FACTA UNIVERSITATIS (NIŠ)
SER. MATH. INFORM. Vol. 28 No 1 (2013), 75–86

SIMPLIFICATIONS OF RATIONAL MATRICES BY USING UML*

Milan B. Tasić, Ivan P. Stanimirović

Abstract. The simplification process on rational matrices consists of simplifying each entry represented by a rational function. We follow the classic approach of dividing the numerator and denominator polynomials by their common GCD polynomial, and provide the activity diagram in UML for this process. A rational matrix representation as the quotient of a polynomial matrix and a polynomial is also discussed here and illustrated via activity diagrams. Also, a class diagram giving the links between the class of rational matrices and the classes of rational functions and polynomials is obtained.

Keywords: UML, rational matrix, simplification, activity diagrams, class diagrams, Visual C++.

1. Introduction

Rational (polynomial) matrix is the matrix consisting of entries being rational functions (polynomials). Therefore, rational matrices are the generalization of polynomial ones. These are a mathematical tool often used for dynamical systems. Furthermore, polynomial matrices can be used for advanced controller design and in some signal processing applications. Matrix decompositions are often considered in many applications, such as finding matrix inverses and generalized inverses [2]. Decomposition of a polynomial matrix is frequently observed to evaluate polynomial generalized inverses, as explained in [3, 5]. Also, some well-known polynomial algorithms of computer algebra are provided in [11].

The Unified Modeling Language (UML) is a graphical language for visualizing, specifying and constructing classes of software intensive systems [1]. UML is the unification of a series of efforts to build notations for expressing models of Object Oriented Analysis and Design (OOAD), and therefore, UML represents the standard for Object Oriented modeling.

Received September 21, 2012.; Accepted February 17, 2013.
2010 *Mathematics Subject Classification.* Primary 65F30; Secondary 68N15
*The authors were supported in part by research project 174013 of the Serbian Ministry of Education and Science

The standard definition of Object Oriented Analysis (OOA) implies developing requirements and specifications expressed as an object model (population of interacting objects) of a system, as opposed to the traditional data or functional views. OOA is a discovery process which clarifies and documents the requirements of a system and focuses on understanding the problem domain. Significantly, it discovers and documents the key problem domain classes, concerned with developing an object-oriented model of the problem domain. The identified objects reflect the entities that are associated with the problem to be solved.

Object Oriented Design (OOD) definition is concerned with developing object-oriented models of a software/system to implement the requirements identified during OOA. It creates abstractions and mechanisms necessary to meet behavioral requirements determined during analysis. Therefore, OOD provides an object-oriented model of a software system to implement the identified requirements.

Notice that in UML a class (as the structural element) represents a description of a set of objects that share the same attributes, operations and relations. Classes are usually graphically rendered as rectangles often including class names, attributes and operations. Interface is a collection of operations that specifies a service of a class. It describes the externally visible behavior of a class and defines a set of operations (but not their implementations). Usually, interfaces are graphically represented as a circle with a name below. Use Case is the structural element that provides the description of a sequence of actions that a system performs to deliver an observable result to a particular actor and is used to structure the behavioral elements in a model. Use Cases are graphically depicted as ellipses drawn with a solid line.

The most famous concept of UML is the diagram. Diagrams are graph representations of a set of elements and relationships with the nodes being elements and the edges being relationships. They are the projections of systems and can visualize a system from various perspectives. UML is characterized by nine major diagrams: 1) class; 2) object; 3) use case; 4) sequence; 5) collaboration; 6) statechart; 7) activity; 8) component; 9) deployment.

Many algorithms involving the processing of rational matrices and their simplifications were introduced. The algorithm generating the full-rank LDL* factorization of a polynomial Hermitian matrix was presented in [6]. Therefore, the simplification procedure is performed on the entries of rational matrices L and D, which are obtained. Two methods for the computation of $A^{(2)}_{T,S}$ inverses of a given polynomial matrix A are derived in [6, 7]. These methods use the full-rank decomposition procedures (LDL^* and QDR decomposition), and Algorithm 3.2 from [10] to compute the inverse of a polynomial matrix. Notice that, in order to apply Algorithm 3.2 from [10] to a rational matrix, one needs to transform it to the form involving the quotient of a polynomial matrix and a polynomial, in which case the polynomial acts as a constant in the evaluation of the inverse matrix. Also, a method based on the LDL^* factorization of matrix product A^*A, for symbolic computation of Moore-Penrose inverse matrix is developed in [9]. Some implementation methods of algorithms including polynomial matrices in MATHEMATICA

were derived in [8].

