J. Symbolic Computation (2008) **23**, 503–515

Teaching Computational Abstract Algebra[†]

ERICH KALTOFEN [‡]

Department of Mathematics, North Carolina State University

Raleigh, N.C. 27695-8205, U.S.A.

(Received January 1995)

We report on the contents and pedagogy of a course in abstract algebra that was taught with the aid of educational software developed within the Mathematica system. We describe the topics covered and the didactical use of the corresponding Mathematica packages, as well as draw conclusions for future such courses from the students' comments and our own experience.

1. Introduction

In the Spring semester of 1990 we undertook the educational experiment of teaching undergraduate mathematics and computer science students some fundamental notions of algebra by using computers. The course was entitled *Computational Abstract Algebra* and offered at Rensselaer Polytechnic Institute as a cross-listed mathematics and computer science course. Twelve students enrolled in it, five of whom were graduate students. The guiding principle of the experiment was to present the notions of abstract algebra in a constructive manner following the style of (van der Waerden 1940) text book. Moreover, full-fledged Mathematica (Wolfram 1991) implementations of the discussed methods were written by me. Students were asked to take the Mathematica packages and solve homework and examination problems with them. We fully intended to include both pencil-and-paper theorem proving and on-the-computer problem solving.

The topics covered in this experimental course are not typical for an abstract algebra course. There was a very limited coverage of group theory, mostly that of properties of the multiplicative group of finite fields, and an extensive coverage of integral domains. Homomorphisms were introduced in the context of modular and ring ideal arithmetic. In place of the classical material, many applications of algebraic techniques were discussed. Modular maps were applied to integer factorization by the Pollard algorithm and to polynomial factorization by the Cantor-Zassenhaus algorithm. Euclidean division of polynomials was applied to polynomial real and complex root computation by Sturm sequences. The concepts of quotient rings and reduction modulo an ideal was applied to

[†] This material is based on work supported in part by the National Science Foundation under Grants No. CCR-87-05363 (educational s upplement) and No. CCR-9319776. A major part of this work was done while the author was at the Department of Computer Science at Rensselaer Polytechnic Institute.

[‡] E-mail: kaltofen@eos.ncsu.edu

0747-7171/97/010001 + 13 sy960104 $25.00/0

both the construction of a Gröbner basis of a system of multivariate polynomial equations and to the computation of the Hermite normal form of a matrix.

In this paper the course content and software is described, and conclusions are drawn from the students' reactions and the author's own experience. All course material can be retrieved from its home directory `http://www4.ncsu.edu/~kaltofen/public.html/Courses/ComputAlgebra/Mathematica`.

2. Syllabus

The main purpose of the course was to demonstrate the power of mathematical abstraction. The large fraction of the course centered on the application of Euclid's algorithm. Here are the subjects that the author covered, with the amount of time spent on them and in the order in which they were presented. The first half of the semester consisted of an in-depth study of the properties of the integers and the abstraction to a Euclidean domain, with full mathematical proofs. The second half constituted an overview of several applications, where mathematical facts were demonstrated with Mathematica rather than rigorously proven. At Rensselaer Polytechnic Institute a semester-long course meets three times a week for 50 minutes each class. There are 14 weeks in each semester.

1 *Peano arithmetic (3 weeks):* Introduction to integer addition, multiplication, and exponentiation in an axiomatic way. Higher order binary integer operations, such as towers of exponentiations and Ackermann-like functions. Furthermore, exponential speed-up of exponentiation by repeated squaring and correctness proof of the algorithm. Demonstration of the underlying principle by the "Russian peasants' method" for integer multiplication. Primality testing by the little Fermat theorem.

2 *Euclidean domains (3 weeks):* Integer greatest common divisors (GCDs) by the Euclidean algorithm. The abstract notion of Euclidean division and Euclidean domain. Speeded integer GCDs by taking absolutely smallest remainders. The Gaussian integers as a Euclidean domain. Univariate polynomials over the rationals as a Euclidean domain. Quotient sequences and continued fractions. Unique factorization in Euclidean domains.

3 *Integer factorization (2 weeks):* Finding periods in periodic sequences by the Floyd and Brent algorithms. The Pollard-ρ algorithm for factoring integers and its analysis.

4 *Real root finding (1.5 weeks):* Sturm's theorem and the bisection method.

5 *Complex root finding (1.5 weeks):* The argument principle. The Routh-Hurwitz method for computing the winding number along a polygonal contour (see §4).

