LUND UNIVERSITY

Symbolic Evaluation of Certain Complex Integrals

Nilsson, Johan

1994

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

Link to publication

Citation for published version (APA):
Nilsson, J. (1994). *Symbolic Evaluation of Certain Complex Integrals*. (Technical Reports TFRT-7523). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:
1

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

ISSN 0280–5316
ISRN LUTFD2/TFRT--7523--SE

Symbolic Evaluation of Certain Complex Integrals

Johan Nilsson

Department of Automatic Control
Lund Institute of Technology
August 1994

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

Document name
INTERNAL REPORT

Date of issue
August 1994

Document Number
ISRN LUTFD2/TFRT--7523--SE

Author(s)
Johan Nilsson

Supervisor
Karl Johan Åström

Sponsoring organisation

Title and subtitle
Symbolic Evaluation of Certain Complex Integrals

Abstract

Algorithms to evaluate quadratic loss functions in discrete and continuous time are derived. The algorithms are implemented in a toolbox for Maple.

Key words
Loss function formulas, Variance evaluation, Computer Algebra, Maple.

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
English

Number of pages
14

Recipient's notes

Security classification

The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Fax +46 46 110019, Telex: 33248 lubbis lund.

1. Introduction

Analysis of linear deterministic and stochastic systems with quadratic criteria leads to a certain class of complex integrals. An early example is given in [4] where analytic expressions for one type of integral is given for systems of low order. Similar tables are found in [3]. In [1] it was shown that the integrals can be computed recursively using algorithms that are closely related to stability tests using the Routh-Hurwitz (continuous time) or the Schur-Cohn (discrete time) method. In [2] it is also shown how the integrals are related to the reflection coefficients associated with a linear transfer function.

In this paper we will present a toolbox in Maple for evaluating the integrals.

2. Problem Formulation

Consider a linear system with the impulse response $h(t)$. The mean square of the impulse response of the system is

$$I = \int_0^\infty h^2(t)dt$$

If the transfer function of the system is

$$H(s) = \frac{B(s)}{A(s)}$$

it follows from Parsevals theorem that

$$I = \frac{1}{2\pi i}\int_{-i\infty}^{i\infty} \frac{B(s)B(-s)}{A(s)A(-s)}ds \tag{1}$$

Similarly we find that the variance of the output of the system if the input is white noise is also given by the integral (1), see Figure 1 and [1].

A similar problem for discrete time systems leads to the integral

$$I = \frac{1}{2\pi i}\oint_{|z|=1} \frac{B(z)B(z^{-1})}{A(z)A(z^{-1})}\frac{dz}{z} \tag{2}$$

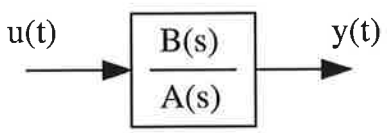

Figure 1. Filter driven by the white noise source $u(t)$.

Algorithms to evaluate this type of integrals are given in [1], [3] and [4]. When evaluating the covariance between the outputs of two filters driven by the same white noise, see Figure 2, integrals on the form

$$I = \frac{1}{2\pi i}\oint_{|z|=1} \frac{B(z)C(z^{-1})}{A(z)A(z^{-1})}\frac{dz}{z} \tag{3}$$

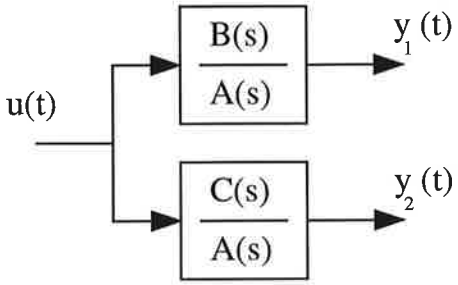

Figure 2. Two filters driven by the white noise source $u(t)$.

and

$$I = \frac{1}{2\pi i}\int_{-i\infty}^{i\infty} \frac{B(s)C(-s)}{A(s)A(-s)}ds \tag{4}$$

appears.

To evaluate these integrals the algorithms in [1] have to be generalized.