Example 1.1. In the paper [6] the following matrix entry was encountered:

$$a(x) = \frac{x^5 - 11x^4 + 30x^3 + 22x^2 - 95x - 75}{x^4 + 2x^3 - 24x^2 - 50x - 25}.$$

Due to the simplification that was carried out, their greatest common divisor (GCD) was obtained as:

$$GCD(x) = (x+1)^2 \cdot (x-5),$$

and therefore, the simplified entry is

$$a(x) = \frac{x^2 - 8x + 15}{x + 5}.$$

This example shows how crucial the simplification is, because of the high storage requirements and low performances when dealing with non-simplified intermediate results. Therefore, we implemented this simplification process in Visual C++ to provide a graphical interface and for demonstration purposes (see Figure 7.1 in Appendix).

Our main motivation is to demonstrate the simplification procedures used on rational matrices in the above mentioned algorithms. This demonstration is due to the diagrams in UML, which will later provide a good basis for the implementations. Also, a goal is to identify the role of each UML diagram (use cases, statecharts and activity diagrams) in the performance process under an object-oriented perspective. The final objective is to obtain a set of annotated UML diagrams that should be the input to create a performance model (in terms of some performance modeling paradigm).

Our goal is to use UML to develop a model of the class of rational matrices. The benefits of applying UML on such complex class types are numerous. Notice that UML provides views for development and deployment, and it is process-independent. UML is highly suitable for use with the processes that are iterative and incremental. Therefore, the class of rational matrices and methods for rational matrix manipulations can be modeled in UML. Also, UML provides an expressive, visual modeling language for developing meaningful models. UML's main characteristic is it supports specifications that are independent of particular programming languages and development processes, and provides a basis for understanding specification languages.

The rest of the paper is organized as follows. In the second section a class diagram of the class of rational matrices is provided, also illustrating the correlations with the classes of rational functions and polynomials. In the third section the use-case diagram is depicted showing the position of the simplification procedures in the algorithm evaluation. Some activity diagrams are given in the fourth section determining some simplification procedures for rational matrices. Illustrative examples are provided in the fifth section in order to demonstrate simplification techniques. In appendix we give the partial C++ code for the simplification function on the class of rational functions.

2. Class diagrams

The most widely used diagram of UML is the class diagram. It is used to model the static design view of a system and to specify the structure, interfaces and relationships between classes that underlie the system architecture. Class diagram stands as the primary diagram for generating codes from UML models.

The purpose of a class diagram is to depict the classes within a model. In an object oriented application, classes have attributes (member variables), operations (member functions) and relations with other classes. The UML class diagram can depict all these things quite easily.

First, notice that the class of rational matrices requires the class of rational functions, because the entries of a rational matrix are given in the form of rational functions. Then, since any rational function is a quotient of two polynomials, the class Polynomial should be written as the basic class, along with the class of rational numbers for the coefficients of the polynomials.