6 *Gröbner basis (1 week):* The notions of term order and reduction by a set of multivariate polynomials over the rationals. Buchberger's algorithm for computing a Gröbner basis and its Church-Rosser property.

7 *Polynomial factorization (1 week):* Finite fields and squarefree decomposition of polynomials modulo a prime integer by differentiation. The distinct-degree factorization algorithm and the Cantor-Zassenhaus algorithm. Factorization over the rational numbers by use of a sufficiently large prime modulus and by factor combination. Application to computing square roots of residues modulo a prime number.

8 *Diophantine linear equations (1 week):* The Hermite normal form of an integer matrix and integer solutions to linear systems of integer equations.

The course was intended to be substitutable for an abstract algebra course and therefore had the same prerequisite courses, namely a calculus and linear algebra course sequence. The course was taken by three undergraduate mathematics students, three undergraduate computer science students, two mathematics and one computer science students in the master of science program, and two mathematics students in the Ph.D program. Although the course covered a wide spectrum of skills, from the ability to give mathematically sound proofs to formulating algorithms in the Mathematica language, deficiencies in preparation for it were minor. The main goal of the course, namely to learn the principles and uses of mathematical abstraction, appears to have been accomplished with all students.

The selection of the topics poses a serious problem. The author knows of no textbook that covers a substantial fraction of the above topics. The main textbooks for the course were (Lipson 1981) and (Wolfram 1991). Lipson's book covers the first two topics, Peano arithmetic and the Euclidean algorithm over an abstract Euclidean domain. The integer and polynomial factorization material was taken from (Knuth 1981). Sturm's theorem was taken from (Jacobson 1974) and the Routh-Hurwitz theorem from (Gantmacher 1960). We wrote our own lecture notes both on the Gröbner basis algorithm based on Buchberger (1985) survey article and on the Hermite normal form algorithm (see, e.g., (Sims 1984)). Very little time, perhaps one lecture in total, were devoted to teaching the subtleties of Mathematica and its programming language, as this was designed as a course in mathematics and not in programming.

Students' grades were determined from seven sets of homework problems (see subdirectory `Homeworks`), one mid-semester in-class written examination, and one take-home final examination (see subdirectory `Exams`). Furthermore, the students had to do one programming project at the end of the semester. Students were allowed to form teams of two for carrying out their projects. One of the projects was, for example, the implementation of Uspensky's method (see, e.g., (Collins and Loos 1982)) for computing the real roots of a polynomial. Due to the template-like structure of the packages provided by me, the students had no difficulty implementing their algorithms, as the procedures from an algorithm-design point of view were quite simple. None of the homework problems required programming, but several of them could only be done by using Mathematica and the provided packages. The problems on the final examination were designed specifically so that they required the assistance of Mathematica in their solution. For example, Problem 3 on the final exam was to find an integer solution to the equation $x^2 + y^2 = 10^{25} + 13$ via the Gaussian integer GCD of $10^{25} + 13$ and $z + i$, where z is a residue satisfying $z^2 + 1 \equiv 0 \pmod{10^{25} + 13}$.

3. Software Written for the Course

The students were given Mathematica packages for all topics, with the exception of Gröbner basis computation and Hermite form computation. These two latter packages were produced by student teams as semester-end projects. In order to demonstrate the principles of abstract algebraic domains, the software was designed genericly (Musser and Stepanov 1989). Genericity means that the supplied procedure definitions are usable for inputs in an unspecified domain, and was accomplished in Mathematica by the use of function arguments. As an example, consider the extended Euclidean algorithm. Following is the Mathematica code for the generic procedure defined in the package `Euclid`.

```
ExtendedGeneric::usage =
```

```
"ExtendedGeneric[a_, b_, quotient_, unitnormalize_]
 returns {g, s, t} such that g == GCD[a, b] and s a + t b = g,
 where quotient must be bound to a quotient function on the type
 of elements in a and b and unitnormalize must be a function
 for normalizing the GCD to g by multiplying by a unit."

ExtendedGeneric[a_, b_,
                quotient_, (* the quotient function *)
                unitnormalize_:Identity
                   (* function returning its unit-normalized argument*)
               ]:=
(* The extended Euclidean algorithm for Euclidean domains *)
   Block[{x, y, q, sx, sy, tx, temp, g, u},
         x = a; y = b; sx = 1; sy = 0;
         While[y =!= 0, (* != does not work on polynomials *)
               ETrace[StringForm["Dividend=``, Divisor=``, sy=``", x, y, sy] ];
               q = quotient[x, y];
               temp = y;  y  = Expand[x  - q y];  x  = temp;
               If[y =!= 0,
                     (* unit-normalize the new remainder *)
                     temp = unitnormalize[y];
                     u = quotient[temp, y]; y = temp;
                     temp = sy; sy = Expand[u (sx - q sy)]; sx = temp,
                  (* else *) sx = sy
                ]
              ];
         tx = quotient[Expand[x - sx a], b];
         Return[ List[ x, sx, tx ] ]
        ]
```

