A minimal partial realization algorithm for MIMO systems VII : An Ada program

Marc Van Barel
Dirk Kinnaes
Adhemar Bultheel

Report TW 110, March 1988

Department of Computer Science, K.U.Leuven

ABSTRACT

In this report we describe an Ada program to compute the building blocks of the canonical matrix continued fraction that has for its convergents the successive solutions of the minimal partial realization problem (see [2]) or equivalently the solutions of the matrix Padé problem as described in [3]. The algorithm uses exact rational arithmetic.

1. The problems solved

1.1 The minimal partial realization problem

In the minimal partial realization problem we are given a number of Markov parameters.

$F_1, F_2, F_3, \ldots, F_\omega \in \mathbb{Q}^{p \times m}$

and we try to find for $n = 1,2,\ldots,\omega$ the matrix polynomials

$Y_{0,n}(z) \in \mathbb{Q}^{p \times m}[z]$ and $M_{0,n}(z) \in \mathbb{Q}^{m \times m}[z]$

such that

1. $Y_{0,n}(z)\, M_{0,n}(z)^{-1} = \sum_{k=1}^{\omega} F_k z^{-k} + O(z^{-n-1})\,,\ z \to \infty$
2. deg det $M_{0,n}(z)$ is minimal.

p is the number of outputs, m is the number of inputs and n is called the level. Solutions can be obtained by performing unimodular transformations V_n on a G-matrix which is initialized as

$$G_{0,0}(z) = \begin{bmatrix} I_p & O \\ O & I_m \\ -I_p & N_0(z) \end{bmatrix} \quad \text{with} \quad N_0(z) = \sum_{k=1}^{\omega} F_k z^{-k}$$

where ω is the number of Markov parameters given.

$$G_{0,n}(z) = G_{0,n-1}(z)\, V_n(z) = \begin{bmatrix} X_{0,n}(z) & Y_{0,n}(z) \\ L_{0,n}(z) & M_{0,n}(z) \\ P_{0,n}(z) & N_{0,n}(z) \end{bmatrix}\,.$$

The blocks of the $G_{0,n}$ matrix satisfy

$[I_p \ -N_0(z) \ I_p]\, G_{0,n}(z) = 0\,,$

so the $N_{0,n}(z)$ can be considered as residuals, i.e.

$$(\sum_{k=1}^{\omega} F_k\, z^{-k})\, M_{0,n}(z) - Y_{0,n}(z) = N_{0,n}(z)\,.$$

The columns of $M_{0,n}(z)$ are called solution vectors and the colums of $L_{0,n}(z)$ are called auxiliary vectors.
By imposing extra conditions, the $V_n(z)$ matrices can be made unique. The algorithm is then called canonical (see [2]).

The minimal partial realizations $Y_{0,n}\, M_{0,n}^{-1}$ are the convergents of the continued fraction

$$\cfrac{Y_0 + X_0 \cfrac{Y_1 + X_1 \cfrac{\cdots}{\cdots}}{M_1 + L_1 \cfrac{\cdots}{\cdots}}}{M_0 + L_0 \cfrac{Y_1 + X_1 \cfrac{\cdots}{\cdots}}{M_1 + L_1 \cfrac{\cdots}{\cdots}}} \qquad \text{with} \quad \frac{A}{B} = AB^{-1}$$

where

$$V_n = \begin{bmatrix} X_n & Y_n \\ L_n & M_n \end{bmatrix} .$$

1.2 The Padé problem

In this case we are given the matrices

$F_0, F_1, F_2, \ldots, F_{\omega-1} \in \mathbb{Q}^{p \times m}$

and we try to find for $n = 1,2,\ldots,2(\omega-1)$ the matrix polynomials

$N_{0,n}(z) \in \mathbb{Q}^{p \times m}[z]$ and $M_{0,n}(z) \in \mathbb{Q}^{m \times m}[z]$

such that

1. $\left(\sum_{k=0}^{\omega-1} F_k z^k \right) M_{0,n}(z) - N_{0,n} = O(z^{\omega})$, $z \to 0$
2. $N_{0,n}(z)\, M_{0,n}(z)^{-1} = O(z^{\omega-n-1})$, $z \to \infty$
3. deg det $M_{0,n}(z)$ is as low as possible.

