LQG-I/O - A CTRL-C Library for LQG Design

Nilsson, Bernt

1986

Document Version:
Publisher's PDF, also known as Version of record

CODEN: LUTFD2/(TFRT-7329)/1-26/(1986)

LQG-I/O
A CTRL-C library for LQG design

Bernt Nilsson

Department of Automatic Control
Lund Institute of Technology
August 1986

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Report

Date of issue
August 1986

Document Number
CODEN: LUTFD2/(TFRT-7329)/1-26/(1986)

Author(s)
Bernt Nilsson

Supervisor
Karl Johan Åström

Sponsoring organisation
STU Contract No 85-4809

Title and subtitle
LQG-I/O , a CTRL-C library for LQG design.

Abstract

This is a documentation of a CTRL-C library, LQG-I/O. It handles Linear Quadratic Gaussian Control design based on Input/Output approach.

A simple HELP-facility for CTRL-C libraries is also developed.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
25

Recipient's notes

Security classification

1. Introduction

This report is a documentation of a Ctrl-C library, LQGI/O . This stands for Linear Quadratic Gaussian (LQG) control design based on Input/Output approach (I/O). The library is based on the polynomial design in Chapter 12 in Åström and Wittenmark, Computer Controlled Systems.

In Ctrl-C there are already good facilities for LQG-synthesis based on the state space approach. LQGI/O is an alternative which has some good and new possibilities.

In this report a HELP-facility for CTRL-C-libraries is also developed.

2. LQGIO-library

The LQGIO-library contains macros for Optimal Control Design based on the Input/Output approach. A theoretical reference is chapter 12 in Åström and Wittenmark, Computer Controlled Systems. The main subjects for the library are Optimal Prediction, Minimum-Variance Control and Linear Quadratic Gaussian Control based on Input/Output approach. But to support this a number of other macros are also required.

Library Contents:

CLVAR	- Calculate variances in a closed loop system.
COVAR	- Calculate the Covariance Function of a ARMA-process.
CPL1	- Plots the COVAR.
MINVAR	- Calculate a Minimum-Variance Controller.
OPTPRED	- Calculate a Optimal d-step Predictor.
DIOPHANT	- Solve the Diophantine equation.
PRED	- Solve the predictor problem.
FACTOR	- Factorization of a polynomial.
LQGA	- Calculate a LQG-controller (I/O-approach)
LQGCF	- Same as LQGA, but can handle Common Factors.
COMFAC	- Find the Greatest Common Divisor.
LQGSPEC	- Calculate the P-polynomial in the LQG-problem.
PZPLOT	- Plots poles and zeros for a discrete time system
CLPZ	- Plots poles and zeros for a closed loop system

This library takes about 44 procent of the function buffer in CTRL-C. The HELP-facility, described in chapter 3, takes 16 procent and the macros only 28 procent of the buffer.

When building libraries like this, one should think about the size of the library. Its important not to build to big libraries, which makes it hard to have more then one library activated, in the function buffer, on the same time.

Small libraries gives you the opportunity to create and use your one favorite environment of macros.

A conclusion from the discussion above is that the library described in this paper in to big. One way to make this library smaller would be to make the HELP-macro as a separate function. An other is to divide the hole library in sublibraries.

Macro Description

Below each macro is described. What its doing and how it is done. Also which submacros it needs and a reference to the algorithm it uses. The macro codes you will find in Appendix I. All polynomials are in the forward shift operator, q.

[] = CLPZ(a,b,c,r,s,t)

Plots poles and zeros for a discrete time closed loop system in a singular diagram with the unit circle. Poles are represented by diagonal cross, zeros by octagons and noise zeros by squares. a,b and c are the process polynomials and r,s and t are the controller.
This macro uses PZPLOT.

[yvar,uvar] = CLVAR(a,b,c,var,r,s)

This macro computes the output variance, yvar, and control signal variance, uvar, of a closed loop system with the process polynomials, a,b and c, and the controller polynomials, r and s. The argument, var, is the noise variance.
It uses the COVAR-macro.

[a1,b1,a2m,a2p] = COMFAC(a,b)

Finds the greatest common divisor, a2, of a and b. a2m is the factor of a2 witch has its zeros on or outside the unit disc and a2p has its inside. a1 and b1 are the polynomials left after cancellation of a2 in a and b.

[ry,rye] = COVAR(a,c,kmax);

Uses the Yule-Walker equation for computing the covariance function, ry, and the cross covariance function, rye, for an ARMA-process, a and c. kmax is the time axes.
Developed by Mats Lilja.
Reference : Olbjer , TIDSSERIEANALYS , appendix

[] = CPL1(r);

A macro for plotting the covariance function, r, from the COVAR-macro.

[x,y] = DIOPHANT(a,b,c);

Solves the Diophantine equation, A X + B Y = C, where A,B,C,X and Y are polynomials.
Developed by Mats Lilja.