3. Derivation of Algorithms

The generalized algorithms are derived in the same way as Theorem 5.2.3 and Theorem 5.3.3 in [1]. Introduce

$$I_k = \frac{1}{2\pi i}\int_{-i\infty}^{i\infty} \frac{B_k(s)C_k(-s)}{A_k(s)A_k(-s)}ds$$

where the polynomials $A_k(s)$ and $B_k(s)$ are defined by (5.3.9) and (5.3.10) in [1], and

$$C_k(s) = c_1^k s^{k-1} + c_2^k s^{k-2} + ... + c_k^k$$

which is defined recursively from

$$C_n(s) = C(s)$$
$$C_{k-1}(s) = C_k(s) - \gamma_k \tilde{A}_k(s)$$

We observe that $I_n = I$. An algorithm to calculate I follows from the following theorem.

THEOREM 1
Assume that the polynomial $A(s)$ has all its roots in the left half plane. Then

$$I_k = I_{k-1} + \frac{1}{2}(\gamma_k\tilde{\beta}_k + \beta_k\tilde{\gamma}_k - \frac{\beta_k\gamma_k}{\alpha_k}) \qquad k = 1, 2, ..., n$$
$$I_0 = 0$$

where

$$\alpha_k = \frac{a_0^k}{a_1^k}$$
$$\beta_k = \frac{b_1^k}{a_1^k}$$

$$\tilde{\beta}_k = \frac{b_1^k}{a_0^k}$$
$$\gamma_k = \frac{c_1^k}{a_1^k}$$
$$\tilde{\gamma}_k = \frac{c_1^k}{a_0^k}$$

Proof:

The proof is done as Theorem 5.3.3 in [1] with the change that

$$\begin{aligned} I_{k-1} &= \frac{1}{2\pi i}\int_{-i\infty-\epsilon}^{i\infty-\epsilon} \frac{B_{k-1}(s)C_{k-1}(-s)}{A_k(s)A_k(-s)}ds = \\ &= \frac{1}{2\pi i}\{\int \frac{B_k(s)C_k(-s)}{A_k(s)A_k(-s)}ds - \gamma_k \int \frac{B_k(s)\tilde{A}_k(-s)}{A_k(s)A_k(-s)}ds \\ &- \beta_k \int \frac{\tilde{A}_k(s)C_k(-s)}{A_k(s)A_k(-s)}ds + \beta_k\gamma_k \int \frac{\tilde{A}_k(s)\tilde{A}_k(-s)}{A_k(s)A_k(-s)}\} \\ &= I_k - \frac{1}{2}\gamma_k\tilde{\beta}_k - \frac{1}{2}\beta_k\tilde{\gamma}_k + \frac{1}{2}\frac{\beta_k\gamma_k}{\alpha_k} \end{aligned}$$

From this it follows that

$$I_k = I_{k-1} + \frac{1}{2}(\gamma_k\tilde{\beta}_k + \beta_k\tilde{\gamma}_k - \frac{\beta_k\gamma_k}{\alpha_k})$$

which completes the proof. □

In the same way the discrete variant of the theorem, Theorem 5.2.3 in [1], can be generalized. Introduce

$$I_k = \frac{1}{2\pi i}\oint_{|z|=1} \frac{B_k(z)C_k(z^{-1})}{A_k(z)A_k(z^{-1})}\frac{dz}{z} \tag{5}$$

where $A_k(z)$ and $B_k(z)$ are defined by (5.2.5) and (5.2.6) in [1], and

$$C_k(z) = c_o^k z^k + c_1^k z^{k-1} + ... + c_k^k$$

with the recursion

$$\begin{aligned} C_n(z) &= C(z) \\ C_{k-1}(z) &= z^{-1}\{C_k(z) - \gamma_k A_k^*(z)\} \end{aligned}$$

As in the continuous case we observe that $I = I_n$. An algorithm to calculate I_n follows from the following theorem.

