Computer algebra systems, formal proofs and interactive theorem proving

Anders Mörtberg

Mathematical Conversation – Institute for Advanced Study

February 5, 2016

The Misfortunes of a Trio of Mathematicians Using Computer Algebra Systems. Can We Trust in Them?

Antonio J. Durán, Mario Pérez, and Juan L. Varona

Introduction

Nowadays, mathematicians often use a computer algebra system as an aid in their mathematical research; they do the thinking and leave the tedious calculations to the computer. Everybody "knows" that computers perform this work better than people. But, of course, we must trust in the results derived via these powerful computer algebra systems. First of all, let us clarify that this paper is not, in any way, a comparison between different computer algebra systems, but a sample of the current state of the art of what mathematicians can expect when they use this kind of software. Although our example deals with a concrete system, we are sure that similar situations may occur with other programs.

We are currently using Mathematica to find examples and counterexamples of some mathematical results that we are working out, with the aim of finding the correct hypotheses and eventually constructing a mathematical proof. Our goal was to improve some results of Karlin and Szegő [4] related to orthogonal polynomials on the real line. The details are not important; this is just an example of the use of a computer algebra system by a typical research mathematician, but let us explain it briefly. It is not necessary to completely understand the mathematics, just to realize that it is typical mathematical research using computer algebra as a tool.

Our starting point is a discrete positive measure on the real line, $\mu = \sum_{n\geq 0} M_n \delta_{a_n}$ (where δ_a denotes the Dirac delta at a, and $a_n < a_{n+1}$) having a sequence of orthogonal polynomials $\{P_n\}_{n\geq 0}$ (where P_n has degree n and positive leading coefficient). Karlin and Szegő considered in 1961 (see [4]) the $l \times l$ Casorati determinants

$$(1)\qquad \det\begin{pmatrix} P_n(a_k) & P_n(a_{k+1}) & \dots & P_n(a_{k+l-1}) \\ P_{n+1}(a_k) & P_{n+1}(a_{k+1}) & \dots & P_{n+1}(a_{k+l-1}) \\ \vdots & \vdots & \vdots & \vdots \\ P_{n+l-1}(a_k) & P_{n+l-1}(a_{k+1}) & \dots & P_{n+l-1}(a_{k+l-1}) \end{pmatrix}, \quad n,k \geq 0.$$

They proved that, under the assumption that l is even, these determinants are positive for all nonnegative integers n, k. Notice that the set of indices $\{n, n+1, \dots, n+l-1\}$ for the polynomials P_n consists of consecutive nonnegative integers. We are working out an extension of this remarkable result for more general sets of indices F than those formed by consecutive nonnegative integers. We have some conjectures that we want to prove or disprove.

We have not been able to prove our conjectures yet, and, as far as we can see, this task seems to be rather difficult. On the other hand, just in case our conjectures are wrong, we have been trying to find counterexamples with the help of our computer algebra system. Eventually we hope these experiments can shed some light on the problem as well.

We have then proceeded to construct orthogonal polynomials with respect to discrete positive

Antonio J. Durán is professor of mathematics at Universidad de Sevilla (Spain). His email address is duran@us.es.

Mario Pérez is professor of mathematics at Universidad de Zaragoza (Spain). His email address is mperez@unizar.es.

Juan L. Varona is professor of mathematics and computation at Universidad de La Rioja (Spain). His email address is jvarona@unirioja.es.

This work was partially supported by grants MTM2012-36732-C03-02, MTM2012-36732-C03-03 (Ministerio de Economía y Competitividad), FQM-262, FQM-4643, FQM-7276 (Junta de Andalucía) and Feder Funds (European Union).