[cs,amp,cplus,cmin,cms] = FACTOR(c)

Factorization of polynomial c to the factorized one, cs. The C-polynomial will be divided into two parts. One with its zeros inside the unit disc, cplus, and one with zeros on and outside, cmin. The cmin-polynomial will then be factorized into the unit disc, cms. The noice variance will be amplified with the magnitude amp. The macro uses ROOT to find the roots in C. The roots are then sorted and factorized.
Reference : CCS Theorem 6.3 (p. 142-3)

[r,s,amp] = LQGA(a,b,c,rho)

Calculate a Gaussian Linear Quadratic Controller by an Input/Output approach. You have to enter the process polynomials, a,b and c, and the design parameter, rho. Output from the macro you get the controller polynomials, r and s. Amp is the amplified noise. This macro can't handle common factors in a and b (see LQGCF).
This macro uses following submacros: FACTOR, LQGSPEC and DIOPHANT.
Reference: CCS Lemma 12.2 & Teorem 12.4 (p.301-2)

[r,s,amp] = LQGCF(a,b,c,rho)

An algorithm for LQG-syntesis of a ARMAX-process with common factors in the process polynomials, a and b. This is a special case which LQGA can't handle.
This macro uses: FACTOR, COMFAC, LQGSPEC and DIOPHANT
Reference: CCS Theorem 12.5 (p. 306)

[p,ps,r] = LQGSPEC(a,b,rho)

Solves the spectral factorization in the LQG-problem,
r P P* = rau A A* + z^d B B* .
The macro uses ROOT to find the roots in PP*. The roots are then sorted and factorized in a proper way. Reference : CCS Lemma 12.1 (p. 299)

[r,s,amp] = MINVAR(a,b,c)

The MINVAR-macro calculate the minimum-variance controller, r and s. a,b and c is the process polynomials and amp is the amplified noise.

It solves the prediction problem by using the Diophantine equation. If the C-polynomial has zeros outside or on the unit disc it will spectral factorized. Also the B-polynomial will be factorized in 'good' and 'bad' parts , so that the controller will be stable.
It uses the FACTOR- and PRED-macro.
Reference : CCS Theorem 12.2 and 12.3 (p. 293-6)

[r,s,pvar] = OPTPRED(a,c,d,var)

To calculate the optimal d-step predictor OPTPRED solves the Diophantine equation. To get a stable predictor the C-polynomial must have its zeros inside the unit disc (Factorization). The arguments are the process polynomials, a and c, the prediction horizon, d, and the noise variance, var. The OPTPRED will answer with predictor polynomials, r and s, and the predictor variance, pvar.
OPTPRED uses the FACTOR- and the PRED-macros.
Reference : CCS Theorem 12.1 (p. 288 and 290-1 (unstable C))

[f,g] = PRED(a,b,c,d)

This macro solves the d-step predictor problem by solving the Diophantine equation, A F + B G = C q^(d-1).
This macro uses ML:s DIOPHANT-macro.
Reference : CCS Theorem 12.1 (p. 288)

[] = PZPLOT(a,b,c)

Plots poles and zeros for a discrete time system in a singular diagram with the unit circle. a,b and c are the process polynomials. Poles are represented by diagonal cross, zeros by octagones and noise zeros by squares.

3. HELP-facility for CTRL-C libraries

A big disadvantage with the use of libraries in CTRL-C has been the lack of help facilities. If you use a library from an other user you don't have any help facility at all (exept if you have printed documentation).

In MATHLAB-PC you have a very good help facility. If you type HELP <func.name>, you get the comments in the function head.

This was the idea for making this help facility for libraries. The result become much more general, but instead it takes a lot of function buffer.

The HELP-macro is an ordinary CTRL-C function (you find the code for the macro in Appendix II). The macro is called HLPLQG and has one argument, witch is a string of at least four letters. If the string has more then four letters it will be truncated.

A string of letters in CTRL-C will be translated to numbers in a vector. This vector can be compeared with a known vector (word) and if the two match a comment is written on the screen using the echo facility in Ctrl-C.

An example:

This macro below is an example of the HELP-facility idea.

```
//[] = HLP(string)
// An example of the HELP-facility for CTRL-C libraries

string = string(1,1:4);
word = 'NEWS';

if max(abs(string-word))=0 , echo=2 ;
//
//   NO NEWS IS GOOD NEWS !
//
echo=0;
```

And if we run it we get following:

```
[> hlp('news')
[> //
[> //   NO NEWS IS GOOD NEWS !
[> //
[> echo=0;
```

Observe that the dubble slash is important, because it indicates that this line is a comment.

You can now get some help for each macro in the HLPLQG-macro.

You can also get an overview of what the library contents by the argument 'SHOW'. This will type the list of macros like the one from page 1.

There is an other reserved word, 'NEWS'. The string 'NEWS' stands for a opportunety to communicate with other users and declaire news and changes in the library.

This is an example of using HLPLQG-macro :

```
[>  hlplqg('minvar')
//[r,s,amp]=MINVAR(a,b,c)
//
// This macro calculates the minimum variance controller
//                        G             s
//           u(k) = - ------- y(k) = - - - y(k)
//                    Bplus F           r
//
// a,b,c - Process polynomials
// r,s   - Controller polynomials
// amp   - amplified noise due to non-monic C and Spec.Fac.
```

```
//
// Reference: CCS Theorem 12.2 & 12.3 (p. 293-6)

// To do this it uses the SPEC-macro and the PRED-macro.
echo=0;
```

4. Exemples

In this last chapter some very simple exemples from the course in Computer Controlled Systems are solved by the macros in the LQGIO-library.