The solutions of this problem will have row degrees for $N_{0,n}(z)$ bounded by $\nu_i(n)$, and column degrees for $M_{0,n}(z)$ bounded by $\mu_j(n)$ such that

$$p \cdot (\omega-1) \geqslant \sum_{i=1}^{p} \nu_i(n) + \sum_{j=1}^{m} \mu_j(n)$$

and they are therefore called matrix Padé approximants (see [3]). The solution of this problem requires the same computations as the solution of the minimal partial realization problem. The difference is a shift in the powers of z and another interpretation of the matrices. We have to perform unimodular transformations $V_n(z)$ on a G matrix which is initialized as

$$G_{0,0}(z) = \begin{bmatrix} z^{\omega} I_p & O \\ O & I_m \\ -z^{\omega} I_p & N_0(z) \end{bmatrix} \quad \text{with} \quad N_0(z) = \sum_{k=0}^{\omega-1} F_k z^k$$

to give

$$G_{0,n}(z) = G_{0,n-1}(z)\, V_n(z) = \begin{bmatrix} X_{0,n}(z) & Y_{0,n}(z) \\ L_{0,n}(z) & M_{0,n}(z) \\ P_{0,n}(z) & N_{0,n}(z) \end{bmatrix} .$$

Now $N_{0,n}(z)$ is interpreted as a numerator matrix and $Y_{0,n}(z)$ is a residual matrix since we have

$$\left(\sum_{k=0}^{\omega-1} F_k z^k \right) M_{0,n}(z) - N_{0,n}(z) = Y_{0,n}(z) .$$

As before we call the colums of $M_{0,n}(z)$ solution vectors and the colums of $L_{0,n}(z)$ the auxiliary vectors. n is called the level.

If the $V_n(z)$ matrices are made canonical (i.e. unique) we call the solution canonical.

The convergents of the continued fraction

$$N_0 - z^{\omega+1} I_p \; \cfrac{}{M_1 + L_1 \cfrac{Y_2 + X_2 \cfrac{\cdots}{\cdots}}{M_2 + L_2 \cfrac{\cdots}{\cdots}}} \quad \text{with} \quad \frac{A}{B} = AB^{-1}$$

are the successive solutions $N_{0,n}\, M_{0,n}^{-1}$.

In [3] we give some algorithms to perform the computations of the $V_n(z)$ matrix.

The Ada program implements the minimal partial realization algorithm and if the Padé problem has to be solved, a shift with z^{ω} is applied before printing out the results. This shift has to be applied to the numerator and residual parts of the G matrix only.

2. Implementation of the algorithms

We have based the procedure `mprpade` that solves the minimal partial realization (`real`) or Padé (`Pade`) problem on several Ada packages :

1. The package `Universal_Integer_Arithmetic`.
   This is an implementation for infinite precision integer arithmetic.
   This package is originally written by Gerry Fisher [1].
   However, when we used this package, it was clear from the start that if the Ada compiler did no garbage collection, it needed an unreasonable amount of memory. So we have rewritten the package to include garbage collection.
2. The package `Universal_Integer_Io`.
   This implements the input and output of `Universal_Integers`.
3. The package `Universal_Real_Arithmetic`.
   This is in fact an implementation of infinite precision rational arithmetic.
   As the `Universal_Integer_Arithmetic` package, it was originally written by Gerry Fisher [1] but we adapted it to include garbage collection.
4. The package `Universal_Real_Io`.
   This implements the input and output of `Universal_Real` variables.
5. The package `Laurent_Series_Arithmetic`.
   This implements operations with `Laurent_Series` needed in `mprpade`.
   The garbage collection which is included is completely similar to the one in the other packages.
6. The package `Laurent_Series_Io`.
   This implements the output of `Laurent_Series`.