DOI: http://dx.doi.org/10.1090/noti1173

```
In[236]:= basicMat =
 ( -32   69   89  -60  -83  -22  -14  -58   85   56  -65  -30  -86   -9 )
 (   6   99   11   57   47  -42  -48  -65   25   50  -70   -3  -90   31 )
 (  78   38   12   64  -67   -4  -52  -65   19   71   38  -17   51   -3 )
 ( -93   30   89   22   13   48  -73   93   11  -97  -49   61  -25   -4 )
 (  54  -22   54  -53  -52   64   19    1   81  -72  -11   50    0  -81 )
 (  65  -58    3   57   19   77   76  -57  -80   22   93  -85   67   58 )
 (  29  -58   47   87    3   -6  -81    5   98   86  -98   51  -62  -66 )
 (  93  -77   16  -64   48   84   97   75   89   63   34  -98  -94   19 ) ;
 (  45  -99    3  -57   32   60   74    4   69   98  -40  -69  -28  -26 )
 ( -13   51  -99   -2   48   71  -81  -32   78   27  -28  -22   22   94 )
 (  11   72  -74   86   79  -58  -89   80   70   55  -49   51  -42   66 )
 ( -72   53   49  -46   17  -22  -48  -40  -28  -85   88  -30   74   32 )
 ( -92  -22  -90   67  -25  -28  -91   -8   32  -41   10    6   85   21 )
 (  47  -73  -30  -60   99    9  -86  -70   84   55   19   69   11  -84 )

In[235]:= smallMat =

 (  528   853  -547  -323   393  -916   -11  -976   279  -665   906  -277   103  -485 )
 (  878   910  -306  -260   575  -765   -32    94   254   276  -156   625    -8  -566 )
 ( -357   451  -475   327   -84   237   647   505  -137   363  -808   332   222  -998 )
 (  -76    26  -778   505   942  -561  -350   698  -532  -507   -78  -758   346  -545 )
 ( -358    18  -229  -880  -955  -346   550  -958   867  -541  -962   646   932   168 )
 (  192   233   620   955  -877   281   357  -226  -820   513  -882   536  -237   877 )
 ( -234   -71  -831   880  -135  -249  -427   737   664   298  -552    -1  -712  -691 )
 (   80   748   684   332   730  -111  -643   102  -242   -82   -28   585   207  -986 ) ;
 (  967     1  -494   633   891  -907  -586   129   688   150  -501  -298   704   -68 )
 (  406  -944  -533  -827   615   907  -443  -350   700  -878   706     1   800   120 )
 (   33  -328  -543   583  -443  -635   904  -745  -398  -110   751   660   474   255 )
 ( -537  -311   829    28   175   182  -930   258  -808  -399   -43   -68  -553   421 )
 ( -373  -447  -252  -619  -418   764   994  -543   -37  -845    30  -704   147  -534 )
 (  638   -33   932  -335   -75  -676  -934   239   210   665   414  -803   564  -805 )

In[234]:= powersMat = DiagonalMatrix[{10 ^123, 10 ^152, 10 ^185, 10 ^220, 10 ^397, 10 ^449,
    10 ^503, 10 ^563, 10 ^979, 10 ^1059, 10 ^1143, 10 ^1229, 10 ^1319, 10 ^1412}];

In[232]:= bigMat = basicMat.powersMat + smallMat;

In[227]:= a = Det[bigMatrix];

In[228]:= b = Det[bigMatrix];

In[237]:= a == b

Out[238]= False
```

The Misfortunes of a Trio of Mathematicians Using Computer Algebra Systems. Can We Trust in Them?

"In attempting to isolate the computational problem, we finally realized that, in some circumstances, Mathematica (version 9.0.1 at that time) makes some strange mistakes when computing determinants whose entries are large integers. Errors do not always occur – only in some cases. Even worse, given the same matrix, the determinant function can give different values!"

The Misfortunes of a Trio of Mathematicians Using Computer Algebra Systems. Can We Trust in Them?

"In attempting to isolate the computational problem, we finally realized that, in some circumstances, Mathematica (version 9.0.1 at that time) makes some strange mistakes when computing determinants whose entries are large integers. Errors do not always occur – only in some cases. Even worse, given the same matrix, the determinant function can give different values!"

"This resembles the well-known Pentium division bug discovered by Thomas Nicely in 1994, which only affected certain kinds of numbers. But it seems Mathematica is a black box even darker than the internals of a microprocessor, so it is difficult to try to understand what kinds of numbers are affected by the Mathematica bug that we are describing."

Bugs in computer algebra systems (CAS)

- The bug was reported October 7, 2013 (Mathematica 8), and was still there in June 2014 (Mathematica 9)...
- Yesterday I tried on my office machine (Mathematica 10) and the particular determinant bug seems to have been fixed – but I was still getting incorrect counterexamples to the conjecture from the paper...