Exemple 12.3

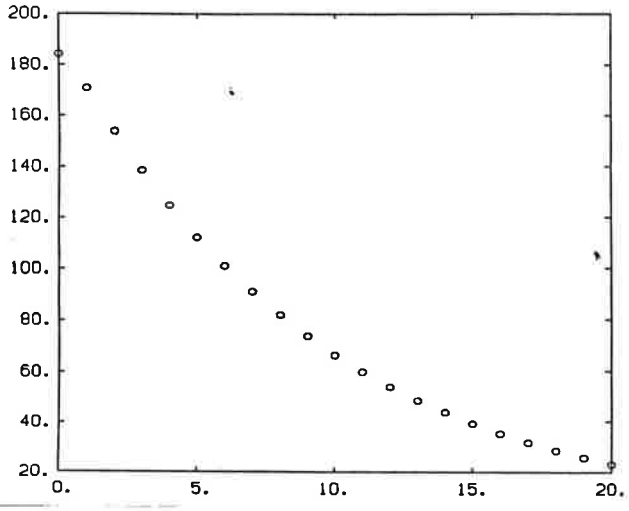
A stochastic process is described by

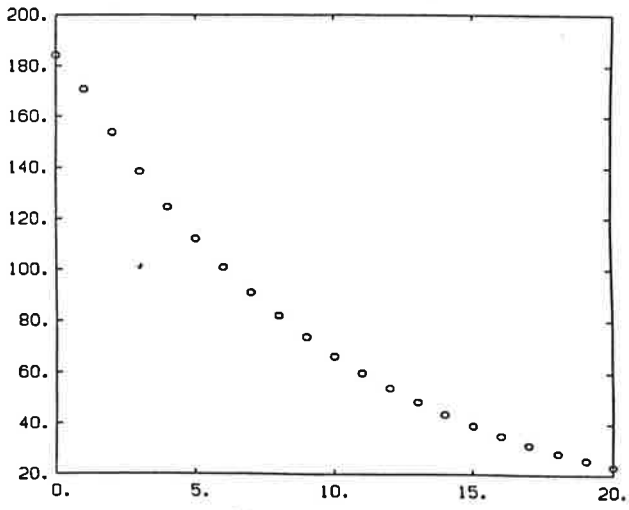
$$y(k) - 0.9y(k-1) = e(k) + 5e(k-1)$$

a) Determine an eqvivalent description such that the zero of a corresponding polynomial C is inside the unit circle.

```
[>  a=[1 -.9];

[>  c=[1 5];

[>  hlplqg('factor')
//[cs,amp,cplus,cmin,cms]=FACTOR(c)
//
// Factorization of polynom C(z).
//    c = cplus*cmin
//    cms = spec.fac(cmin)
//    cs  = cplus*cms    ;    amp = amplifid noice from spec.fac.
//
//Reference : CCS Theorem 6.3 (p. 142-3)
echo=0;

[>  [cs,amp]=factor(c)

AMP   =

    5.

CS    =

   1.0000    0.2000

[>  css=amp*cs

CSS   =

    5.    1.

[>  ry=covar(a,c,20)

[>  rycss=covar(a,css,20)
```

```
[>  cpl1(ry)

[>  cpl1(rycss)
```





We see that the two different ARMA-processes has the same covariance functions.

b) Determine the two-step-ahead predictor for the process and the variance of the predictor error.

```
[>  hlplqg('optpred')
//[r,s,pvar]=OPTPRED(a,c,d,var)
//
// Calculate the optimal d-step predictor.
//      ^           qG        s
//     y(k+d|k) = -- y(k) = - y(k)
//                 C*        r
// C* has its zeros inside the unit disk.
// PVAR is the prediction variance.
//
//Reference : CCS Theorem 12.1 (p. 288)
//
// This macro uses the PRED-macro and the SPEC-macro.
echo=0;

[>  d=2;

[>  [r,s,pvar]=optpred(a,c,d,1)

PVAR  =

  55.2500

S     =

    0.0900         0.0000
```

```
R     =

    1.0000      0.2000
```

Exemple 12.11

Given the process

$$y(k) - 1.5y(k-1) + 0.7y(k-2) = u(k-2) - 0.5u(k-3) + e(k) - 0.2e(k-1)$$

a) Assume no disturbance and compute the deadbeat controller for the system.

Mats Lilja has developed a very nice CTRL-C macro for polynomial synthesis based on the algorithm described in CCS, Algorithm 10.1 (p. 231). The macro is called DRST2 (OBS! it is not included in LQGIO-library).

```
[>  bminus = 1;

[>  am = [1 0 0 0];

[>  bm1 = [1 0];

[>  ao = [1 0];

[>  ar = 1;

[>  [r,s,t]=drst2(a,b,bminus,am,bm1,ao,ar)

T     =

    1.    0.    0.

S     =

   1.5500   -1.0500    0.0000

R     =

   1.0000    1.0000   -0.7500
```

b) Assume now that we have disturbances. Compute the minimum-variance control law.

```
[>  [rmv,smv]=minvar(a,b,c)

SMV   =

   1.2500   -0.9100    0.0000

RMV   =

   1.0000    0.8000   -0.6500
```

c) What is the steady-state variance of y (and u) when the deadbeat and the minimum-variance controllers are used. Assume Var(e)=1.

```
[>  [yvps,uvps]=regvar(a,b,c,1,r,s)

UVPS  =

   2.7538

YVPS  =

   2.7800

[>  [yvmv,uvmv]=regvar(a,b,c,1,rmv,smv)

UVMV  =

   1.6708

YVMV  =

   2.6900
```

The difference in variances of y is not very large but the control signal variance is reduced by 40 procent.

d) Use the LQG-macros to calculate LQG-controllers with some different weightings on the control signal.

```
[>  [rl1,sl1]=lqga(a,b,c,0)

SL1   =

   1.2500   -0.9100    0.0000

RL1   =

   1.0000    0.8000   -0.6500
```

When rho is set to zero we get the minimum-variance control law obtained in b).

```
[>  [rl2,sl2]=lqga(a,b,c,0.5)

SL2   =

   0.8596   -0.6575    0.0000    0.0000

RL2   =

   1.0000    0.4928   -0.4696    0.0000

[>  [yvl2,uvl2]=clvar(a,b,c,1,rl2,sl2)

UVL2  =
```

```
   0.7770

YVL2  =

   2.8487
```

This controller decreases the control signal variance but increases the output variance compeared with the deadbeat and the minimum-variance controllers.