Next, we exhibit how this procedure can be run with various Euclidean domains. Our examples are the integers with two different Euclidean divisions, the Gaussian integers, and rational polynomials.

```
In[6]:= Euclid`ETrace[s__]:=Print[s]

In[7]:= ExtendedGeneric[4284179, 4288507, System`Quotient, Abs]
Dividend=4284179, Divisor=4288507, sy=0
Dividend=4288507, Divisor=4284179, sy=1
Dividend=4284179, Divisor=4328, sy=-1
Dividend=4328, Divisor=3787, sy=990
Dividend=3787, Divisor=541, sy=-991

Out[7]= {541, -991, 990}

In[8]:= (* Using a quotient function that leaves absolutely smallest remainders *)
ExtendedGeneric[4284179, 4288507, Numbers`Quotient, Abs]
Dividend=4284179, Divisor=4288507, sy=0
Dividend=4288507, Divisor=4328, sy=-1
Dividend=4328, Divisor=541, sy=-991

Out[8]= {541, -991, 990}

In[9]:= ExtendedGeneric[185 - 195 I, -162 - 376 I, Gaussian`Quotient,
                                                   Gaussian`Normalize]
Dividend=185 - 195 I, Divisor=-162 - 376 I, sy=0
```

```
Dividend=-162 - 376 I, Divisor=191 + 33 I, sy=-1
Dividend=191 + 33 I, Divisor=39 + 37 I, sy=-2 + I

Out[9]= {39 + 37 I, -2 + I, 1 + I}

In[10]:= Euclid`ETrace[s__]:=s

In[11]:= f = RandomPoly[4, 10, x]
                        2      3    4
Out[11]= -2 + 9 x - 9 x  + 8 x  + x

In[12]:= g = RandomPoly[3, 10, x]
                         2      3
Out[12]= -2 - 2 x + 10 x  + 7 x

In[13]:= ExtendedGeneric[f, g, Function[PolynomialQuotient[#1, #2, x]],
                               Function[#]]
                                                          2
          2572734630  1532626012   54848814066 x   786769 x
Out[13]= {----------, ---------- + ------------- + ---------,
          1609373689  4368300013    30578100091     762223
                                                       2           3
       35169361069    28598266381 x   37465939780 x    786769 x
>    -(-----------) + ------------- - -------------- - ---------}
       30578100091     30578100091     30578100091      5335561

In[14]:= ExtendedGeneric[f, g, Function[PolynomialQuotient[#1, #2, x]],
                               Function[MakeMonic[#,x]]] (* monic remainders *)
                                  2                                 2         3
             6818    11619 x   40117 x     44701    36349 x   4762 x    5731 x
Out[14]= {1, ----- + ------- + --------, -(-----) + ------- - ------- - -------}
             31065    10355     62130      62130     62130     6213      62130
```

Already in this simple algorithm, a non-trivial observation can be made. By normalizing the remainders to leading coefficient 1 in the polynomial remainder chain, the occurring reduced rational coefficients stay smaller. The mathematical explanation of this phenomenon by the fundamental theorem of subresultants could, however, not be given in this course.

We now list the packages and their major procedures. The Mathematica code can be retrieved from the subdirectory `Packages` and Mathematica scripts ("notebooks") for demonstration of the packages' functionality from the subdirectory `Notebooks`.

`Numbers`: The functions `PeasantMultiplication`, `BinaryExponentiation`, and `GenericExponentiation` all demonstrate the doubling algorithm for fast "exponentiation"; `FermatPrimeQ` is the Fermat primality test (which fails on Carmichael numbers); `CycleIndices` is Floyd's method for finding a cycle in a periodic function; `PollardRho` is the Pollard ρ integer factoring algorithm. Demo notebooks `expo`, `rho`.

`Gaussian`: Euclidean quotient, remainder, and unit normalization functions for Gaussian integers. Used for examples of the Euclidean algorithm (see above).

