about as sound as SVD.

There must exist a vector x, such that $Q_2 x = [\, v_i \,;\, q_i \,]$, that is:



Only m components of the v_i-vector can be chosen freely. This is in agreement with the previously found statements that in the single-input case the pole placement problem has a unique solution, whereas the system with n inputs allows for free selection of the entire modal matrix as well.

We select v_i as follows:



In this way, we try to decouple the solution as much as possible. In our algorithm, we cancel those rows of Q_{21} and v_i which we cannot choose freely, and solve the remaining (non-singular) system for x. Then, we use x to determine q_i. A CTRL_C function which implements this algorithm is as follows:

```
// [k]=polm(a,b,lambda)
// Pole Placement for MI-System with m Linearly Independent Inputs
DEFF qim
flag = 'FALSE';
[n1,n2] = SIZE(a);
IF n1<>n2,...
 DISPLAY('A must be a square matrix'),...
 flag = 'TRUE';...
 RETURN,...
END
n = n1;
[n1,m] = SIZE(b);
IF n1<>n,...
```

```
 DISPLAY('B must have n rows'), ...
 flag = 'TRUE'; ...
 RETURN, ...
END
IF RANK(b)<m, ...
 DISPLAY('B-matrix must have full rank'), ...
 flag = 'TRUE'; ...
 RETURN, ...
END
[n1,m1] = SIZE(lambda);
IF m1<>1, ...
 IF n1<>1, ...
  DISPLAY('Poles must form a vector'), ...
  flag = 'TRUE'; ...
  RETURN, ...
 END, ...
END
nn = n1*m1;
IF nn<>n, ...
 DISPLAY('Number of poles inconsistent with system order'), ...
 flag = 'TRUE'; ...
 RETURN, ...
END
[i2,h] = SORT(REAL(lambda));
h = lambda(i2);
FOR i=1:n-1, ...
 d = NORM(h(i+1)-h(i)); ...
 IF d<10*EPS, ...
  DISPLAY('Poles must be distinct'), ...
  flag = 'TRUE'; ...
  RETURN, ...
 END, ...
END
v = EYE(a);
qq = ONES(m,1);
FOR i=1:n, ...
 li = lambda(i); ...
 vi = v(:,i); ...
 [qi,vvi] = QIM(a,b,n,m,li,vi,i); ...
 v(:,i) = vvi; ...
 qq = [qq,qi]; ...
END
qq = qq(:,2:n+1);
f = qq/v;
k = REAL(f);
RETURN

// [qi,vvi]=qim(a,b,n,m,li,vi,i)
// Pole Placement with m Inputs
// Auxilliary Macro
// Calculates the Null-Space and the qi-Vector
s = [(a-li*EYE(a)),b];
mm = [s',s'];
mm = mm(:,1:n+m);
```

```
[q,r] = QR(mm);
q2 = q(:,n+1:n+m);
q21 = q2(1:n,:);
k1 = i - n + m;
k2 = i;
IF k1<1, ...
 k2 = k2 - k1 + 1; ...
 k1 = 1; ...
END
vii = vi(i);
qq21 = q21(i,:);
IF k1>1, ...
 vii = [vi(1:k1-1);vii]; ...
 qq21 = [q21(1:k1-1,:);qq21]; ...
END
IF k2<n, ...
 vii = [vii;vi(k2+1:n)]; ...
 qq21 = [qq21;q21(k2+1:n,:)]; ...
END
xx = qq21\vii;
vvi = q21*xx;
q22 = q2(n+1:n+m,:);
qi = q22*xx;
RETURN
```

Of course, in the single-input case, there does not remain any area of zero elements, and the closed-loop system matrix is full. However, the problem of finding the feedback matrix is reduced to n QR-decompositions and $(n+1)$ linear system solutions. Thus, we have justified hope to face a better numerical behavior of this algorithm.