LQGCF is a macro which solves the LQG-problem with common factors in A and B. This algorithm is taken from CCS theorem 12.5. If we use it on a problem with no common factors, we don't get any direct term in the controller. In such cases one should NOT use this algorithm.

```
[>  [rr,ss]=lqgcf(a,b,c,0.3)

SS    =

   0.7204   -0.6757    0.0000

RR    =

   1.0000    0.5762    0.4456   -0.4826
```

If we have a problem with for instance drifting disturbance, then we get a unstable A-polynomial and a common factor in A and B. In this exemple we have the same process as before but A and B has a common factor and its zero is on the unit circle at 1.

```
[>  aa=[1 -2.5 2.2 -0.7 0];

[>  bb=[1 -1.5 0.5];

[>  root(aa)

ANS   =

   0.7500 + 0.3708i
   1.0000 - 0.0000i
   0.7500 - 0.3708i
   0.0000 + 0.0000i

[>  [rcf,scf]=lqgcf(aa,bb,c,0.3)

SCF   =

   3.0784   -3.8973    1.4792    0.0000

RCF   =

   1.0000    0.5762   -2.6328    1.0566

[>  root(rcf)

ANS   =
```

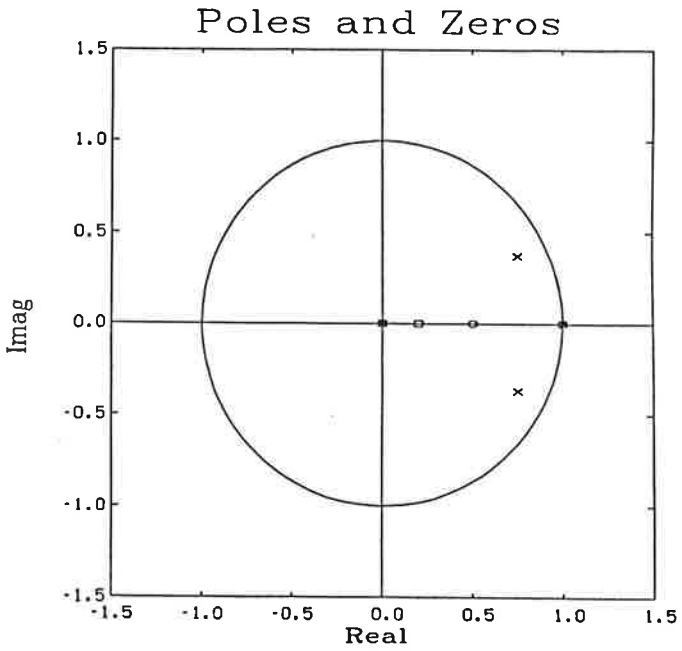
-2.0834 - 0.0000i
1.0000 - 0.0000i
0.5072 + 0.0000i

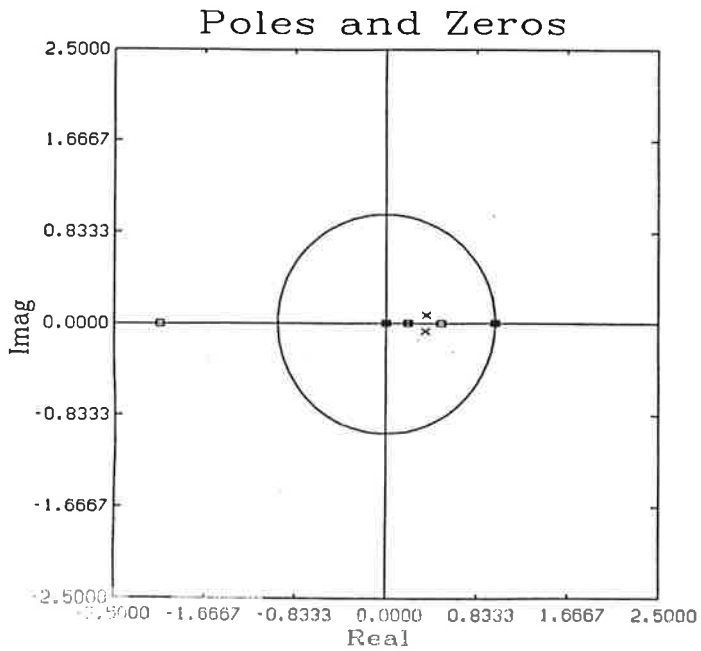
This controller has a direct term and the R-polynomial contains the common factor which has its zeros outside or on the unit circle.

[> pzplot(aa,bb,c)

[> t=c;

[> clpz(aa,bb,c,rcf,scf,t)