- Bugs in CAS are serious, not only because mathematicians can end up "proving" something false or finding incorrect counterexamples, but because CAS are used a lot also in industry (cars, aviation, military, medicine...)[1]

[1] `https://en.wikipedia.org/wiki/List_of_software_bugs`

How to increase reliability?

The conclusion of the article is:

> *"Software bugs should not prevent us from continuing this mutually beneficial relationship [between mathematicians and computers] in the future. However, for the time being, when dealing with a problem whose answer cannot be easily verified without a computer, it is highly advisable to perform the computations with at least two computer algebra systems."*

How to increase reliability?

"We run millions and millions of tests on every version of Mathematica, trying to exercise every part of the system. And doing that is orders of magnitude more powerful at catching bugs than any kind of pure human testing." – Stephen Wolfram (2007)

How to increase reliability?

"We run millions and millions of tests on every version of Mathematica, trying to exercise every part of the system. And doing that is orders of magnitude more powerful at catching bugs than any kind of pure human testing." – Stephen Wolfram (2007)

"Program testing can be used to show the presence of bugs, but never to show their absence!" – Dijkstra (1970)

How to increase reliability?

"We run millions and millions of tests on every version of Mathematica, trying to exercise every part of the system. And doing that is orders of magnitude more powerful at catching bugs than any kind of pure human testing." – Stephen Wolfram (2007)

"Program testing can be used to show the presence of bugs, but never to show their absence!" – Dijkstra (1970)

Can we do better?

How to increase reliability?

"We run millions and millions of tests on every version of Mathematica, trying to exercise every part of the system. And doing that is orders of magnitude more powerful at catching bugs than any kind of pure human testing." – Stephen Wolfram (2007)

"Program testing can be used to show the presence of bugs, but never to show their absence!" – Dijkstra (1970)

Can we do better?

Yes! By formally verifying the correctness of the implementation using a proof assistant.

Formalizing Bareiss algorithm

- Want: Polynomial time algorithm for computing the determinant of a matrix with coefficients in any commutative ring
- Formally verified implementation in COQ – that can be used to compute determinants correctly
- COQ is both a *proof assistant* and a *programming language*. One can write both programs and their correctness proofs in the system, and COQ then checks that the proofs are correct.

Bareiss algorithm

- Erwin Bareiss: *"Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination"* (1968)
- Compute determinant of integer matrices in polynomial time
- Similar to Gaussian elimination, but some subtle differences:
  - After n "steps" a_{nn} is the determinant of the $n \times n$ upper-left submatrix
  - All divisions are **guaranteed to be exact**, ie. the computations are *fraction-free*

Bareiss algorithm

- Erwin Bareiss: *"Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination"* (1968)
- Compute determinant of integer matrices in polynomial time
- Similar to Gaussian elimination, but some subtle differences:
    - After n "steps" a_{nn} is the determinant of the $n \times n$ upper-left submatrix
    - All divisions are **guaranteed to be exact**, ie. the computations are *fraction-free*

"We [he and Halmos] share a philosophy about linear algebra: we think basis-free, we write basis-free, but when the chips are down we close the office door and compute with matrices like fury." – Irving Kaplansky

Bareiss algorithm: Example

$$\begin{pmatrix} 2 & 2 & 4 & 5 \\ 5 & 8 & 9 & 3 \\ 1 & 2 & 8 & 5 \\ 6 & 6 & 7 & 1 \end{pmatrix}$$

Bareiss algorithm: Example

$$\begin{pmatrix} 2 & 2 & 4 & 5 \\ 0 & 6 & -2 & -19 \\ 0 & 2 & 12 & 5 \\ 0 & 0 & -10 & -28 \end{pmatrix}$$

Bareiss algorithm: Example

$$\begin{pmatrix} 2 & 2 & 4 & 5 \\ 0 & 6 & -2 & -19 \\ 0 & 0 & 76 & 68 \\ 0 & 0 & -60 & -168 \end{pmatrix}$$