3. The garbage collection

If someone wants to write his own procedures to add to this program he shall have to incorporate the garbage collection. Therefore we give here some details on how this can be obtained.

Objects of type `Universal_Integer` and `Laurent_Series` are represented as pointers to dynamically allocated arrays. The actual length of such an array must only be specified when the array is allocated. In this way, there is no need to specify a static upper bound limiting the set of representable universal integer, universal real numbers and Laurent series. Unfortunately the original implementation of the universal arithmetic packages written by Gerry Fisher does not release temporary objects, producing enormous amounts of garbage. For large applications this is only acceptable if the Ada compiler supports automatic garbage collection.

In order to circumvent this problem we have reimplemented the packages such that temporary objects are deallocated (freed) as soon as possible. We only assumed that the Ada compiler has an implementation of the predefined generic procedure `Unchecked_Deallocation`.

A very simple implementation would be to add a single procedure universal integer or real X. Unfortunately this would imply that the object resulting from a universal integer, real or Laurent series operation can not be freed unless it is stored in some variable. For example, after the assignment

```
a := b + (c * d) - e;
```

the objects resulting from the subexpressions (c * d) and (b + (c * d) have become inaccessible and can not be freed any more. So one would have to write something like

```
a1 := c * d;
a2 := b + a1;
a := a2 - e;
free (a1);
free (a2);
```

Obviously much of the expressional power has been lost.

Therefore we have adopted another solution. We have reimplemented the operations such that the operations themselves release as much of the temporary storage as possible. That is, each operation frees its own universal integer, universal real or Laurent series arguments upon completion of the operation, but only if those arguments are not accessible from elsewhere. For example, when evaluating the expression

```
b + (c * d) - e,
```

the object resulting from `(c * d)` is freed by the "+" operation, and the object resulting from `(b + (c * d))` is freed by the "-" operation. Of course the objects in the variables b, c, d and e should not be freed yet because they are still accessible after the evaluation of the expression. These objects must be freed explicitly when they are no longer needed, by calling `free(b)`, `free(c)`, etc.

How are the operations going to decide whether an argument is still accessible ? This is only possible if a special attribute is added to the representation of a universal integer, and similarly to the representation of a Laurent series indicating the number of variables that contain a reference to that object. Then only those arguments are freed that have a reference count of zero. The procedure free will decrement the reference count by one (if not already zero), and free the object when the resulting reference count is zero. Obviously the normal assignment operation of Ada should not be used, since it will not increment the reference count of the assigned object. The procedure `Assign` should be used instead, for example

```
Assign (a, b + (c * d) - e);
```

If the user of the universal integer, real or Laurent series packages chooses to write procedures or functions with arguments of types `Universal_Integer`, `Universal_Real` or `Laurent_series` he is responsible for the deallocation of their arguments in the same way as the procedures and functions of the packages are. The only difference is that the user of the packages has no access to the reference count attribute. The following scenario should be followed when writing procedures and functions with in-parameters of type `Universal_Integer`, `Universal_Real` or `Laurent_Series`:

For procedures:

1. Declare a local variable for each in-parameter of type `Universal_Integer`, `Universal_Real` or `Laurent_Series`.
2. Assign the in-parameters to the corresponding local variables.
3. Replace each occurrence of the in-parameters in the body of the procedure by the corresponding local variable.
4. Free all local variables of types `Universal_Integer` or `Universal_Real` or `Laurent_Series` before each exit point from the procedure.

For functions, the same strategy should be applied, except that it is not possible to free a local variable that occurs in a return expression. An occurrence of such a variable (say X) in a return expression must be replaced by `Result (X)`. An object should not occur more than once in a single return expression.