Appendix I: LQGIO macros

```
//[]=CLPZ(a,b,c,r,s,t)
//
// Plots the poles and zeros for a
// discrete time closed loop system
//
// a,b,c - process polynomials
// r,s,t - controller polynomials
//
// pole       - diagonal cross
// zero       - octagon
// noise zero - square
//
// This macro uses the PZPLOT-macro
//
//       Author : Bernt Nilsson

ar  = conv(a,r);
bs  = conv(b,s);
nar = size(ar);
nbs = size(bs);
d   = nar(2)-nbs(2);
for i=1:d , bs=[0 bs];
arbs= ar + bs;
cr  = conv(c,r);
tb  = conv(t,b);
kmax= nar(2)+1;
pzplot(arbs,tb,cr)
```

```
//[yvar,uvar]=CLVAR(a,b,c,var,r,s)
//
// Computes the variance for a closed loop system
//            CR                  -CS
//     y = ------- e  ;  u = ------- e
//         AR + BS             AR + BS
//
// yvar = output variance
// uvar = control signal variance

// This macro uses ML:s COVARiance macro
// and COMFAC.
//
//       Author : Bernt Nilsson

ar  = conv(a,r);
bs  = conv(b,s);
nar = size(ar);
nbs = size(bs);
d   = nar(2)-nbs(2);
for i=1:d , bs=[0 bs];
arbs= ar + bs;
cr  = conv(c,r);
cs  =-conv(c,s);
kmax= nar(2)+1;
[arb1,cr1]=COMFAC(arbs,cr);
ry  = COVAR(arb1,cr1,kmax);
```

```
yvar= ry(1,2)*var;
[arb2,cs2]=COMFAC(arbs,cs);
ru  = COVAR(arb2s,cs2,kmax);
uvar= ru(1,2)*var;
```

```
//[a1,b1,a2m,a2p]=COMFAC(a,b)
//
// Finds the greatest common divisor, A2, of A and B.
//
// A2m is the factor of A2 which has its zeros on or
// outside the unit disc and A2p has its inside.
//
//      Author : Bernt Nilsson

ra    = root(a);
rb    = root(b);
nra   = size(ra);
nrb   = size(rb);
epi   = 1e-14;
k     = 1;
a2(1) = 1;
for j=1:nrb(1) , ...
  for i=1:nra(1) , ...
    if epi>abs(rb(j)-ra(i)) , ra2(k)=ra(i) ; k=k+1;
for i=1:k-1 , acf=conv(a2,[1 -ra2(i)]);...
  [a1,remain]=deconv(a,acf);...
  if remain<epi , a2=acf;
a1  = deconv(a,a2);
b1  = deconv(b,a2);
na2= size(a2);
if na2(2)>1 , ra2 = root(a2);
a2p = 1;
for i=1:na2(2)-1 , if abs(ra2(i))<1-epi , a2p=conv(a2p,[1 -ra2(i)]);
a2m = deconv(a2,a2p);
```

```
//[ry,rye]=COVAR(a,c,kmax);
//
// Uses Yule-Walker for computing the covariance
// and the cross covariance functions for an
// ARMA-process.
//
// ry  = covariance function (col1 = time , col2 = covar.)
// rye = cross covar. funct. (col1 = time , col2 = cross covar.)
//
// Reference: Olbjer , TIDSERIEANALYS : Appendix
//
// Author : Mats Lilja

 a = [a 0];
 c = [c 0];
 dima = size(a);
 na = dima(2);
 dimc = size(c);
 nc = dimc(2);
 dac = na - nc;
```

```
if dac>0, c = [c 0*(1:dac)]; nc = na;
da = na - 1;
dc = nc - 1;
nn = nc;
ee = eye(nn);
for j=1:nn, eee(j,:) = ee(nn-j+1,:);
for j=1:na, aa(1,j) = a(na-j+1);
for j=1:nn, ma(j,:) = conv(aa,ee(j,:));
for j=1:nn, mc(j,:) = conv(c,eee(j,:));
ma1 = ma(:,na:nn+da);
mc1 = mc(:,nc:nn+dc);
ma2 = ma1;
for j=1:da, ma2(:,j+1) = ma2(:,j+1) + ma(:,na-j);
rye = ma1\c';
ry = ma2\(mc1*rye);
a1 = a/a(1);
a1 = a1(2:na);
k1 = kmax + 1;
for j=(nn+1):k1, ry(j) = -a1*ry(j-1:-1:j-da);
for j=(nn+1):k1, rye(j) = -a1*rye(j-1:-1:j-da);
col1 = (0:kmax)';
col2 = ry(1:k1);
col3 = rye(1:k1);
ry = [col1 col2];
rye = [col1 col3];
```

```
//[] = cpl1(r);
//
// A macro for plotting the COVAR-macro.
//
// Author : Mats Lilja
  plot('scale');
  erase;
  plot(r(:,1),r(:,2),'point=9');
```

```
//[x,y]=Diophant(a,b,c);
//
// Solves the Diophantine equation
//
//     A X + B Y = C
//
// where A,B,C,X and Y are polynomials.
// A unique solution is guaranteed by choosing deg(Y) = deg(A)-1
// and deg(X) = deg(C) - deg(A).
//
// Note that na,nb,nc etc. below are coefficient vector dimensions
// and NOT polynomial degrees.
// Author: Mats Lilja

 dima=size(a);
