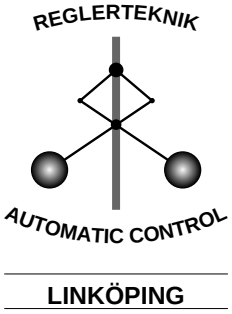
Matrix Algebra Package for *Mathematica*

Johan Gunnarsson, Tomas McKelvey and Anders Helmersson

Department of Electrical Engineering
Linkping University, S-581 83 Linkping, Sweden
WWW: `http://www.control.isy.liu.se`
Email: {`johan,tomas,andersh`}`@isy.liu.se`

March 1999



Report no.: LiTH-ISY-R-2133

Technical reports from the Automatic Control group in Linkping are available by anonymous ftp at the address `ftp.control.isy.liu.se`. This report is contained in the pdf-file `2133.pdf`.

Matrix Algebra Package for *Mathematica*

Johan Gunnarsson
Tomas McKelvey
Anders Helmersson
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden
www: `http://www.control.isy.liu.se`
email: `johan@isy.liu.se`,
`tomas@isy.liu.se`, `andersh@isy.liu.se`

May 4, 1998

1 Introduction

Computer based tools play an important role in most scientific work of today. The modern high performing desk top computers have made it possible to analyze and simulate large scale systems which ten years ago only could be handled by supercomputing facilities. Not only the hardware has improved dramatically. The software of today is keeping up the pace and is providing more and more functionality. Computer based tools for symbolic mathematics is one area which during the last 15 years have developed from basic research into commercial packages such as Maple and *Mathematica*. These tools have made it possible to manipulate and visualize complex mathematical expressions and relieving the scientist or engineer from simple but error prune tasks such as differentiation and integration. The tools of today handles with ease scalar mathematical objects. Functions from basic symbolic linear algebra is also supported in a limited way. Each element of a symbolic matrix needs to be specified. Hence, the existing software packages do not recognize a matrix as an object.

Matrix expression manipulations plays a key role in the development of systems and control theory. Therefore it is natural to ask if it is possible to develop computer based tools which would facilitate such manipulations. It is easy to see uses both for educational purposes as well as in the development of new theory. We can here point to two practical applications. Firstly simple bookkeeping. When manually manipulating matrix expressions it is highly likely that one make mistakes. A computer based matrix expression yellow pad could make life much easier. Also for documentational reasons it is important to be able to automatically export the expressions to some documentation system, e.g. LaTeX. A second application to mention is the task of verifying that certain calculations are correct, e.g. to check if two matrix expressions are equal.

In this paper we describe a small matrix algebra prototype which is implemented in *Mathematica* [2].

2 Approach

Algorithms for manipulating algebraic expressions, such as expression of linear algebra, need the functionality provided by a computer algebra system. For this project *Mathematica* has been chosen for the following reasons.

2.1 Programming Environment

Mathematica provide three different types of programming for the developer: *procedural*, *functional* and *rule based* programming.

By procedural programming we refer to the most common programming style which includes languages like Pascal and C. Common languages keywords here are, e.g., `For`, `Do`, `If`, `Then`, `Else` etc.

The functional programming style can be compared with the language Lisp. Examples of functional programming in *Mathematica* are the keywords `Map` and `Apply` which apply functions to expressions or parts of expressions.

The rule based programming style in *Mathematica* can be compared with a functional language like ML, and is the main reason why *Mathematica* is a powerful programming environment for mathematical manipulations. Mathematical derivation rules can be formulated in close correspondence with formulations in mathematical textbooks. The following *Mathematica* code contains a few rules for how to compute transpose of matrix expressions. The rule

```
Transpose[m_?MatrixSymbolQ] /;
     SymmetricQ[m] := m
```

specifies that `Transpose` of a symbol m representing a matrix is will be equal m if m is symmetric. The corresponding rule for the antisymmetric case is:

```
Transpose[m_?MatrixSymbolQ] /;
     AntisymmetricQ[m] := -m
```

The rule for transpose of a block matrix can be written as

```
Transpose[m_?BlockMatrixQ] :=
     Thread[Map[Transpose,m,{2}]]
```

where the functional operators `Map` and `Thread` are used to perform a transpose of the matrix elements inside the block matrix and on the top level respectively.

2.2 Mathematical Layout

Mathematica has mathematical layout of expressions, which make it possible to have similar layout standards as in text books for both input and output expressions. One major advantage of the layout system is the flexibility and the possibility to design new layout constructions, by using similar markup language facilities as in LaTeX and HTML. The ready made tools for exporting exporting mathematical expression and typeset documents to LaTeX and HTML is also important.

2.3 User Interface

Mathematica provide a base technology for programmable documents called *notebooks*. This technique is used to produce, e.g., palettes and forms that will make it possible to simplify and enhance the tool form the user perspective.

3 Simplification

In mathematical manipulation algorithms, simplification is one key issue. For instance we can show that

$$A(I + BA)^{-1} - (I + AB)^{-1}A = 0 \qquad (1)$$

where A and B are matrices of compatible sizes. In this case it is obvious that the zero in the right-hand side is simpler than the left-hand expression. An introduction to simplification algorithms for matrices is given in [1].

When performing matrix manipulations by hand we usually apply a number of known rules or patterns, such as the one above, sometimes referred to as the push-through rule and the matrix inversion lemma:

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1} \qquad (2)$$

The problem of deciding if two expressions are equal or, equivalently, if an expression is equal to zero, is referred to as the decidability problem. For rational expressions involving scalars this can be decided by putting the terms together with a common denominator. When matrices are involved, the problem becomes much more difficult due to the fact that matrices in a product do no commute. For instance finding the greates common divisor becomes non-trivial.