THEOREM 2

Let the polynomial $A(z)$ have all its zeros inside the unit circle. The integrals I_k then satisfy the following recursive equation

$$\begin{aligned} I_k &= (1-\alpha_k^2)I_{k-1} + \beta_k\gamma_k \qquad k = 1, 2, ..., n \\ I_0 &= \beta_0\gamma_0 \end{aligned}$$

where

$$\alpha_k = \frac{a_k^k}{a_0^k}$$
$$\beta_k = \frac{b_k^k}{a_0^k}$$
$$\gamma_k = \frac{c_k^k}{a_0^k}$$

Proof:

The proof is done as in Theorem 5.2.3 of [1] with the change that

$$I_{k-1} = \frac{1}{1-\alpha_k^2}\frac{1}{2\pi i}\oint_{|z|=1}\frac{B_{k-1}(z)C_{k-1}(z^{-1})}{A_k(z)A_k(z^{-1})}\frac{dz}{z}$$

and thus

$$\begin{aligned}(1-\alpha_k^2)\,I_{k-1} &= \frac{1}{2\pi i}\oint_{|z|=1}\frac{[B_k(z)-\beta_k A_k^*(z)][C_k(z^{-1})-\gamma_k A_k^*(z^{-1})]}{A_k(z)A_k(z^{-1})}\frac{dz}{z} = \\ &= \frac{1}{2\pi i}\oint_{|z|=1}\frac{B_k(z)C_k(z^{-1})}{A_k(z)A_k(z^{-1})}\frac{dz}{z} - \frac{\gamma_k}{2\pi i}\oint_{|z|=1}\frac{B_k(z)A_k^*(z^{-1})}{A_k(z)A_k(z^{-1})}\frac{dz}{z} \\ &- \frac{\beta_k}{2\pi i}\oint_{|z|=1}\frac{A_k^*(z)C_k(z^{-1})}{A_k(z)A_k(z^{-1})}\frac{dz}{z} + \frac{\beta_k\gamma_k}{2\pi i}\oint_{|z|=1}\frac{A_k^*(z)A_k^*(z^{-1})}{A_k(z)A_k(z^{-1})}\frac{dz}{z} = \\ &= I_k - \gamma_k\beta_k - \beta_k\gamma_k + \beta_k\gamma_k = I_k - \beta_k\gamma_k\end{aligned}$$

When $k = 0$ we get from (5) that

$$I_0 = \frac{1}{2\pi i}\oint_{|z|=1}\frac{b_0^0}{a_0^0}\frac{c_0^0}{a_0^0}\frac{dz}{z} = \beta_0\gamma_0$$

And the proof is completed. □

4. Examples

Functions to evaluate the integrals (1)-(4) have been implemented in Maple. The implementation of these functions are given in Appendix A. The use of these functions is illustrated with some examples.

EXAMPLE 1

Consider the setup in Figure 1, where

$$B(s) = 2s + 1$$
$$A(s) = s^2 + 4s + 3$$

and u(t) is white noise with unit variance. The variance of the output, $y(t)$, can then be calculated with the following Maple sequence.

```
> read '/home/johan/maple/var';
> B:=2*s+1:
> A:=s^2+4*s+3:
> filtvarc(A,B);
```

The answer is 13/24. For the theorem to be applicable stability of $A(s)$ is needed. This is easily checked with the following Maple command

```
> stabtestc(A);
```

This expression evaluates to true, i.e. the filter is stable.

We can verify this directly by residue calculus. The function

$$\frac{B(s)B(-s)}{A(s)A(-s)} = \frac{(2s+1)(-2s+1)}{(s+1)(s+3)(s-1)(s-3)}$$

This function has the poles $s_1 = -1$ and $s_2 = -3$ in the left half plane. The residues corresponding to these poles are

$$R_1 = \frac{(-2+1)(2+1)}{(-1+3)(-1-1)(-1-3)} = -\frac{3}{16}$$
$$R_2 = \frac{(-6+1)(6+1)}{(-3+1)(-3-1)(-3-3)} = \frac{35}{48}$$