 na=dima(2);
 dimb=size(b);
 nb=dimb(2);
 dimc=size(c);
 nc=dimc(2);
```

```
nx=nc-na+1;
ny=na-1;
ex=eye(nx);
ey=eye(ny);
for j=1:nx, mx(j,:)=conv(a,ex(j,:));
for j=1:ny, my(j,:)=conv(b,ey(j,:));
m = [mx ; [0*eye(ny,nx-nb+1) my]];
xy = c/m;
x = xy(1:nx);
y = xy(nx+1:nc);
```

```
//[cs,amp,cplus,cmin,cms]=FACTOR(c)
//
// Factorization of polynom C(z).
//   c = cplus*cmin
//   cms = spec.fac(cmin)
//   cs  = cplus*cms   ;   amp = amplifid noice from spec.fac.
//
//Reference : CCS Theorem 6.3 (p. 142-3)
//
//         Author : Bernt Nilsson

//Factorization of C = C+*C-
dimc = size(c);
nc   = dimc(2);
cmin = 1;
r    = roots(c);
for i=1:nc-1 , if abs(r(i))>=1 , cmin=conv(cmin,[1 -r(i)]);

//Factorization of C- into the unit circle
ncmin= size(cmin);
amp  = cmin(ncmin(2));
cplus= deconv(c,cmin);
for j=0:ncmin(2)-1 , csp(j+1)=cmin(ncmin(2)-j)/amp;
cms  = csp';
cs   = conv(cms,cplus);
```

```
//[r,s,amp]=LQGA(a,b,c,rau)
//
// Calculate a Gaussian Linear Quadratic Controller
//        by an input/output approach.
//
// a,b,c - the process polynomials
// rau   - design parameter
// r,s   - controller polynomials
// amp   - amplifid noice due to non-monic C and Spec.Fac.
//
// Reference: CCS Lemma 12.2  & Teorem 12.4 (p.301-2)

// This macro uses following Ctrl-C macros:
// FACTOR, LQGSPEC and DIOPHANT.
//
//          Author : Bernt Nilsson

//Factorization of C
```

```
amp = c(1,1);
cf  = c/amp;
if max(abs(roots(c)))>=1 , [cf,amps]=FACTOR(c); amp=amp*amps;

//Spectral factorization of P
[p,ps,rr] = LQGSPEC(a,b,rau);

//Reciprocal polynomials
epi  = 1e-14;
na   = size(a);
for i=1:na(2) , as(1,i)=a(na(2)-i+1);
while abs(as(1,1))<epi , as=as(1,2:na(2)) ; na(2)=na(2)-1;
nc   = size(cf);
for i=1:nc(2) , cs(1,i)=cf(nc(2)-i+1);

//Solution of the Diophantine equation
rp    = rr*p;
bcs   = conv(b,cs);
[x,ss]= DIOPHANT(as,rp,bcs);

//Polynomial division
vl = conv(ps,x);
if rau<>0 , vl = vl + conv(rau*a,ss);
rs = deconv(vl,b);

//Controller polyomials
nrs  = size(rs);
nss  = size(ss);
d    = nrs(2)-nss(2);
for i=1:nrs(2) , r(1,i)=rs(nrs(2)-i+1);
for i=1:nss(2) , s(1,i)=ss(nss(2)-i+1);
z(1,1)=1;
for i=2:abs(d)+1 , z(1,i)=0;
if d>0 , s=conv(z,s);
if d<0 , r=conv(z,r);
```

```
//[r,s,amp]=LQGCF(a,b,c,rau)
//
// An LQG-algorithm for syntesis of a
// ARMAX-process with common factors
// between A, B.
//
// Reference: CCS Theorem 12.5 (p. 306)

// This macro uses: FACTOR, COMFAC, LQGSPEC and DIOPHANT
//
//        Author : Bernt Nilsson

//Factorization of C
amp  = c(1,1);
cf   = c/amp;
if max(abs(root(c)))>=1 , [cf,amps]=FACTOR(c); amp=amp*amps;

//Common factors, A and B
[a1,b1,a2m,a2p]=COMFAC(a,b);

//Spectral factorization of P
```

```
[p1,ps1,rr1]=LQGSPEC(a1,b1,rau);

//Solution of the Diophantine equation
vl   = conv(p1,cf);
aa2  = conv(a1,a2m);
[r1,s1]=DIOPHANT(aa2,b1,vl);
r    = conv(real(a2m),real(r1));
na2m = size(a2m);
m    = na2m(2)-1;
z    = 1;
for i=2:m , z(1,i)=[z 0];
s    = conv(z,real(s1));
```

```
//[p,ps,r]=LQGSPEC(a,b,rau);
//
// Solves the spectral factorization in the
// LQG-problem.
//         r P P* = rau A A* + z^d B B*
//
//Reference : CCS Lemma 12.1 (p. 299)

//          Author : Bernt Nilsson

//Reciprocal polynomials
epi  = 1e-14;
na   = size(a);
nb   = size(b);
d    = na(2)-nb(2);
for i=1:na(2) , asp(i)=a(na(2)-i+1);
as   = asp';
while abs(as(1,1))<epi , as=as(1,2:na(2)) ; na(2)=na(2)-1;
for i=1:nb(2) , bs(1,i)=b(nb(2)-i+1);
while abs(bs(1,1))<epi , bs=bs(1,2:nb(2)) ; nb(2)=nb(2)-1;

//Polynomial multiplication z^d B*
z(1) = 1;
for i=2:d+1 , z(i)=0;
bs   = conv(z',bs);

//Spectral factorization