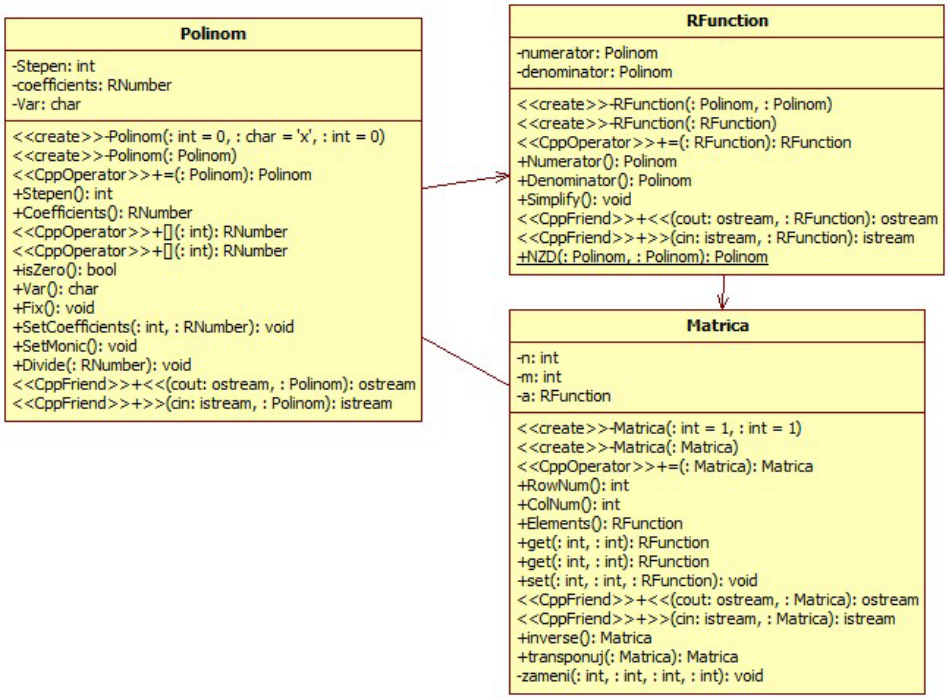

FIG. 2.1: An overview of the application model for three classes: *Polynomial*, *RationalFunction* and *RationalMatrix*.

3. Use case diagram

In UML a use case diagram shows actors and use cases together with their relationships. The relationships are associations between the actors and the use cases, generalizations between the actors, and generalizations, extends and includes among the use cases [4].

A use case represents a coherent unit of functionality provided by a system, a subsystem or a class as manifested by sequences of messages exchanged among the system (subsystem, class) and one or more actors together with actions performed by the system (subsystem, class). The use cases may optionally be enclosed by a rectangle that represents the boundary of the containing system or classifier [4].

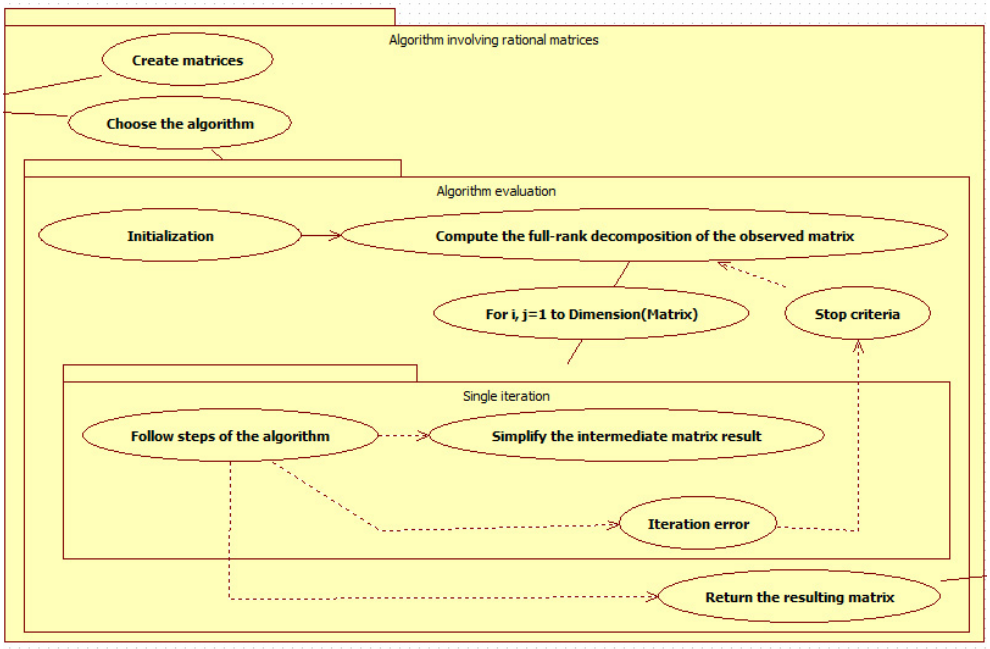

FIG. 3.1: Use-Case diagram of the rational functions algorithm including matrix simplification.