`Polynom`: Auxiliary functions for polynomial manipulation, such as `RootBound` and `FactorCoefficientBound` that bound the absolute values of the roots and coefficients of irreducible factors of integer polynomials; both are used in the packages `Cauchy` and `Factor`; function `SwinnertonDyer` compute the so-called Swinnerton-

Dyer polynomials that constitute especially difficult inputs for the polynomial factorization algorithms (see demo notebook `factor`).

`Euclid:` Functions that implement the Euclidean algorithm and its derivatives generically, such as `ExtendedGeneric`, the extended Euclidean algorithm (see above) and `ContFractGeneric`, which finds a quotient sequence in an abstract Euclidean domain. The latter function is used in the packages `Sturm` and `Routh`. Demo notebooks `xgcd`, `cont_frac`.

`Sturm:` The function `RealRoots` locates the real roots of a rational polynomial to a given precision by Sturm's bisection method. The auxiliary function `Sturm‘SturmSequence` computes the sequence of polynomial quotients that correspond to a Sturm sequence of a rational polynomial and its derivative. Demo notebooks `rroots`, `cheby`.

`Cauchy:` Functions for visualizing complex polynomial functions and the argument principle. See §4. Demo notebook `cauchy`.

`Routh:` The function `ComplexRoots` locates the complex roots of a rational polynomial to a given precision by the Routh-Hurwitz algorithm. See §4. The auxiliary function `Routh‘SturmSequence` computes the the Sturm sequence of two rational polynomials needed for the Cauchy index computation. Demo notebook `croots`.

`Factor:` Function `FactorPolyMod` factors an integer polynomial modulo a prime number using the Cantor-Zassenhaus method; function `FactorSqfPrimPoly` factors a squarefree primitive integral polynomial using the large primes Cantor-Zassenhaus method and factor combination. No Hensel lifting is necessary, since the prime modulus is chosen larger than the factor coefficient bound. Demo notebook `factor`.

`Groebner:` David Ng's educational package for the Buchberger algorithm for computing Gröbner bases. The function `Basis` finds the reduced Gröbner basis with respect to lexicographic variable order. Demo notebook `groebner`.

4. Visualization of a Mathematical Fact: Complex Root Finding

We now discuss the use of Mathematica for the teaching of a classical topic, that of finding complex roots of a polynomial. The fundamental theorem of algebra combines the subjects of functions in a complex variable, real analysis, and abstract algebra. In the course we begin our discussion with the argument principle: the change of the argument of the value of a polynomial function along a simple closed contour is 2π (i.e., $360°$) $\times$ the number of roots contained within the contour. It is assumed that there are no roots on the curve and that the path follows a counterclockwise direction. Figure 1 depicts a visualization of this mathematical fact. We plot the absolute value of the polynomial function $z^3 - 2$ as the height of the surface over the Gaussian plane and the argument as the color. For purpose of reproduction here we use a grey-level ranging from 0 = black to 2π = white. On a computer in class we would use rainbow-like hue values ranging from red, to yellow, green, blue, purple, pink, back to red. The roots are seen as vortices of color. On a contour drawn around all three roots one will encounter three full changes of the color spectrum. The students are made aware of the four-dimensionality of the shown structure, the fourth dimension being the color.

Further study of the phenomenon is made possible by the plots provided in the package `Cauchy‘`. Figure 2 plots the chosen rectangular contour and the positions of the three complex roots of the function z^3-2. Triangular and circular contours are also provided by the package. This plot is meant as an aid for the spacecurve depicted in Figures 3 and 4. As the z-coordinate progresses from 0 to 4, where each unit increment corresponds to

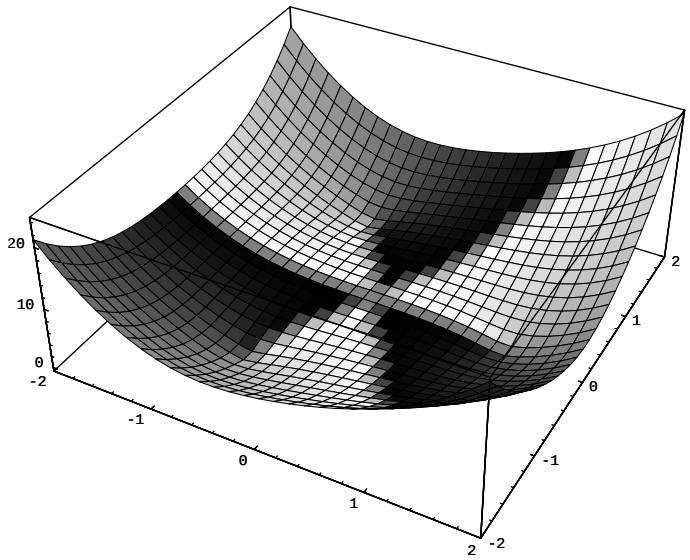

Figure 1. `Cauchy‘PlotComplexFunction[z^3-2, z, {-2, 2}, {-2, 2}, PlotPoints->30]`