// ( when rau=0 we have a special case
//   witch must be handled with some care )
hla  = rau*conv(a,as);
hlb  = conv(b,bs);
hl   = hlb;
k    = size(hla)-size(hlb);
if rau<>0 , for i=1:k(2) , hlb = [0 hlb];
if rau<>0 , hl = hla + hlb;
p    = 1;
rpps = roots(hl);
np   = size(rpps);
[i,y]= sort(abs(rpps));
p    = [1 -rpps(i(1))];
for j=2:nb(2)+d-1 , p = conv(p,[1 -rpps(i(j))]);
rps  = deconv(hl,p);
nrps = size(rps);
r    = rps(nrps(2));
```

```
ps   = rps/r;
p    = real(p);
ps   = real(ps);
r    = real(r);
```

```
//[r,s,amp]=MINVAR(a,b,c)
//
// This macro calculate the minimum variance controller
//                          G              s
//            u(k) = - ------- y(k) = - - y(k)
//                     Bplus F           r
//
// a,b,c - Process polynomials
// r,s   - Controller polynomials
// amp   - amplifid noice due to non-monic C and Spec.Fac.
//
// Reference: CCS Theorem 12.2 & 12.3 (p. 293-6)
//
// To do this it uses the FACTOR-macro and the PRED-macro.
//
//           Author : Bernt Nilsson

//Factorization of C
amp  = c(1,1);
cs   = c/amp;
if max(abs(root(c)))>=1 , [cs,amps]=FACTOR(c); amp=amp*amps;

//Factorization of B=B+*B- and Spec. Fac. of B-
bplus= b;
bmin = 1;
bms  = 1;
ampb = 1;
epi  = 1e-14;
if max(abs(root(b)))>=1-epi , [bs,ampb,bplus,bmin,bms]=FACTOR(b);

//Prediction problem
cc   = conv(cs,bms);
na   = size(a);
nb   = size(b);
d    = na(2)-nb(2);
f    = 1;
[f,g]= PRED(a,bmin,cc,d);

//Minimum-Variance Controller
ss   = g;
rr   = conv(bplus,f);
r    = rr/rr(1);
s    = ss/rr(1);
```

```
//[r,s,pvar]=OPTPRED(a,c,d,var)
//
// Calculate the optimal d-step predictor.
//      ^           qG         s
//      y(k+d|k) = -- y(k) = - y(k)
//                 C*         r
```

```
// C* has its zeros inside the unit disk.
// PVAR is the prediction variance.
//
//Reference : CCS Theorem 12.1 (p. 288)
//
// This macro uses the PRED-macro and the FACTOR-macro.
//
//          Author : Bernt Nilsson

cs    = c;
amp   = 1;
if max(abs(root(c)))>=1 , [cs,amp]=FACTOR(c);
[f,g] = PRED(a,1,cs,d);
r     = cs;
s     = [g 0];
sumf  = 0;
nf    = size(f);
for i=1:nf(2) , sumf=sumf+f(i)**2;
pvar  = sumf*amp**2*var;
```

```
//[f,g]=PRED(a,b,c,d)
//
// This macro solves the d-step predictor problem
// by solving the Diophantine equation.
//
//            A F + B G = C q^(d-1)
//
// This macro uses ML:s DIOPHANT-macro.
//
//          Author : Bernt Nilsson

q(1)  = 1;
for i=2:d , q(i)=0 ;
cc    = conv(c,q');
[f,g] = DIOPHANT(a,b,cc);
```

```
// []=PZPLOT(a,b,c)
//
// Plots the poles and zeros for a
// discrete time system.
//
// a,b,c - process polynomials
//
// pole       - diagonal cross
// zero       - octagon
// noise zero - square
//
//    Author : Bernt Nilsson

na=size(a);
nb=size(b);
nc=size(c);
maxa=0;
maxb=0;
maxc=0;
```

```
if na(1,2)>1 , poles=root(a) ; maxa=max(abs([real(poles);imag(poles)]));
if nb(1,2)>1 , zeros=root(b) ; maxb=max(abs([real(zeros);imag(zeros)]));
if nc(1,2)>1 , nzeros=root(c); maxc=max(abs([real(nzeros);imag(nzeros)]));
scale=roun(2*max([maxa;maxb;maxc;1]) + 0.5)/2;
numb = 6;
dx = 2*scale/numb;
psize=[-scale -scale;scale scale;dx dx];
plot(psize,'SCALE')
erase
window([149/190 1;0 0]);
xaxx=[-scale;scale];
xaxy=[0;0];
yaxx=[-0.0001;0.0001];
yaxy=[scale;-scale];
t=0:0.05:2.1*pi;
x=cos(t);
y=sin(t);
plot(xaxx,xaxy,yaxx,yaxy,'solid',x,y,'solid')
if na(1,2)>1 , plot(real(poles),imag(poles),'point=1')
if nb(1,2)>1 , plot(real(zeros),imag(zeros),'point=9')
if nc(1,2)>1 , plot(real(nzero),imag(nzero),'point=3')
title('Poles and Zeros',' llll lll  llll')
xlabel('Real',' lll')
ylabel('Imag',' lll')
```

