A (hopefully) friendly introduction to the complexity of polynomial matrix computations

Chief among the mathematical functions that symbolic computation aims to handle are those one can fully describe by *integer matrices*. Integer matrices are merely tables of whole numbers. The simplest example is the function which maps variable x to, say, 3 times x: here the associated matrix has a single entry, equal to number 3. On the other hand, real life applications such as cryptography often yield integer matrices with thousands of rows and columns and thus millions of entries. As arrays of numbers, matrices are particularly easy to store in a computer; as representations of the mathematical functions used for modelling, they encode the intrinsic properties of the underlying problem. Hence, among all the scientific problems that are to be solved by computers, many of them eventually reduce to matrix formulation.

Solving such problems requires that computers can combine matrices arithmetically, in the same way as solving simple algebraic equations such as 2x+3=4 is governed by combinations of whole numbers. So perhaps the easiest way to think about matrices in computer science is as a generalization of numbers: just like with numbers, what we can do with two matrices of the same size—that is, the same number of rows and columns—includes *addition*, *subtraction*, *multiplication* and, sometimes, *division*. The result of any of these arithmetic operations is another matrix with the same size. Having more than one row and one column further allows for some matrix operations that do not exist for numbers: given a matrix, one may transform it into mathematically equivalent matrices whose all non-diagonal entries are equal to zero or whose entries are, in a sense, "as small as possible". We usually call these simplifying operations *diagonalization* and *reduction*, respectively. Other operations return some important mathematical characteristics of the matrix that are not immediately visible, like its volume (*determinant*) or a particular relation between its rows or its columns (*linear system solving*). *Matrix computations* usually refer to all to these operations when performed by a computer. It is worth noting at this stage that instead of numbers, matrices may often contain functions of one variable called polynomials: polynomials look very similar to whole numbers but are, perhaps surprisingly, easier to handle mathematically. In this case we say *polynomial matrices* and *polynomial matrix computations*.

Now assume that you want your computer to perform one of these operations exactly for some large matrices. Before starting, a most legitimate and fundamental question is then "How hard is this computation?" Most of today's computers are "binary computers" and every computation ultimately reduces to a sequence of logical operations on zeros and ones called *bit operations*. So, by hardness or *complexity of a computation*, we mean an idea of the number of bit operations associated with this particular computation as a function of the input size: when doubling the size of the input matrix, will producing the exact result take roughly twice as long as before (*linear complexity*)? Or should this be expected to be four times slower (*quadratic complexity*) or eight times slower (*cubic complexity*) or even more? (We recall that 4 = 2 times 2 = 2 squared

and that 8 = 2 times 2 times 2 = 2 cubed.)

In a sense, complexity indicates the intrinsic computational hardness of matrix operations, independently of solution methods (or algorithms) and software and hardware. For example, what is the complexity of adding two matrices of size n? Matrix addition is defined entry-wise: for each row and column, we add the two entries lying on the same row and column. Since the result is a matrix of size n whose all entries may differ from the input values, every addition algorithm requires at least n squared operations on whole numbers and the complexity of matrix addition is thus *lower bounded* by n squared. On the other hand, the definition of matrix addition yields an addition algorithm that costs no more than n squared additions of whole numbers and the complexity is *upper bounded* by n squared. The two bounds coincide and matrix addition has quadratic complexity.

Of course the same holds for matrix subtraction. But for every other operation, even the fastest known algorithms still yield upper bounds much different from output size-based lower bounds. In particular, evaluating the complexity of matrix multiplication alone seems to be a formidable task. Unlike addition, the classic definition of matrix multiplication yields an algorithm that costs n cubed operations on numbers of the same order of magnitude as the input numbers. This upper bound was improved for the first time in 1969 when Strassen discovered a product formula whose cost is about n to the power w with w strictly between 2 and 3. Since then, exponent w has been decreased to less than 2.39 but no method is known where w = 2. However, right after Strassen's breakthrough, multiplication became central to the understanding of the complexity of all other matrix operations: during the thirty years that followed, algorithms for performing matrix operations have been designed that rely only on matrix products (and on other a priori cheaper operations such as addition and subtraction). The reason for this approach is twofold: we get tighter upper bounds for the complexity of all matrix computations every time w is decreased; it may also give a way to relate complexities without having to know their exact value. For example, if the number of matrix products is "small" and if the magnitude of the size and of the entries of the matrices involved have the same order as those of the input then such computations are *not essentially harder* than matrix multiplication. Here by "small" we mean a function of the input matrix size n that grows extremely slowly as n increases. Such a function is called a polylogarithm.

Unfortunately, until last year, incorporating matrix multiplication typically meant too many products. Thus it was not known whether some matrix computations are reduceable to matrix multiplication. At ISSAC 2002, however, Storjohann showed that, for polynomial matrices, diagonalization, determinant computation and linear system solving are not essentially harder than matrix multiplication. He got his result by designing clever algorithms that return a provably correct answer while keeping the number of matrix products small enough. This breakthrough has opened up a new path in the study of how the complexities of matrix computations relate to each other. Are there other matrix operations that can be performed within roughly the same amount of time

as matrix multiplication? Conversely, is matrix multiplication not essentially harder than some other operations?

This year, at ISSAC 2003, we shall give answers in both directions for polynomial matrices. First, we prove that reduction is not essentially harder than matrix multiplication by combining Storjohann's technique with Villard's earlier work on this particular operation. Then we show that the converse of one of Storjohann's results holds true: in fact, matrix multiplication is not essentially harder than computing the determinant. (This second result might be a little surprising at first sight since, as a volume, the determinant is a single number.)

Yet we are still far from a global view of the complexity of polynomial matrix computations, and even less is known in the more difficult case of integer matrices. In particular, are there operations other than computing the determinant that essentially have the same complexity as matrix multiplication? Are some of them—such as linear system solving—strictly easier? We don't know. On the other hand we do know of operations being much harder, such as dividing a matrix by another matrix. The result of matrix division is a matrix of the same size whose entries are in general not whole numbers but fractions like 2/5 or -7/3. Unfortunately the upper and lower parts of each of these n squared fractions usually have n times as many digits as the entries of the two matrix operands. The complexity of matrix division is thus lower bounded by n cubed which, as seen before, is itself lower bounded by the complexity of matrix multiplication. Hence another open problem is to design a division algorithm whose cost matches this cubic lower bound.

It is important to note that most of the above complexity results have been obtained via the design and analysis of concrete algorithms you can implement quite easily into any computer algebra system. How to make these algorithms practically fast in order to reflect the theory is thus another challenge for researchers in this field.

A. Storjohann. *High-order lifting*, proceedings of ISSAC 2002.
V. Strassen. *Gaussian elimination is not optimal*, Numer. Math. 13:4 (1969).
G. Villard. *Computing Popov and Hermite forms of polynomial matrices*, proceedings of ISSAC 1996.