On the diagram in Figure 3.1 we obtain the role of a software developer in order to apply some of the algorithms on rational matrices. A main task is to apply the simplification process correctly, when needed. A condition for the stopping of the process is the iteration error (in a small number of cases) or the ending of the evaluation process (i, j equal to the dimension of the initial matrix).

4. Activity diagrams

Activity diagram provides visual descriptions of the system execution and the flow of activities. These diagrams focus on the activities that are performed and who (or what) is responsible for the performance of those activities.

The elements of an activity diagram are action nodes, control nodes, and object nodes. There are three types of control nodes: initial and final (final nodes have two varieties, activity final and flow final), decision and merge, and fork and join.

Actions are the elemental unit of behavior in an activity diagram. Activities can contain many actions which are what activity diagrams depict.

In the paper [6] the algorithm generating the full-rank LDL* factorization of a polynomial Hermitian matrix was presented. Although the observed matrix is polynomial, the resulting matrices L and D are rational, for the general case. Therefore, explicit formulaes for the evaluation of the coefficients of the entries of rational matrices L and D are obtained. In [6] we created a module for testing and verification purposes, in MATHEMATICA. The implementation was easily done since MATHEMATICA is the package for symbolic computations and has built-functions for manipulations with the expressions. The function Simplify[ ] performs a sequence of algebraic and other transformations on a given expression and returns the simplest form it finds [12]. Of course, one needs to tackle the problem of rational matrix simplification when implementing in procedural programming languages.

The most crucial simplification process on rational matrices is to simplify each entry determined by some rational function. The classic approach to rational function simplification is to evaluate the GCD of numerator and denominator polynomials and then divide these polynomials by the GCD. Also, for the case when the denominator is the constant $c \neq 1$, the divisions of the numerator and denominator by c are also carried out.

Therefore, the simplification process for rational functions can be expressed through the following activity diagram in UML, represented on figure 4.1.

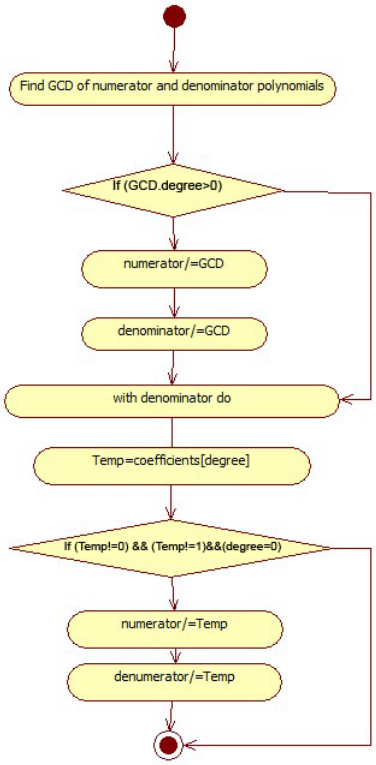

FIG. 4.1: Activity diagram of the simplification function for rational functions.

In papers [6] and [9] emerged the need to represent a rational matrix M of the unknown x to the form:

$$M = \frac{1}{p(x)} M_1, \tag{4.1}$$

where $p(x)$ is a polynomial and M_1 is the polynomial matrix of the unknown x. Specifically, Algorithm 3.1 from [9] and Algorithm 3.3 from [6] require these representations in order to apply the algorithm for the evaluation of the inverse matrix of the polynomial matrix M_1 (therefore, $M^{-1} = p(x) \cdot M_1^{-1}$).

An obvious way of generating the representation (4.1) is to evaluate the Least Common Multiple (LCM) of all denominator polynomials occurring in the matrix M and to set $p(x) = LCM$. Then, a simple multiplication of each numerator polynomial by the LCM divided by the appropriate denominator, is done. The resulting polynomial matrix M_1 consists of these newly generated polynomials.