It follows from residue calculus that the integral is

$$I = -\frac{3}{16} + \frac{35}{48} = \frac{13}{24}$$

Another approach (Warning!) could be to directly calculate the integral in (1). This has been seen to "work", i.e. possible to do analytically by Maple's `int` function , for $A(s)$ of lower degree than two. By the change of variable $t = i\,s$ equation (1) can be written as

$$I = \frac{1}{2\pi}\int_{-\infty}^{\infty} \frac{B(-it)B(it)}{A(-it)A(it)}dt$$

Maple code to evaluate this is given below

```
> B:=2*s+1:
> A:=s^2+4*s+3:
> 1/(2*Pi)*int(subs(s=-I*t,B)*subs(s=I*t,B)
    /(subs(s=-I*t,A)*subs(s=I*t,A)),t=-infinity..infinity);
```

You get the erroneous answer 35/24. The calculation goes wrong somewhere in Maple's `int` function. By changing the code a bit you get round this bug.

```
> B:=2*s+1:
> A:=s^2+4*s+3:
> 1/(2*Pi)*int(collect(subs(s=-I*t,B)*subs(s=I*t,B),t)
    /collect((subs(s=-I*t,A)*subs(s=I*t,A)),t)
                                    ,t=-infinity..infinity);
```

This again gives you the correct answer. □

EXAMPLE 2

Consider the setup in Figure 2 where $u(t)$ is white noise with unit variance. The filters are given by

$$A(z) = z^3 + a_1z^2 + a_2z + a_3$$
$$B(z) = b_0z^2 + b_1z + b_2$$
$$C(z) = c_0z^2 + c_1z + c_2$$

$E\, y_1(t)y_2(t)$ is calculated with the following Maple sequence.

```
> read '/home/johan/maple/var';
> A:=z^3+a1*z^2+a2*z+a3;
> B:=b0*z^2+b1*z+b2;
> C:=c0*z^2+c1*z+c2;
> vard(A,B,C);
```

The following result is obtained

$$
\begin{aligned}
&-(-b_2a_3^2c_2+b_2c_2-b_2a_3c_2a_1+b_0a_1^2c_2-b_0a_1c_1+b_2a_1^2c_0+b_2a_3c_0a_1+b_0a_1c_2a_3\\
&-b_0a_1c_0a_3-b_1a_3a_1c_1-b_1c_0a_1+b_1c_1+a_2b_2a_3c_1+a_2b_1c_2a_3-b_0a_3^2c_0-b_2a_1c_1\\
&-b_1c_2a_1-b_1c_1a_3^2-a_2^2b_0c_2-a_2^2b_2c_0-b_0c_2a_2-b_2a_2c_0+b_0c_0+b_0c_1a_3a_2\\
&+b_1a_3a_2c_0+a_2b_2c_2+a_2b_0c_0+a_2b_1c_1)/(-a_3^4-a_3^3a_1+a_3^2a_1^2+a_2a_3^2+2a_3^2\\
&+a_3^2a_2^2+a_3a_1^3-a_1a_3a_2^2-4a_3a_1a_2+a_3a_1-1+a_2^3-a_2-a_2a_1^2+a_1^2+a_2^2)
\end{aligned}
$$

The answer is correct if the coefficients of $A(z)$ is such that $A(z)$ is stable. Stability of the filters is a condition for the theorems to hold. Stability of the polynomial $A(z)$ can be checked with the following command in Maple.

```
> stabtestd(A);
```

This results in an expression that evaluate to true iff $A(z)$ is stable. □

EXAMPLE 3

In [4] a table of integrals on a similar form to (1) is supplied in an appendix, see Figure 3.

For instance we can try to reproduce the table value I_3. Factor $g_3(x)$ as

$$
\begin{aligned}
g_3(x) &= b_0x^4+b1x^2+b_2 = \\
&= (\sqrt{b_0}x^2+\sqrt{2\sqrt{b_0b_2}-b_1}x+\sqrt{b_2})(\sqrt{b_0}x^2-\sqrt{2\sqrt{b_0b_2}-b_1}x+\sqrt{b_2}) = \\
&= B(x)B(-x)
\end{aligned}
$$

and chose

$$A(x) = h_3(x) = a_0x^3+a_1x^2+a_2x+a_3$$

The table value, or the Maple implementation if you want, is checked by the following Maple sequence.

```
> read '/home/johan/maple/var';
> A:=a0*s^3+a1*s^2+a2*s+a3;
> B:=sqrt(b0)*s^2+sqrt(2*sqrt(b0)*sqrt(b2)-b1)*s+sqrt(b2);
> MapleRes:=filtvarc(A,B);
> I3:=(-a2*b0+a0*b1-a0*a1*b2/a3)/(2*a0*(a0*a3-a1*a2));
> simplify(I3-MapleRes);
```