Appendix II: LQGIO Help macro

```
//[]=hlplqg(string)

//This is a HELP-macro for the LQG-library

string = string(1,1:4);

word='show';
if max(abs(string-word))=0 , echo=2;
//                          LQGI/O Library
//                          --------------
//
// This is a library for Linear Quadratic Gaussian Control Syntesis (LQG)
// based on Input/Output approach. (Reference: CCS chapter 12).
//
// CLVAR          - Calculate the Output- and Cont. Sign. Variance.
// COVAR          - Calculate the Covariance Function of a ARMA-proc.
//   CPL1         - Plots the COVAR.
//
// MINVAR         - Calculate a Minimum-Variance Controller
// OPTPRED        - Calculate a Optimal d-step Predictor.
//   DIOPHANT     - Solves the Diophantine equation
//   PRED         - Solves the predictor problem by solving the Diophantine eq.
//   FACTOR       - Factorization of a polynomial.
//
// LQGA           - Calculates a LQG-controller (I/O-appr.).
// LQGCF          - Same as LQG but handle Common Factors.
//   COMFAC       - Finds the Greatest Common Divisor.
//   LQGSPEC      - Calculate the P-polynomial in the LQG-problem.
//
// PZPLOT         - Plots poles and zeros for a discrete time system.
// CLPZ           - Plots poles and zeros for a closed loop system.
//
echo=0;

word='news';
if max(abs(string-word))=0 , echo=2;
//
//  NO NEWS IS GOOD NEWS !
//
echo=0;

word='clva';
if max(abs(string-word))=0 , echo=2;
//[yvar,uvar]=CLVAR(a,b,c,var,r,s)
//
// Computs the variance for a closed loop system
//          CR                -CS
//    y = ------- e  ;  u = ------- e
//        AR + BS             AR + BS
//
// yvar = output variance
// uvar = control signal variance
//
// This macro uses ML:s COVARiance macro
```

```
echo=0;

word='cova';
if max(abs(string-word))=0 , echo=2;
//[ry,rye]=COVAR(a,c,kmax);
//
// Uses Yule-Walker for computing the covariance
// and the cross covariance functions for an
// ARMA-process.
//
// ry  = covariance function (col1 = time , col2 = covar.)
// rye = cross covar. funct. (col1 = time , col2 = cross covar.)
//
// Reference: Olbjer , TIDSERIEANALYS : Appendix
echo=0;

word='cpl1';
if max(abs(string-word))=0 , echo=2;
//[] = cpl1(r);
//
// A macro for plotting the COVAR-macro.
echo = 0;

word='minv';
if max(abs(string-word))=0 , echo=2;
//[r,s,amp]=MINVAR(a,b,c)
//
// This macro calculate the minimum variance controller
//                        G            s
//          u(k) = - ------- y(k) = - - y(k)
//                   Bplus F           r
//
// a,b,c - Process polynomials
// r,s   - Controller polynomials
// amp   - amplifid noice due to non-monic C and Spec.Fac.
//
// Reference: CCS Theorem 12.2 & 12.3 (p. 293-6)

// To do this it uses the FACTOR-macro and the PRED-macro.
echo=0;

word='optp';
if max(abs(string-word))=0 , echo=2;
//[r,s,pvar]=OPTPRED(a,c,d,var)
//
// Calculate the optimal d-step predictor.
//     ^          qG        s
//    y(k+d|k) = -- y(k) = - y(k)
//               C*        r
// C* has its zeros inside the unit disk.
// PVAR is the prediction variance.
//
//Reference : CCS Theorem 12.1 (p. 288)

// This macro uses the PRED-macro and the FACTOR-macro.
echo=0;

word='diop';
```

```
if max(abs(string-word))=0 , echo=2;
//[x,y]=Diophant(a,b,c);
//
// Solves the Diophantine equation
//
//     A X + B Y = C
//
// where A,B,C,X and Y are polynomials.
// A unique solution is guaranteed by choosing deg(Y) = deg(A)-1
// and deg(X) = deg(C) - deg(A).
echo=0;

word='pred';
if max(abs(string-word))=0 , echo=2;
//[f,g]=PRED(a,b,c,d)
//
// This macro solves the d-step predictor problem
// by solving the Diophantine equation.
//
//            A F + B G = C q^(d-1)
//
// This macro uses ML:s DIOPHANT-macro.
echo=0;

word='fact';
if max(abs(string-word))=0 , echo=2;
//[cs,amp,cplus,cmin,cms]=FACTOR(c)
//
// Factorization of polynom C(z).
//   c = cplus*cmin
//   cms = spec.fac(cmin)
//   cs  = cplus*cms    ;   amp = amplifid noice from spec.fac.
//
//Reference : CCS Theorem 6.3 (p. 142-3)
echo=0;

word='lqga';
if max(abs(string-word))=0 , echo=2;
//[r,s,amp]=LQGA(a,b,c,rau)
//
// Calculate a Gaussian Linear Quadratic Controller
//        by an input/output approach.
//
// a,b,c - the process polynomials
// rau   - design parameter
// r,s   - controller polynomials
// amp   - amplifid noice due to non-monic C and Spec.Fac.
//
// Reference: CCS Lemma 12.2  & Teorem 12.4 (p.301-2)

// This macro uses following Ctrl-C macros:
// FACTOR, LQGSPEC and DIOPHANT.
echo=0;

word='lqgc';
if max(abs(string-word))=0 , echo=2;
//[r,s,amp]=LQGCF(a,b,c,rau)
//
```

```
// An LQG-algorithm for syntesis of a
// ARMAX-process with common factors
// between A, B.
//
// Reference: CCS Theorem 12.5 (p. 306)

// This macro uses: FACTOR, COMFAC, LQGSPEC and DIOPHANT
echo=0;

word='lqgs';
if max(abs(string-word))=0 , echo=2;
//[p,ps,r]=LQGSPEC(a,b,rau)
//
// Solves the spectral factorization in the
// LQG-problem.
//         r P P* = rau A A* + z^d B B*
//
//Reference : CCS Lemma 12.1 (p. 299)
echo=0;

word='comf';
if max(abs(string-word))=0 , echo=2;
//[a1,b1,a2m,a2p]=COMFAC(a,b)
//
// Finds the greatest common divisor, A2, of A and B.
//
// A2m is the factor of A2 witch has its zeros on or
// outside the unit disc and A2p has its inside.
echo=0;

word='pzpl';
if max(abs(string-word))=0 , echo=2;
//[] = PZPLOOT(a,b,c)
//
// Plots poles and zeros for a
// discrete time system
//
// a,b,c - process polynomials
//
// pole       - diagonal cross
// zero       - octagon
// noise zero - square
echo=0;

word='clpz';
if max(abs(string-word))=0 , echo=2;
//[] = CLPZ(a,b,c,r,s,t)
//
// Plots poles and zeros for a
// discrete time closed loop system
//
// a,b,c - process polynomials
// r,s,t - controller polynomials
//
// pole       - diagonal cross
// zero       - octagon
// noise zero - square
echo=0;
```