Bareiss algorithm: Example

$$\begin{pmatrix} 2 & 2 & 4 & 5 \\ 0 & 6 & -2 & -19 \\ 0 & 0 & 76/2 & 68/2 \\ 0 & 0 & -60/2 & -168/2 \end{pmatrix}$$

Bareiss algorithm: Example

$$\begin{pmatrix} 2 & 2 & 4 & 5 \\ 0 & 6 & -2 & -19 \\ 0 & 0 & 38 & 34 \\ 0 & 0 & -30 & -84 \end{pmatrix}$$

Bareiss algorithm: Example

$$\begin{pmatrix} 2 & 2 & 4 & 5 \\ 0 & 6 & -2 & -19 \\ 0 & 0 & 38 & 34 \\ 0 & 0 & 0 & -2172 \end{pmatrix}$$

Bareiss algorithm: Example

$$\begin{pmatrix} 2 & 2 & 4 & 5 \\ 0 & 6 & -2 & -19 \\ 0 & 0 & 38 & 34 \\ 0 & 0 & 0 & -2172/6 \end{pmatrix}$$

Bareiss algorithm: Example

$$\begin{pmatrix} 2 & 2 & 4 & 5 \\ 0 & 6 & -2 & -19 \\ 0 & 0 & 38 & 34 \\ 0 & 0 & 0 & -362 \end{pmatrix}$$

Bareiss algorithm: Example

$$\det \begin{pmatrix} 2 & 2 & 4 & 5 \\ 5 & 8 & 9 & 3 \\ 1 & 2 & 8 & 5 \\ 6 & 6 & 7 & 1 \end{pmatrix} = -362$$

Bareiss algorithm: problems

- 0 on diagonal implies division by zero?

Bareiss algorithm: problems

- 0 on diagonal implies division by zero?
    - No problem, find a nonzero pivot and reorganize

Bareiss algorithm: problems

- 0 on diagonal implies division by zero?
  - No problem, find a nonzero pivot and reorganize

- Generalize to any commutative ring?

Bareiss algorithm: problems

- 0 on diagonal implies division by zero?
  - No problem, find a nonzero pivot and reorganize
- Generalize to any commutative ring?
  - No, the ring need to have an "explicit" divisibility function:

$$a \mid b \Leftrightarrow \exists x . b = ax$$

  - Examples: Euclidean domains ($\mathbb{Z}, k[x]$) and polynomial rings over these ($\mathbb{Z}[x,y], k[x,y,z], \ldots$)

Bareiss algorithm: problems

- 0 on diagonal implies division by zero?
  - No problem, find a nonzero pivot and reorganize
- Generalize to any commutative ring?
  - No, the ring need to have an “explicit” divisibility function:

    $$a \mid b \Leftrightarrow \exists x. b = ax$$

  - Examples: Euclidean domains ($\mathbb{Z}, k[x]$) and polynomial rings over these ($\mathbb{Z}[x, y], k[x, y, z], \dots$)

But, there is a neat trick that allows us to generalize this algorithm to any commutative ring...

A nice trick

- Apply the algorithm to $xI - M$
- Compute on $R[x]$ with polynomial pseudo-division instead of division on R
- Put $x = 0$ in the result

A nice trick

- Apply the algorithm to $xI - M$
- Compute on $R[x]$ with polynomial pseudo-division instead of division on R
- Put $x = 0$ in the result

Benefits:

- More general
- No need for pivoting (we have x along the diagonal)
- Get characteristic polynomial for free
- Algorithm is the same as Bareiss'

Sasaki-Murao: *Efficient Gaussian Elimination Method for Symbolic Determinants and Linear Systems* (1982)

Correctness?

Why are all the divisions always exact?

Correctness?

Why are all the divisions always exact?

Bareiss' original paper has a complicated proof relying on quite complicated identities...

(b) to diagonal form

such that the elements of the reduced system are integers, provided the elements a_{ij} of

(5) $$A^{(0)} = A \oplus B \qquad (A \textit{ augmented by } B)$$

are integers.