For example:

```
function Square (X: Universal_Integer) return Universal_Integer is
 XI, R: Universal_Integer;
begin
 Assign (XI, X);
 Assign (R, XI * XI);
 Free (XI);
 return Result (R);
end Square;
```

The following implementation would be incorrect:

```
function Square (X: Universal_Integer) return Universal_Integer is
 XI: Universal_Integer;
begin
 Assign (XI, X);
 return Result (XI) * Result (XI);
end Square;
```

because `Result (XI)` appears twice in the return expression.

WARNING: The procedures `Assign`, `Free` and the function `Result` must be applied with great care. The implementation will not be able to detect an incorrect use of these subprograms, possibly resulting in dangling references or ineffective garbage collection. In order to be able to detect whether garbage collection has been effective, two functions have been added to the `Universal_Integer_Arithmetic` package:

```
function Ui_New return Natural;

-- Returns the number of Universal_Integer objects that have been
-- allocated, including those that have been deallocated again.
-- The number of Universal_Integer objects that are still allocated is
-- Ui_New - Ui_Free.

function Ui_Free return Natural;

-- Returns the number of Universal_Integer objects that have been
-- deallocated.
```

If the scenario described above is followed rigorously, both functions should return the same result.

Similarly two functions `Ls_New` and `Ls_Free` have been added to the `Laurent_Series_Arithmetic` package.

4. The procedure mprpade

1. The program will first ask you what you want to solve : the minimal partial realization problem (in this case type `real`) or the Padé problem (in this case type `Pade`).
2. It will ask you next if you want output at each level or not (type `y` or `n`). If you choose `n`, only the final result will be printed.
3. If you choose `y` , it will give you the choice to ask for a check or not (`y` or `n`).
   For the `real` problem, this check consists in printing out the difference between the given Markov parameters and those of the solution. When you are at level n, the difference should be $O\,(z^{-n-1})$ around $z = \infty$. For the `Pade` problem, the (matrix) power series around $z = \infty$ of the matrix Padé approximant is printed. At level n, this should be $O\,(z^{\omega-n-1})$ around $z = \infty$ where ω indicates the number of coefficients given at the beginning.
4. The program will also ask wether you want the complete G matrix to be printed or not (`y` or `n`).
   If not, only numerator and denominator will be printed.
5. The next input asked for is the dimensions of the F_k.
   m = the number of inputs (`real`) = number of colums (`Pade`)
   p = the number of outputs (`real`) = number of rows (`Pade`)
6. The number of F_k given is asked next
   These numbers should be given as `Universal_Real` numbers. I.e. they should be given as an integer numerator-denominator pair which is interpreted as a rational number. If the denominator is 1, it can be left out. If both are given, they should be separated by at least one character which is not a digit.

   E.g.

   ```
   (15/4)
   15.4
   15/4
   abc15/*4
   ```

   all represent 15/4

and

```
(13,1)
13@
13 @
13*@
```

all represent 13/1 (@ represents a line terminator).

7. Then the program will prompt you for the coefficients F_k.

In the appendix, some sample sessions with the mprpade program are given. The first one is an example due to Antoulas of a problem with 3 inputs and 2 outputs. The second one computes the Padé approximants for the scalar exponential.

5. Final note

We are aware of the fact that the program is not given its most efficient implementation. We welcome all remarks or suggetions for improvement.

The program is available on request. Please contact

Name : Marc Van Barel
UUCP : marc@kulcs.UUCP
BITNET: marc@kulcs.bitnet
ARPA : na.vanbarel@SCORE.STANFORD.EDU
Phone : +(32) 16 20 06 56 x3562

Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200 A
B-3030 Leuven (Heverlee)
BELGIUM

or

Name : Adhemar Bultheel
UUCP : ade@kulcs.UUCP
BITNET: ade@kulcs.bitnet
ARPA : na.bultheel@SCORE.STANFORD.EDU
Phone : +(32) 16 20 06 56 x3540

Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200 A
B-3030 Leuven (Heverlee)
BELGIUM

References :

[1] G. FISHER : Universal arithmetic packages. Ada Letters vol. 3 (no. 6), p 30-47, May, June 1984.