The result is 0, which of course means that we get the same answer by the table in [4] and by the Maple procedure. The following expression evluates to true iff the polynomial $A(s)$ is stable.

```
> stabtestc(A);
```

□

APPENDIX

TABLE OF INTEGRALS

The following is a table of integrals of the type

$$I_n = \frac{1}{2\pi j} \int_{-\infty}^{\infty} dx \, \frac{g_n(x)}{h_n(x)h_n(-x)},$$

where

$$h_n(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n,$$
$$g_n(x) = b_0x^{2n-2} + b_1x^{2n-4} + \cdots + b_{n-1},$$

and the *roots of* $h_n(x)$ *all lie in the upper half plane.* The table lists the integrals I_n for values of n from 1 to 7 inclusive.[1]

$$I_1 = \frac{b_0}{2a_0a_1}$$

$$I_2 = \frac{-b_0 + \dfrac{a_0b_1}{a_2}}{2a_0a_1}$$

$$I_3 = \frac{-a_2b_0 + a_0b_1 - \dfrac{a_0a_1b_2}{a_2}}{2a_0(a_0a_3 - a_1a_2)}$$

$$I_4 = \frac{b_0(-a_1a_4 + a_2a_3) - a_0a_3b_1 + a_0a_1b_2 + \dfrac{a_0b_3}{a_4}(a_0a_3 - a_1a_2)}{2a_0(a_0a_3^2 + a_1^2a_4 - a_1a_2a_3)}$$

Figure 3. Table of integrals from [4]. Only table entries up to order four is displayed. The whole table contains integrals up to order seven, which is an expression that covers just over a half page. A similar table for systems up to order ten is given in [3].

Appendix A

This appendix contains a straightforward implementation of the algorithms described in the previous sections. The maple functions are named

`filtvard(A,B)`	Evaluate integrals on the form (2)
`filtvarc(A,B)`	Evaluate integrals on the form (1)
`vard(A,B,C)`	Evaluate integrals on the form (3)
`varc(A,B,C)`	Evaluate integrals on the form (4)
`stabtestd(A)`	Test stability of discrete time polynomial
`stabtestc(A)`	Test stability of continuous time polynomial