A. *Reduction of A to Triangular Form.*

1. *Division-free algorithms.* The simplest reduction algorithm is given by the recurrence formulas (known as Gaussian elimination algorithm)

(6) $$a_{ij}^{(0)} = a_{ij}\,, \quad a_{ij}^{(k)} = \begin{vmatrix} a_{kk}^{(k-1)} & a_{kj}^{(k-1)} \\ a_{ik}^{(k-1)} & a_{ij}^{(k-1)} \end{vmatrix}$$

$$(k = 1, 2, \cdots, n-1)\ (i = k+1, \cdots, n)\ (j = k+1, \cdots, n, n+1, \cdots, m)\,.$$

The advantage of this formula is the absence of any division operations. The disadvantage lies in large absolute integers $a_{ij}^{(k)}$.

The next simplest division-free transformation is given by Eq. (1.6),† if the divisor is disregarded and $l = k - 2$. The result is

(7) $$a_{ij}^{(k)} = \begin{vmatrix} a_{k-1,k-1}^{(k-2)} & a_{k-1,k}^{(k-2)} & a_{k-1,j}^{(k-2)} \\ a_{k,k-1}^{(k-2)} & a_{kk}^{(k-2)} & a_{kj}^{(k-2)} \\ a_{i,k-1}^{(k-2)} & a_{ik}^{(k-2)} & a_{ij}^{(k-2)} \end{vmatrix}.$$

It is also instructive to obtain (7) directly from (6) instead of from (1.6) by applying (6) twice as follows:

$$\begin{aligned} a_{ij}^{(k)} &= \begin{vmatrix} a_{kk}^{(k-1)} & a_{kj}^{(k-1)} \\ a_{ik}^{(k-1)} & a_{ij}^{(k-1)} \end{vmatrix} \\ &= (a_{k-1,k-1}^{(k-2)}a_{kk}^{(k-2)} - a_{k-1,k}^{(k-2)}a_{k,k-1}^{(k-2)})(a_{k-1,k-1}^{(k-2)}a_{ij}^{(k-2)} - a_{k-1,j}^{(k-2)}a_{i,k-1}^{(k-2)}) \\ &\quad - (a_{k-1,k-1}^{(k-2)}a_{ik}^{(k-2)} - a_{k-1,k}^{(k-2)}a_{i,k-1}^{(k-2)})(a_{k-1,k-1}^{(k-2)}a_{kj}^{(k-2)} - a_{k-1,j}^{(k-2)}a_{k,k-1}^{(k-2)}) \\ &= (a_{k-1,k-1}^{(k-2)}a_{kk}^{(k-2)} - a_{k-1,k}^{(k-2)}a_{k,k-1}^{(k-2)})a_{k-1,k-1}^{(k-2)}a_{ij}^{(k-2)} \\ &\quad - (a_{k-1,k-1}^{(k-2)}a_{ik}^{(k-2)} - a_{k-1,k}^{(k-2)}a_{i,k-1}^{(k-2)})a_{k-1,k-1}^{(k-2)}a_{kj}^{(k-2)} \\ &\quad - a_{k-1,k-1}^{(k-2)}a_{kk}^{(k-2)}a_{k-1,j}^{(k-2)}a_{i,k-1}^{(k-2)} + [a_{k-1,k}^{(k-2)}a_{k,k-1}^{(k-2)}a_{k-1,j}^{(k-2)}a_{i,k-1}^{(k-2)}] \\ &\quad + a_{k-1,k-1}^{(k-2)}a_{ik}^{(k-2)}a_{k-1,j}^{(k-2)}a_{k,k-1}^{(k-2)} - [a_{k-1,k}^{(k-2)}a_{i,k-1}^{(k-2)}a_{k-1,j}^{(k-2)}a_{k,k-1}^{(k-2)}]\,. \end{aligned}$$

The two products indicated by brackets [ ] cancel. The remaining terms have the common factor $a_{k-1,k-1}^{(k-2)}$. It then follows easily that for (6)

$$a_{ij}^{(k)} = a_{k-1,k-1}^{(k-2)} \begin{vmatrix} a_{k-1,k-1}^{(k-2)} & a_{k-1,k}^{(k-2)} & a_{k-1,j}^{(k-2)} \\ a_{k,k-1}^{(k-2)} & a_{kk}^{(k-2)} & a_{kj}^{(k-2)} \\ a_{i,k-1}^{(k-2)} & a_{ik}^{(k-2)} & a_{ij}^{(k-2)} \end{vmatrix}.$$