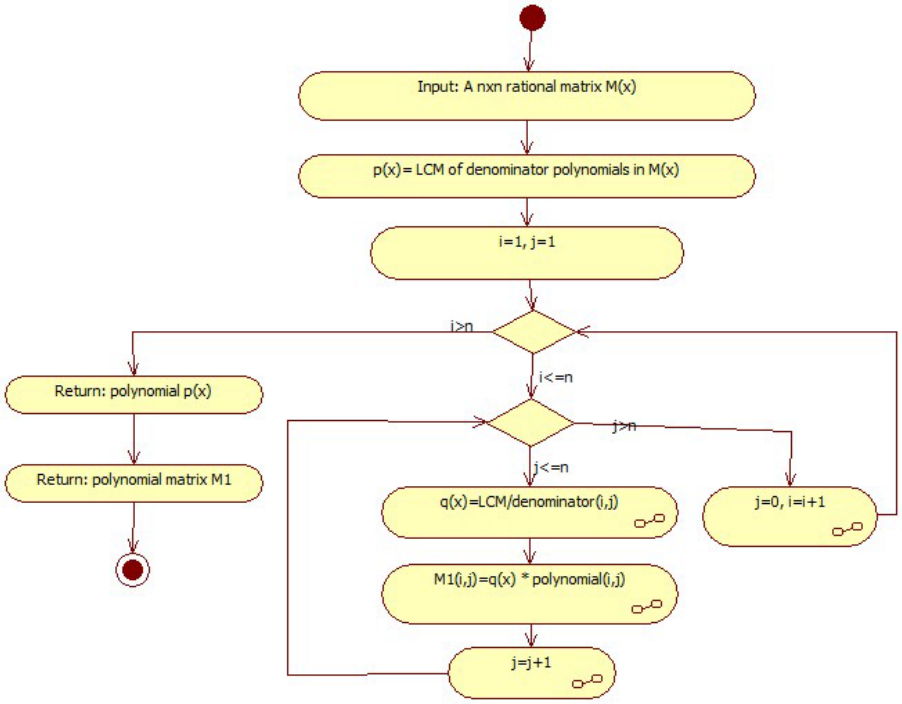

FIG. 4.2: Activity diagram for representation (4.1) of a rational matrix.

5. Numerical examples

The following example demonstrates the crucial element of rational matrix simplification used in [6].

Example 5.1. Consider the full-rank LDL^* decomposition of the matrix:

$$S_3 = \begin{bmatrix} 1+x & x & 1+x \\ x & -1+x & x \\ 1+x & x & 1+x \end{bmatrix}.$$

The matrices L and D are therefore given by

$$L = \begin{bmatrix} 1 & 0 \\ \frac{x}{1+x} & 1 \\ 1 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 1+x & 0 \\ 0 & -1+x-\frac{x^2}{1+x} \end{bmatrix}.$$

The simplification of the rational matrix D involves the crucial simplification of the rational function

$$D_{2,2}(x) = -1+x-\frac{x^2}{1+x}.$$

This function should be represented as a quotient of two polynomials. Therefore, the polynomial $-1+x$ is replaced by $\frac{-1+x}{1}$. The simplification process then applies the reduction of a rational function

$$D_{2,2}(x) = \frac{(-1+x)(1+x)}{1+x} - \frac{x^2}{1+x}.$$

So, the numerator and denominator polynomials are divided by their common LCM and the resulting function is $D_{2,2}(x) = -\frac{1}{1+x}$.

Example 5.2. Consider the following polynomial matrices from [7]:

$$A = \begin{bmatrix} -4x^2-3 & 2-7x & 4 \\ -9x & 3x^2-3 & -5 \\ 9x^2-2x & 9x^2 & -5 \\ -4x^2-3 & 2-7x & 4 \end{bmatrix}, \quad W = \begin{bmatrix} 3 & 7x & 4 & 5 \\ -9x & 3x^2-3 & 5 & x+5 \\ -6 & -14x & -8 & -10 \end{bmatrix}.$$