The functions are stored in the file `/home/johan/maple/var`, and are included in MapleV with the command `>read '/home/johan/maple/var';`.

```
#Functions to evaluate loss functions in continuous and discrete time.
#Author Johan Nilsson , Dept. of Aut. Control, LTH, SE, 930618
#Reference: Astrom Karl Johan, 1970.
#          Introduction to Stochastic Control Theory.
#          New York: Academic Press.
#          Page 115-142

############################################################
################ HELP TEXTS ################################
############################################################

`help/text/filtvarc` := TEXT(
`FUNCTION: filtvarc - calculate the variance of filtered white noise`,
`                     in continuous time. The polynomials A and B should`,
`                     be in the operator s.`,
`   `,
`CALLING SEQUENCE: filtvarc(A,B);`,
`   `,
`SEE ALSO: varc,filtvard,vardstabtestc,`):


`help/text/filtvard` := TEXT(
`FUNCTION: filtvard - calculate the variance of filtered white noise`,
`                     in discrete time. The polynomials A and B should`,
`                     be in the operator z.`,
`   `,
`CALLING SEQUENCE: filtvard(A,B);`,
`   `,
`SEE ALSO: vard,filtvarc,varc,stabtestd`):


`help/text/vard` := TEXT(
`FUNCTION: vard - evaluate the integral`,
`  `,
`                    / `,
`             1      |   B(z)*C(z^-1)  dz`,
`           ------  0  -------------- --`,
`            2piI    |   A(z)*A(z^-1)  z`,
`                  /  `,
`   `,
`     The polynomials A,B and C should be in the operator z.`,
`   `,
`CALLING SEQUENCE: vard(A,B,C);`,
`   `,
`SEE ALSO: filtvard,filtvarc,varc,stabtestd`):
```

```
'help/text/varc' := TEXT(
'FUNCTION: varc - evaluate the integral',
'  ',
'                  / ',
'            1     |   B(s)*C(-s)  ',
'          ------  |  ------------ ds',
'           2piI   |   A(s)*A(-s)  ',
'                 /  ',
'   ',
'     The polynomials A,B and C should be in the operator s.',
'   ',
'CALLING SEQUENCE: varc(A,B,C);',
'   ',
'SEE ALSO: filtvarc,filtvard,vard,stabtestc'):


'help/text/stabtestc' := TEXT(
'FUNCTION: stabtestc - test stability of polynomial in continuous time',
'  ',
'     The polynomial A should be in the operator s.',
'  ',
'CALLING SEQUENCE: stabtestc(A);',
'   ',
'SEE ALSO: stabtestd'):


'help/text/stabtestd' := TEXT(
'FUNCTION: stabtestd - test stability of polynomial in discrete time',
'  ',
'     The polynomial A should be in the operator z.',
'  ',
'CALLING SEQUENCE: stabtestd(A);',
'   ',
'SEE ALSO: stabtestc'):


#################################################################
################## CODE #########################################
#################################################################

################## filtvarc #####################################
filtvarc := proc(A,B)
   local k,Ak,Akt,Bk,V,alpha,beta,n:
   if not type(A,'polynom') and type(B,'polynom') then
     ERROR('A and B must be polynomials')
   fi:

   n:=degree(A,s):

   alpha:=array(1..n):
   beta:=array(1..n):

   Ak:=array(1..n):