Disregarding the factor $a_{k-1,k-1}^{(k-2)}$ in this equation yields (7). Therefore, the coeffi-

Correctness

When formalizing this algorithm and its correctness proof we found a simpler argument

Let a be the element to divide with and M the submatrix we are simplifying. The key observation is that the recursive call satisfies the following invariants:

- a is regular (ie. not a zero divisor)
- a^k divides all $k + 1$ minors of M
- All principal minors of M are regular

Coquand-M-Siles: *A Formal Proof of Sasaki-Murao Algorithm* (2012)

```
Lemma bareiss_recE : forall m a (M : 'M[{poly R}]_(1 + m)),
  a \is monic ->
 (forall k (f g : 'I_k.+1 -> 'I_m.+1), rdvdp (a ^+ k) (minor f g M)) ->
 (forall p (h h' : p < 1 + m), pminor h h' M \is monic) ->
  a ^+ m * (bareiss_rec a M) = \det M.
Proof.
elim=> [a M _ _ _|m ih a M am hpm hdvd] /=.
  by rewrite expr0 mul1r {2}[M]mx11_scalar det_scalar1.
have ak_monic k : a ^+ k \is monic by apply/monic_exp.
set d := M 0 0; set M' := _ - _; set M'' := map_mx _ _; simpl in M'.
have d_monic : d \is monic.
  have -> // : d = pminor (ltn0Sn _) (ltn0Sn _) M.
  have h : widen_ord (ltn0Sn m.+1) =1 (fun _ => 0)
    by move=> x; apply/ord_inj; rewrite ord1.
  by rewrite /pminor (minor_eq h h) minor1.
have dk_monic : forall k, d ^+ k \is monic by move=> k; apply/monic_exp.
have hM' : M' = a *: M''.
  pose f := fun m (i : 'I_m) (x : 'I_2) => if x == 0 then 0 else (lift 0 i).
  apply/matrixP => i j.
  rewrite !mxE big_ord1 !rshift1 [a * _]mulrC rdivpK ?(eqP am,expr1n,mulr1) //.
  move: (hdvd 1%nat (f _ i) (f _ j)).
  by rewrite !minor2 /f /= expr1 !mxE !lshift0 !rshift1.
rewrite -[M]submxK; apply/(@lregX _ d m.+1 (monic_lreg d_monic)).
have -> : ulsubmx M = d%:M by apply/rowP=> i; rewrite !mxE ord1 lshift0.
rewrite key_lemma -/M' hM' detZ mulrCA [_ * (a ^+ _ * _)]mulrCA !exprS -!mulrA.
rewrite ih // => [p h h'|k f g].
  rewrite -(@monicMl _ (a ^+ p.+1)) // -detZ -submatrix_scale -hM'.
  rewrite -(monicMl _ d_monic) key_lemma_sub monicMr //.
  by rewrite (minor_eq (lift_pred_widen_ord h) (lift_pred_widen_ord h')) hpm.
case/rdvdpP: (hdvd _ (lift_pred f) (lift_pred g)) => // x hx; apply/rdvdpP => //.
exists x; apply/(@lregX _ _ k.+1 (monic_lreg am))/(monic_lreg d_monic).
rewrite -detZ -submatrix_scale -hM' key_lemma_sub mulrA [x * _]mulrC mulrACA.
by rewrite -exprS [_ * x]mulrC -hx.
Qed.
```

Computation

"Beware of bugs in the above code; I have only proved it correct, not tried it." – Donald Knuth (1977)

Computation

"Beware of bugs in the above code; I have only proved it correct, not tried it." – Donald Knuth (1977)

We ran the computation of the big determinant in Coq and got the correct result in 46s

Mathematica 10 computes the correct results in 1.5s

We also "extracted" the Coq code to Haskell, compiled and ran the computation and got the result in 0.5s

Conclusions

- One should not blindly trust computer algebra systems! And there are alternatives to running the computations in two systems...
- Formal proofs and interactive theorem proving can help to increase the reliability, and these tools are now getting so good that it is becoming feasible to use them for real world problems!
- Vision: a computer algebra system with formal correctness proofs of critical parts? Maybe one day Coq be used as a computer algebra system?

Thank you for your attention!