A more general approach to the simplification problem is to find a canonical form, which is a unique, representation for each expression. The canonical form should be as "simple" as possible. For instance the expression in (1) should have 0 as its canonical form.

It is not known to the authors if there exists a canonical form for each expression, neither if a matrix expression is decidable.

As one step towards mathematical derivation, an expression metrics could be used. With this we mean a measure of the "complexity" of an expression. The derivation could be be directed in order to reduce the complexity.

As an alternative to mathematical derivation, we may instead use pattern matching. With this we mean that certain patterns are recognized, such as the matrix inversion lemma (2). In order to direct the simplification it may be beneficial to involve some user interaction.

4 Implementation Issues

4.1 Definitions

Mathematica supports manipulations of matrices instantiated matrices, i.e., the matrix must be created with elements of numbers or scalar symbolic expressions. We have to add mechanisms for making symbols like A and B to act like matrices. If we evaluate

$$\texttt{Inverse}[\begin{pmatrix} A & B \\ C & D \end{pmatrix}] \tag{3}$$

we get the result

$$\begin{pmatrix} \frac{D}{-(B\,C)+A\,D} & -\frac{B}{-(B\,C)+A\,D} \\ -\frac{C}{-(B\,C)+A\,D} & \frac{A}{-(B\,C)+A\,D} \end{pmatrix} \tag{4}$$

since the symbols A, B, C, and D are treated as scalars.

The solution we have chosen is to have a declaration of the matrix symbols. By

$$\texttt{DefineMatrix}[A, B] \tag{5}$$

the symbols A and B are set to matrices. This will make, e.g., the behavior of `Transpose` different

$$\texttt{Transpose}[\begin{pmatrix} A & B \\ C & D \end{pmatrix}] \tag{6}$$

which returns

$$\begin{pmatrix} A^T & C \\ B^T & D \end{pmatrix} \tag{7}$$

since C and D are still treated as scalars. By this the matrix multiplication

$$\texttt{B.D.C.A} \tag{8}$$

will return

$$CD\ B.A \tag{9}$$

where scalars in the matrix product are commuted to the left. The *Mathematica* functions `Dot`, `Transpose`, and `Inverse` have been redefined to make them test if the input expression is a block matrix. If so the these functions stays unevaluated.

In addition to being able to declare matrix symbols, it must be possible to set matrix properties to these symbols. `SetMatrixProperty` is used for this, e.g.,

$$\texttt{SetMatrixProperty}[A, \{\texttt{Symmetric}\}] \tag{10}$$

means that A will be always treated as a symmetric matrix symbol. Other properties supported are: antisymmetric, diagonal, upper triangular, lower triangular, hermitian, etc.

The matrix symbol declarations and the assignment of matrix properties are implemented using predicates or test functions. For matrix symbol declaration, the predicate `MatrixSymbolQ`$[s]$ returns true if and only if the symbol s is a declared matrix symbol. The predicate `SymmetricQ` is used to test the symmetric property in the same way. The predicates are an important mechanism in *Mathematica* to be able to write simple derivation rules for the mathematical manipulations.

4.2 Notation

Standard *Mathematica* commands are usually named to complete words that describe the command. In the same way we have the matrix functions like `Transpose`, `Inverse` etc. for which there is a standard mathematical notation. Notations can be designed in *Mathematica* such that we write

$$A^T \quad A^{-1} \quad A.B \tag{11}$$

instead of using `Transpose`, `Inverse`, and `Dot`. At the same time as the notation can be designed to give nice layouts of computational results, it is important that the notation can be parsed correctly as inputs without any ambiguities. This means, e.g., that the dot (".") notation of matrix product should be kept for both outputs and inputs.

4.3 Computations

The concept of rules is used in all *Mathematica* computations. As stated earlier the there are simple rules for the computation of, e.g., transpose of expressions, as well as rules for more complex matrix simplifications. Typically the user would expect simple rules to be evaluated directly, i.e., the transpose should evaluated on expression without using any extra commands that explicitely starts to compute the transpose. This will also make it possible to introduce a standard for the form of a matrix manipulation result due to the automatic derivations that are always performed. On the other hand, rules for finding and rewriting expressions that fit to the matrix inversion lemma (2) should not be done automatically since that would slow done all computations drastically and the user would not have control over when these rules are used or not.

For certain rules, it is hard to decide if the rule should be automatically computed or if it should be manually applied by the user. This indicates that it might useful to change the computational mode of a rule and that the user can decide interactively if the rule should computed automatically or not.

5 Simple Example

A simple session of using the package follows. The symbols `ii` and $\mathbb{I}$ are used as short forms of the identity matrix.

```
In[1]:= Needs[MatrixAlgebra‘]

In[2]:= DefineMatrix[A, B]

In[3]:= SetMatrixProperty[B, {Antisymmetric}]


In[4]:= U  = (ii — B) . (ii + B)^-1

Out[4]= (-B + ℐ).((B + ℐ)^-1)

In[5]:= UnitaryQ[U]

Out[5]= True

In[6]:= MatrixSimplify[U^T — U^-1]

Out[6]= 0
```

References

[1] J. W. Helton, M. Stankus, and J. J. Wavrik. Computer simplification of formulas in linear system theory. *IEEE Transactions on Automatic Control*, 43:302–314, March 1998.

[2] S. Wolfram. *The Mathematica Book*. Wolfram Media, third edition, 1996. ISBN 0-9650532-02.