[2] M. VAN BAREL, A. BULTHEEL : A minimal partial realization algorithm for MIMO systems, V : A canonical matrix continued fraction. Report TW100, January 1988. Department of Computer Science, K.U.Leuven.

[3] A. BULTHEEL, M. VAN BAREL : A matrix Euclidean algorithm and matrix Padé approximation. Report TW104, January 1988. Department of Computer Science, K.U.Leuven.

```
What do you want to solve ?
  - the minimal partial realization problem ( type "real" )
  - the matrix Pade approximation problem   ( type "Pade" )
real

Do you want output at each level ? ( type "y" or "n" )
n
Do you want the complete G matrix at the output ? ( type "y" or "n" )
n

Give the number of inputs and outputs :
3 2
Give the number of Markov parameters : 5
Give Markov parameter    1 :
(   1,   1) : 1
(   1,   2) : 1
(   1,   3) : 1
(   2,   1) : 2
(   2,   2) : 2
(   2,   3) : 0
Give Markov parameter    2 :
(   1,   1) : 2
(   1,   2) : 2
(   1,   3) : 1
(   2,   1) : 4
(   2,   2) : 4
(   2,   3) : 0
Give Markov parameter    3 :
(   1,   1) : 4
(   1,   2) : 6
(   1,   3) : 3
(   2,   1) : 8
(   2,   2) : 10
(   2,   3) : 2
Give Markov parameter    4 :
(   1,   1) : 8
(   1,   2) : 11
(   1,   3) : 6
(   2,   1) : 16
(   2,   2) : 21
(   2,   3) : 7
Give Markov parameter    5 :
(   1,   1) : 17
(   1,   2) : 51/2
(   1,   3) : 31/2
(   2,   1) : 34
(   2,   2) : 46
(   2,   3) : 21
The final result  at level    5 :

The numerator matrix Y connected with the solution  vectors :
(   1,   1) :
              z **    0 : ( 1195, 1)
              z **    1 : ( 71, 2)
              z **    2 : ( 1, 1)
```

```
(  1,  2) :
              z **   0 : ( 1651, 1)
              z **   1 : ( 95, 2)
              z **   2 : ( 1, 1)

(  1,  3) :
              z **   0 : ( 0, 1)

(  2,  1) :
              z **   0 : ( 2390, 1)
              z **   1 : ( 71, 1)
              z **   2 : ( 2, 1)

(  2,  2) :
              z **   0 : ( 3300, 1)
              z **   1 : ( 95, 1)
              z **   2 : ( 2, 1)

(  2,  3) :
              z **   0 : (-2, 1)


The solution vector  matrix M :
(  1,  1) :
              z **   0 : (-9559, 4)
              z **   1 : ( 2249, 2)
              z **   2 : ( 69, 2)
              z **   3 : ( 1, 1)

(  1,  2) :
              z **   0 : (-6605, 2)
              z **   1 : ( 1556, 1)
              z **   2 : ( 91, 2)

(  1,  3) :
              z **   0 : ( 2, 1)

(  2,  1) :
              z **   0 : ( 3, 4)
              z **   1 : (-1, 2)
              z **   2 : (-1, 1)

(  2,  2) :
              z **   0 : ( 2, 1)
              z **   1 : (-2, 1)
              z **   2 : ( 0, 1)
              z **   3 : ( 1, 1)

(  2,  3) :
              z **   1 : (-1, 1)

(  3,  1) :
              z **   0 : ( 1, 1)

(  3,  2) :
              z **   0 : (-1, 2)
```

```
(   3,   3) :
                z **    0 : (-1, 1)
                z **    1 : ( 1, 1)


The variables that control the garbage collection system are :
Ui_New  :       26249
Ui_Free :       26249
Ls_New  :         653
Ls_Free :         653
```

```
What do you want to solve ?
  - the minimal partial realization problem ( type "real" )
  - the matrix Pade approximation problem   ( type "Pade" )
Pade

Do you want output at each level ? ( type "y" or "n" )
y
Do you want a check at each level ? ( type "y" or "n" )
n
Do you want the complete G matrix at the output ? ( type "y" or "n" )
n

Give the number of columns and rows of the power series coefficients :
1 1
Give the number of coefficients of the power series in z : 5