The matrix W was chosen randomly, but with appropriate dimensions. Algorithm 2.1 from [7] produced the full-rank QDR factorization of W, where the rational matrices Q and R are given as:

$$Q = \begin{bmatrix} 3 & \frac{9x(8x^2-1)}{9x^2+5} \\ -9x & \frac{15(8x^2-1)}{9x^2+5} \\ -6 & -\frac{18x(8x^2-1)}{9x^2+5} \end{bmatrix}, \quad R = \begin{bmatrix} 9\left(9x^2+5\right) & 3x\left(44-9x^2\right) & 60-45x & 75-9x(x+5) \\ 0 & \frac{45\left(1-8x^2\right)^2}{9x^2+5} & \frac{15(12x+5)\left(8x^2-1\right)}{9x^2+5} & \frac{15(16x+5)\left(8x^2-1\right)}{9x^2+5} \end{bmatrix},$$

and therefore, the matrix product RAQ is as follows:

$$RAQ = \begin{bmatrix} 9\left(-240+175x+716x^2-1164x^3+417x^4+81x^6\right) \\ -\frac{45\left(-35-96x+530x^2+1040x^3-2324x^4-2392x^5+2592x^6+1728x^7\right)}{5+9x^2} \\ -\frac{45\left(80-586x-58x^2+4178x^3-4755x^4+4044x^5+792x^6+288x^7\right)}{5+9x^2} \\ -\frac{45\left(-1+8x^2\right)^2\left(-95+120x+857x^2-1263x^3+156x^4\right)}{\left(5+9x^2\right)^2} \end{bmatrix}.$$

This matrix is derived after the simplification process of each single entry represented by a rational function. To represent the matrix RAQ to the form of quotient of a polynomial matrix and a polynomial, one should evaluate the LCM of the denominators in RAQ. Therefore:

$$RAQ = \frac{1}{25+90x^2+81x^4} \cdot \begin{bmatrix} -54000+39375x-33300x^2-120150x^3+498825x^4-815265x^5+877959x^6-848556x^7+369603x^8+59049x^{10} \\ 7875+21600x-105075x^2-195120x^3+308250x^4+117000x^5+358020x^6+579960x^7-1049760x^8-699840x^9 \\ -18000+131850x-19350x^2-702720x^3+1093365x^4-2601990x^5+1747575x^6-1702620x^7-320760x^8-116640x^9 \\ 4275-5400x-106965x^2+143235x^3+883620x^4-1254960x^5-2355840x^6+3637440x^7-449280x^8 \end{bmatrix}.$$

6. Conclusion

Use-Case diagram for algorithms involving matrix simplifications and some activity diagrams for simplifying matrix entries and different representation were provided via UML. Algorithms involving different matrix simplifications and representations were derived in previous papers. Here we explained these methods with more details and used UML to represent them. We illustrated this simplification techniques via several numerical examples.

7. Appendix

A. C++ implementation case of the class RFunction.

```
class RFunction
{
public:
RFunction ( const Polinom& = Polinom(0), const Polinom& = Polinom(0));
    RFunction ( const RFunction& );
    RFunction& operator = ( const RFunction& );
    Polinom Numerator() const;
    Polinom Denominator() const;
    void Simplify();
friend ostream &operator << (ostream &cout, const RFunction &);
friend istream &operator >> (istream &cin, RFunction &);
    static Polinom NZD (Polinom&, Polinom&);

private:
Polinom numerator;
Polinom denominator;
};

void RFunction::Simplify()
{
Polinom NZDPoly = NZD(numerator, denominator);
if (NZDPoly.Stepen() > 0)
    {
numerator = numerator / NZDPoly;
        denominator = denominator / NZDPoly;
    }
RNumber tmp = this -> denominator.Coefficients() [this -> denominator.Stepen()];
if((tmp!=RNumber(1,1))&&(tmp!=RNumber(0,1))&&(this->denominator.Stepen()==0))
{ numerator.Divide(tmp);
denominator.Divide(tmp);
}
}

Polinom RFunction::NZD ( Polinom& p, Polinom& q )
{
Polinom pp = Polinom (p);
Polinom qq = Polinom (q);
if (p.Stepen() < q.Stepen())
{ pp = Polinom (q);
qq = Polinom (p);
```

```
}
if (pp.isZero () && qq.isZero ()) return Polinom (0, 'x', 1);
while (qq.isZero() == false)
{ Polinom tmp = qq;
qq = pp % qq;
pp = tmp;
}
pp.SetMonic();
return pp;
}
```