an edge of the rectangular contour, the x-y values correspond to the real and imaginary parts of the function value. Viewing the spacecurve from directly above, one sees the two revolutions around the origin, which is a change of argument by 4π. Immediately, the problem of measuring the change of argument when the contour runs through the origin, that is, when a root lies on it, is exhibited. In the subsequent algorithm this situation must be avoided. The students were encouraged to use these visualization tools to confirm the theorem on Mathematica for their choices of polynomials.

At this point in the course, the students already know Sturm's theorem, which measures a global quantity, namely the number of real zeros in an interval, by local observations, namely the number of sign variations of the Sturm sequence at the endpoints. Its generalization to the measurement of the change of argument by computing some local quantity at the vertices of the polygonal contour is, however, not at all obvious. From Figure 4 one can argue that it suffices to count the crossings of the real axis from positive to negative or negative to positive imaginary values on each side of the imaginary axis. Since the polynomial function on each straight edge of contour can be expresses as $f(z) = u(t) + iv(t)$, where u and v are polynomials in the real parameter t with $0 \leq t \leq 1$, the real roots of v in relation to the real roots of u appear useful for counting the crossings. The author has given these considerations as a motivation for the Cauchy index of u/v, which ultimately yields an analog to Sturm's bisection method for complex root isolation and approximation problem. The special case $f(it)$, where $-\infty < t < \infty$, counts the number of roots on the right half-plane and is Routh's 1875 stability criterion.

Various implementations and papers on this approach have been written in the more recent past (Pinkert 1976, Collins 1977, Wilf 1978, Collins and Krandick 1992). The author's package is purely educational and does not account for any efficiency considerations. We should mention, though, that the sign variation of the Sturm sequence for u, v at any endpoint of a line segment must be adjusted when $v = 0$ at that endpoint.

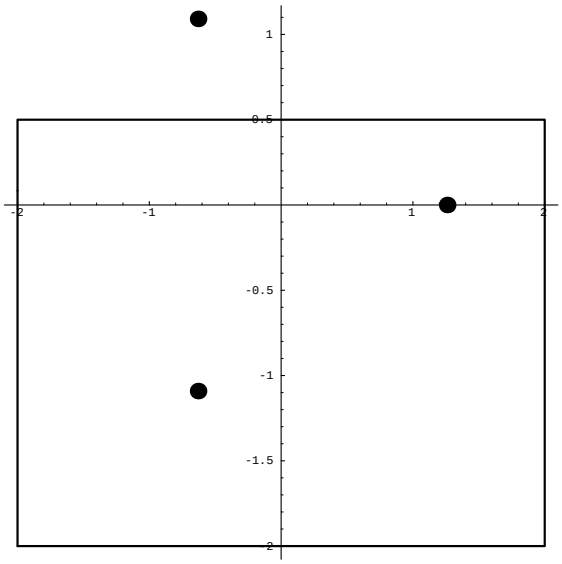

Figure 2. `Cauchy‘PlotRootsContour[z^3 - 2, z, ComplexRectangle[-2-2 I, 2+I/2, t], {t, 0, 4}]`