   Akt:=array(1..n):
   Bk:=array(1..n):

   Ak[n]:=A:
   Akt[n]:=1/2*(Ak[n]-(-1)^(degree(Ak[n],s))*subs(s=-s,Ak[n])):
   Bk[n]:=B:
   alpha[n]:=coeff(Ak[n],s,n)/coeff(Ak[n],s,n-1):
   beta[n]:=coeff(Bk[n],s,n-1)/coeff(Ak[n],s,n-1):
```

```
  for k from n-1 by -1 to 1 do
    Ak[k]:=collect(Ak[k+1]-alpha[k+1]*s*Akt[k+1],s):
    Bk[k]:=collect(Bk[k+1]-beta[k+1]*Akt[k+1],s):
    Akt[k]:=collect(1/2*(Ak[k]-(-1)^(degree(Ak[k],s))*subs(s=-s,Ak[k])),s):
    alpha[k]:=coeff(Ak[k],s,k)/coeff(Ak[k],s,k-1):
    beta[k]:=coeff(Bk[k],s,k-1)/coeff(Ak[k],s,k-1):
  od:

  V:=0:
  for k from 1 to n do
    V:=V+beta[k]^2/(2*alpha[k]):
  od:

  simplify(V):
end:


################## filtvard ###################################
filtvard := proc(A,B)
  local k,Ak,Aks,Bk,V,alpha,beta,n:
  if not type(A,'polynom') and type(B,'polynom') then
    ERROR(`A and B must be polynomials`)
  fi:

  n:=degree(A,z):

  alpha:=array(0..n):
  beta:=array(0..n):
  Ak:=array(0..n):
  Bk:=array(0..n):

  Ak[n]:=collect(A,z):
  Bk[n]:=collect(B,z):
  alpha[n]:=coeff(Ak[n],z,0)/coeff(Ak[n],z,n):
  beta[n]:=coeff(Bk[n],z,0)/coeff(Ak[n],z,n):

  for k from n-1 by -1 to 0 do
    Aks:=z^(k+1)*subs(z=z^(-1),Ak[k+1]):
    Ak[k]:=collect(z^(-1)*(Ak[k+1]-alpha[k+1]*Aks),z):
    Bk[k]:=collect(z^(-1)*(Bk[k+1]-beta[k+1]*Aks),z):
    alpha[k]:=coeff(Ak[k],z,0)/coeff(Ak[k],z,k):
    beta[k]:=coeff(Bk[k],z,0)/coeff(Ak[k],z,k):
  od:

  V:=beta[0]^2:
  for k from 1 to n do
    V:=(1-alpha[k]^2)*V+beta[k]^2:
  od:

  simplify(V):
end:


################## vard ######################################
vard := proc(A,B,C)
  local k,n,Ak,Bk,Ck,Aks,alpha,beta,gamma,V:
  if not type(A,'polynom') and type(B,'polynom') and type(C,'polynom') then
    ERROR(`A,B and C must be polynomials`)
  fi:

  n:=degree(A,z):
```

```
  alpha:=array(0..n):
  beta:=array(0..n):
  gamma:=array(0..n):
  Ak:=array(0..n):
  Bk:=array(0..n):
  Ck:=array(0..n):

  Ak[n]:=collect(A,z):
  Bk[n]:=collect(B,z):
  Ck[n]:=collect(C,z):
  alpha[n]:=coeff(Ak[n],z,0)/coeff(Ak[n],z,n):
  beta[n]:=coeff(Bk[n],z,0)/coeff(Ak[n],z,n):
  gamma[n]:=coeff(Ck[n],z,0)/coeff(Ak[n],z,n):

  for k from n-1 by -1 to 0 do
    Aks:=z^(k+1)*subs(z=z^(-1),Ak[k+1]):
    Ak[k]:=collect(z^(-1)*(Ak[k+1]-alpha[k+1]*Aks),z):
    Bk[k]:=collect(z^(-1)*(Bk[k+1]-beta[k+1]*Aks),z):
    Ck[k]:=collect(z^(-1)*(Ck[k+1]-gamma[k+1]*Aks),z):
    alpha[k]:=coeff(Ak[k],z,0)/coeff(Ak[k],z,k):
    beta[k]:=coeff(Bk[k],z,0)/coeff(Ak[k],z,k):
    gamma[k]:=coeff(Ck[k],z,0)/coeff(Ak[k],z,k):
  od:

  V:=beta[0]*gamma[0]:
  for k from 1 to n do
    V:=(1-alpha[k]^2)*V+beta[k]*gamma[k]:
  od:

  simplify(V):
end:


################## varc ####################################
varc := proc(A,B,C)
  local k,Ak,Akt,Bk,Ck,V,alpha,beta,n,gamma,beta1,gamma1:
  if not type(A,'polynom') and type(B,'polynom') then
    ERROR(`A and B must be polynomials`)
  fi:
  if not ((degree(C,s)<degree(A,s)) and degree(B,s)<degree(A,s)) then
    ERROR(`B and C must be polynomials of lower order than A`)
  fi:

  n:=degree(A,s):

  alpha:=array(1..n):
  beta:=array(1..n):
  gamma:=array(1..n):