Give the coefficient of the term of degree    0 :
(   1,   1) : 1
Give the coefficient of the term of degree    1 :
(   1,   1) : 1
Give the coefficient of the term of degree    2 :
(   1,   1) : 1/2
Give the coefficient of the term of degree    3 :
(   1,   1) : 1/6
Give the coefficient of the term of degree    4 :
(   1,   1) : 1/24
The result at level    1 :

The numerator matrix N connected with the solution  vectors :
(   1,   1) :
               z **    0 : (-4, 1)
               z **    1 : (-3, 1)
               z **    2 : (-1, 1)
               z **    3 : (-1, 6)


The solution vector  matrix M :
(   1,   1) :
               z **    0 : (-4, 1)
               z **    1 : ( 1, 1)


The result at level    2 :

The numerator matrix N connected with the solution  vectors :
(   1,   1) :
               z **    0 : (-4, 1)
               z **    1 : (-3, 1)
               z **    2 : (-1, 1)
               z **    3 : (-1, 6)


The solution vector  matrix M :
(   1,   1) :
               z **    0 : (-4, 1)
               z **    1 : ( 1, 1)
```

```
The result at level    3 :

The numerator matrix N connected with the solution  vectors :
(   1,    1) :
                z **    0 : ( 12, 1)
                z **    1 : ( 6, 1)
                z **    2 : ( 1, 1)


The solution vector  matrix M :
(   1,    1) :
                z **    0 : ( 12, 1)
                z **    1 : (-6, 1)
                z **    2 : ( 1, 1)


The result at level    4 :

The numerator matrix N connected with the solution  vectors :
(   1,    1) :
                z **    0 : ( 12, 1)
                z **    1 : ( 6, 1)
                z **    2 : ( 1, 1)


The solution vector  matrix M :
(   1,    1) :
                z **    0 : ( 12, 1)
                z **    1 : (-6, 1)
                z **    2 : ( 1, 1)


The result at level    5 :

The numerator matrix N connected with the solution  vectors :
(   1,    1) :
                z **    0 : (-24, 1)
                z **    1 : (-6, 1)


The solution vector  matrix M :
(   1,    1) :
                z **    0 : (-24, 1)
                z **    1 : ( 18, 1)
                z **    2 : (-6, 1)
                z **    3 : ( 1, 1)


The result at level    6 :

The numerator matrix N connected with the solution  vectors :
(   1,    1) :
                z **    0 : (-24, 1)
                z **    1 : (-6, 1)
```

```
The solution vector  matrix M :
(   1,   1) :
                z **    0 : (-24, 1)
                z **    1 : ( 18, 1)
                z **    2 : (-6, 1)
                z **    3 : ( 1, 1)


The result at level    7 :

The numerator matrix N connected with the solution  vectors :
(   1,   1) :
                z **    0 : ( 24, 1)


The solution vector  matrix M :
(   1,   1) :
                z **    0 : ( 24, 1)
                z **    1 : (-24, 1)
                z **    2 : ( 12, 1)
                z **    3 : (-4, 1)
                z **    4 : ( 1, 1)


The result at level    8 :

The numerator matrix N connected with the solution  vectors :
(   1,   1) :
                z **    0 : ( 24, 1)


The solution vector  matrix M :
(   1,   1) :
                z **    0 : ( 24, 1)
                z **    1 : (-24, 1)
                z **    2 : ( 12, 1)
                z **    3 : (-4, 1)
                z **    4 : ( 1, 1)



The variables that control the garbage collection system are :
Ui_New  :        6003
Ui_Free :        6003
Ls_New  :         151
Ls_Free :         151
```