Numerical Behavior of Eigenstructure Method

We tried this algorithm by use of the following CTRL_C test function:

```
// [err,er1]=con3(a,b,lambda,res)
// Calculates the Numerical Condition of POLM
DEFF csrt
DEFF polm
flag = 'FALSE';
GLOB(flag);
k = POLM(a,b,lambda);
IF flag='TRUE', ...
 RETURN, ...
END
aa = a + b*k;
p = EIG(aa);
p = CSRT(p);
err = NORM(lambda-p,'INF');
```

```
erl = err/res;
RETURN
```

The results were as follows:

System Order	ERR	ERL
5	$1.53 \cdot 10^{-16}$	3.05
10	$3.74 \cdot 10^{-16}$	5.44
15	$5.96 \cdot 10^{-16}$	5.84
20	$8.88 \cdot 10^{-16}$	5.87
25	$2.01 \cdot 10^{-15}$	10.68
30	$2.58 \cdot 10^{-15}$	11.17
35	$2.60 \cdot 10^{-15}$	10.35

We stopped at order 35, as the algorithm required already 21 min CPU time to execute on a VAX 11/750. However, it is obvious that this algorithm could easily be used for larger systems as well.

We also compared the number of floating point operations for Ackermann's algorithm and the eigenstructure approach. For low order systems, they were not much different. However, Ackermann's algorithm is roughly proportional to $9 \times \text{order}^3$, the other to $100 \times \text{order}^3$.

Summary

An algorithm has been shown which allows to compute the state feedback matrix of the pole placement problem for single-input and multi-input systems. This algorithm differs from those previously known in that its numerical behavior is much more stable, and therefore allows for much larger system orders to be treated.

As more and more controller design problems are solved today by use of digital computers, such numerical considerations are of utmost importance.

Acknowledgments

We would like to thank Alan Laub for posing this problem. It was certainly an investigation which was worthwhile looking into. We would also like to express our gratitude to Cleve Moler for suggesting the QR algorithm for the Nullspace evaluation.

References

J. Ackermann [1], «Entwurf durch Polvorgabe» («Design by Pole Placement»), *Regelungstechnik,* (6), 1977, pp. 173-179, and (7) 1977, pp. 209-215.

A. Laub [2] *et alia*, «Algorithms and Software for Pole Assignment by State Feedback», Proc. of the CACSD'85 Conference, Sta. Barbara CA, March 13-15, 1985.

G. Roppenecker [3], «Polvorgabe durch Zustandsrückführung» («Pole Assignment by State Feedback»), *Regelungstechnik* , (7), 1981, vol.29, pp. 228-233.

Systems Control Technology [4], «CTRL_C – A Language for the Computer-Aided Design of Multivariable Control Systems», 1901 Page Mill Road, P.O.Box 10180, Palo Alto CA 94303, 1983.

C. Moler [5], «MATLAB – A Matrix Laboratory, User's Guide», Dept. of Computer Science, University of New Mexico, Albuquerque NM, 1981.

M. Mansour [6], «Systemtheorie», Lecture Notes, Dept. of Automatic Control, Swiss Federal Institute of Technology, CH-8092 Zürich, Switzerland, 1984.

B.C.Moore [7], «On the Flexibility Offered by State Feedback in Multivariable Systems Beyond Closed Loop Eigenvalue Assignment», *IEEE Trans. Automatic Control*, (21), 1976, pp. 209-212.

B.C.Moore and G.Klein [8], «Eigenvector Selection in the Linear Regulator Problem: Combining Modal and Optimal Control», Proc. of the 7th IEEE Conference on Decision and Control, December 1976, Clearwater FL, pp. 214-215.

B.Porter and J.J.D'Azzo [9], «Closed-Loop Eigenstructure Assignment by State Feedback in Multivariable Linear Systems», *Int. Journal of Control*, (27), 1978, pp. 487-492.

B.Porter and J.J.D'Azzo [10], «Algorithm for Closed-Loop Eigenstructure Assignment by State Feedback in Multivariable Linear Systems», *Int. Journal of Control*, (27) 1978, pp. 943-947.