  beta1:=array(1..n):
  gamma1:=array(1..n):

  Ak:=array(1..n):
  Akt:=array(1..n):
  Bk:=array(1..n):
  Ck:=array(1..n):

  Ak[n]:=collect(A,s):
  Akt[n]:=collect(1/2*(Ak[n]-(-1)^(degree(Ak[n],s))*subs(s=-s,Ak[n])),s):
  Bk[n]:=collect(B,s):
  Ck[n]:=collect(C,s):
  alpha[n]:=coeff(Ak[n],s,n)/coeff(Ak[n],s,n-1):
```

```
  beta[n]:=coeff(Bk[n],s,n-1)/coeff(Ak[n],s,n-1):
  beta1[n]:=coeff(Bk[n],s,n-1)/coeff(Ak[n],s,n):
  gamma[n]:=coeff(Ck[n],s,n-1)/coeff(Ak[n],s,n-1):
  gamma1[n]:=coeff(Ck[n],s,n-1)/coeff(Ak[n],s,n):

  for k from n-1 by -1 to 1 do
    Ak[k]:=collect(Ak[k+1]-alpha[k+1]*s*Akt[k+1],s):
    Bk[k]:=collect(Bk[k+1]-beta[k+1]*Akt[k+1],s):
    Ck[k]:=collect(Ck[k+1]-gamma[k+1]*Akt[k+1],s):
    Akt[k]:=collect(1/2*(Ak[k]-(-1)^(degree(Ak[k],s))*subs(s=-s,Ak[k])),s):

    alpha[k]:=coeff(Ak[k],s,k)/coeff(Ak[k],s,k-1):
    beta[k]:=coeff(Bk[k],s,k-1)/coeff(Ak[k],s,k-1):
    gamma[k]:=coeff(Ck[k],s,k-1)/coeff(Ak[k],s,k-1):
    beta1[k]:=coeff(Bk[k],s,k-1)/coeff(Ak[k],s,k):
    gamma1[k]:=coeff(Ck[k],s,k-1)/coeff(Ak[k],s,k):

  od:

  V:=0:
  for k from 1 to n do
    V:=V+1/2*(gamma[k]*beta1[k]+beta[k]*gamma1[k]-beta[k]*gamma[k]/alpha[k]):
  od:

  simplify(V):
end:


################## stabtestc ###################################
stabtestc := proc(A)
  local k,Ak,Akt,alpha,n,V:
  if not type(A,'polynom') then
    ERROR(`A must be a polynomial`)
  fi:

  n:=degree(A,s):

  alpha:=array(1..n):

  Ak:=array(1..n):
  Akt:=array(1..n):

  Ak[n]:=A:
  Akt[n]:=1/2*(Ak[n]-(-1)^(degree(Ak[n],s))*subs(s=-s,Ak[n])):
  alpha[n]:=coeff(Ak[n],s,n)/coeff(Ak[n],s,n-1):
  V:=(coeff(Ak[n],s,n)>0) and (coeff(Ak[n],s,n-1)>0);

  for k from n-1 by -1 to 1 do
    Ak[k]:=collect(Ak[k+1]-alpha[k+1]*s*Akt[k+1],s):
    Akt[k]:=collect(1/2*(Ak[k]-(-1)^(degree(Ak[k],s))*subs(s=-s,Ak[k])),s):
    alpha[k]:=coeff(Ak[k],s,k)/coeff(Ak[k],s,k-1):
    V:=V and coeff(Ak[k],s,k-1)>0;
  od:

  V:
end:


################## stabtestd ###################################
stabtestd := proc(A)
  local k,Ak,Aks,V,alpha,n:
  if not type(A,'polynom') then
```

```
    ERROR('A must be a polynomial')
  fi:

  n:=degree(A,z):

  alpha:=array(0..n):
  Ak:=array(0..n):

  Ak[n]:=collect(A,z):
  alpha[n]:=coeff(Ak[n],z,0)/coeff(Ak[n],z,n):
  V:=coeff(Ak[n],z,n)>0:

  for k from n-1 by -1 to 0 do
    Aks:=z^(k+1)*subs(z=z^(-1),Ak[k+1]):
    Ak[k]:=collect(z^(-1)*(Ak[k+1]-alpha[k+1]*Aks),z):
    alpha[k]:=coeff(Ak[k],z,0)/coeff(Ak[k],z,k):
    V:=V and coeff(Ak[k],z,k)>0:
  od:

  V:
end:
```

References

[1] Karl J. Åström. *Introduction to Stochastic Control Theory*. Academic Press, Inc., 1970.

[2] Karl Johan Åström. Evaluation of quadratic loss functions for linear systems. In *Symposium on Fundamentals of Discrete-Time Systems*, Chicago, Illinois, 1992. A meeting in honor of Professor Eliahu I. Jury.

[3] James F. Kaiser George C. Newton JR., Leonard A. Gould. *Analytical Design of Linear Feedback Controls*. John Wiley & Sons, Inc., 1957.

[4] Ralph S. Phillips Hubert M. James, Nathaniel B. Nichols. *Theory of Servomechanisms*. McGraw-Hill Book Company, Inc., 1947.