B. Graphical interface of the application for simplifying rational functions is depicted on the following figure 7.1.

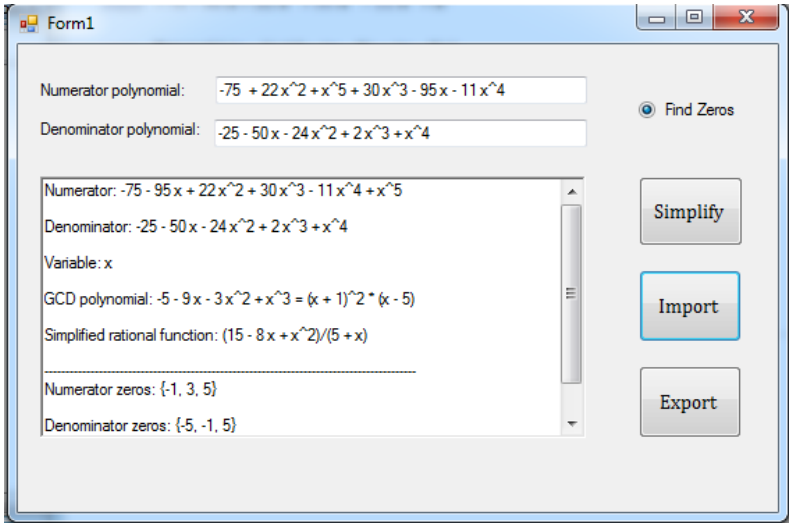

FIG. 7.1: Application's graphical interface.

REFERENCES

1. G. BOOCH, J. RUMBAUGH, I. JACOBSON: *The Unified Modeling Language User Guide*, Addison Wesley, 1999.
2. G. H. GOLUB, C. F. VAN LOAN: *Matrix Computations, Third edition*, The Johns Hopkins University Press, Baltimore, 1996.
3. A.N. MALYSHEV: *"Matrix equations: Factorization of matrix polynomials" M. Hazewinkel (ed.)*, Handbook of Algebra I, Elsevier (1995), 79-116.
4. OBJECT MANAGEMENT GROUP, HTTP:/WWW.OMG.ORG: *OMG Uniffied Modeling Language specification*, March 2003, version 1.5.
5. L. RODMAN: *"Matrix functions" M. Hazewinkel (ed.) , Handbook of Algebra* , I , Elsevier (1995), 117-154.

6. I.P. Stanimirović, M.B. Tasić, A.M. Ilić: *Full-rank LDL* Decomposition and Generalized Inverses of Polynomial Hermitian Matrices*, submitted to Information and Computation.
7. P. Stanimirović, D. Pappas, V. Katsikis, I. Stanimirović: *Symbolic computation of $A^{(2)}_{T,S}$-inverses using QDR factorization*, Linear Algebra Appl. **437** (2012), 1317-1331.
8. M.B. Tasić, I.P. Stanimirović: *Implementation of partitioning method*, Facta Universitatis (Niš) Ser. Math. Inform. **25** (2010), 25–33.
9. M.B. Tasić, I.P. Stanimirović: *Symbolic computation of Moore-Penrose inverse using the LDL* decomposition of the polynomial matrix*, Filomat (2012), accepted.
10. M.B. Tasić, P.S. Stanimirović, M.D. Petković: *Symbolic computation of weighted Moore-Penrose inverse using partitioning method*, Appl. Math. Comput. **189** (2007), 615–640.
11. F. Winkler: *Polynomial Algorithms in Computer Algebra*, Springer, 1996.
12. S. Wolfram: *The Mathematica Book, 4th ed.*, Wolfram Media/Cambridge University Press, 1999.

Milan B. Tasić
Faculty of Sciences and Mathematics,
Department of Computer Sciences,
P. O. Box 224, Višegradska 33,
18000 Niš, Serbia
milan12t@ptt.rs

Ivan P. Stanimirović
Faculty of Sciences and Mathematics,
Department of Computer Sciences,
P. O. Box 224, Višegradska 33,
18000 Niš, Serbia
ivan.stanimirovic@gmail.com