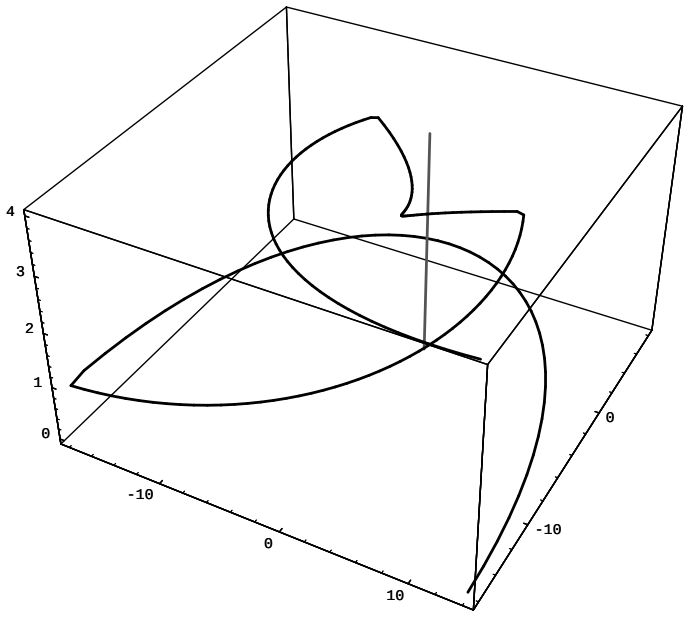

Figure 3. `Cauchy‘PlotArgumentPrinciple[z^3 - 2, z, ComplexRectangle[-2-2 I, 2+I/2, t], {t, 0, 4}]`

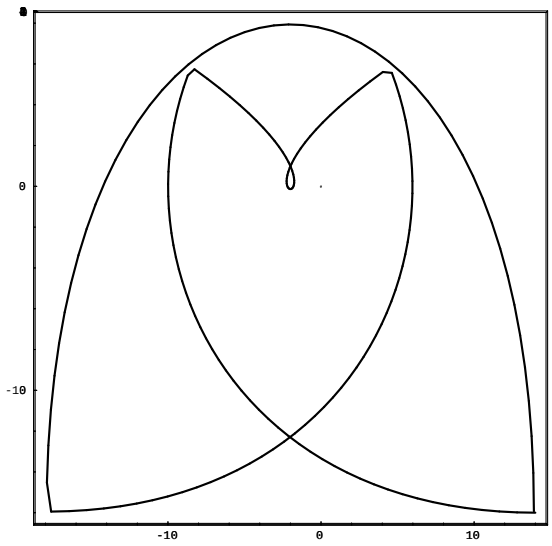

Figure 4. `Cauchy‘PlotArgumentPrinciple[z^3 - 2, z, ComplexRectangle[-2-2 I, 2+I/2, t], {t, 0, 4}, ViewPoint->{0,0,100}]`

As can be proven, one may simply add $\frac{1}{2}$ to the computed variation in order to obtain the correct count (see the function `Routh‘Variation`). A partial trace of the author's implementation of the bisection method follows.

```
In[5]:= Routh‘ComplexRoots[z^3 - 2, z, 1/8]
                29          997      35          128
lineS = {{-1, -(--) + t, -(-----), -(--) + t, -(---), -32 + 64 t, -96, 0}, t,
                64          12288    64           3

>    {0, 3}, {1, 1}}
                1                                  1    2 t
lineE = {{-1, -(-) + t, -6, -48 + 96 t, -44, -(-) + ---}, t, {0, 2}, {1, 1}}
                2                                  3    3
                35          997      29          128
lineN = {{-1, -(--) + t, -(-----), -(--) + t, -(---), -32 + 64 t, -96, 0}, t,
                64          12288    64           3

>    {0, 3}, {1, 1}}
                1                             124     1    2 t
lineW = {{-1, -(-) + t, -10, -48 + 96 t, -(---), -(-) + ---}, t, {0, 2}, {1, 1}}
                2                              3      3    3

Box searched: {-2 - 2 I, 2 - 2 I, 2 + 2 I, -2 + 2 I}, Delta arg =3

                                                                 2       3
New sequence between -2 I and 2 I, ss={-1, -4 + 24 t - 48 t  + 32 t }

                1
>   , vertratio=-
```

```
               2
1 real root(s) on contour
New sequence between 2 - I and -2 - I, ss=

           11         11    5         32                                  1
>   {-1, -(--) + t, -(---), -- - t, -(--), 32 - 64 t, -48, 0}, horzratio=-
           16         768   16        3                                   4
Box searched: {-2 - 2 I, -2 I, -I, -2 - I}, Delta arg =1

...

10.5 Second

           5    17 I    5    37 I  21   I
Out[5]= {-(-) - ----, -(-) + ----, -- + --}
           8     16     8     32   16   32
```

In the above, e.g., `lineS` shows the quotient sequence of $u(t)$ and $v(t)$ along the Southern edge of the enclosing rectangle and the variations at $t = 0$, in this case 3, and $t = 1$, namely 1. The corresponding Cauchy index is thus 2, and the sum of the Cauchy indices of all edges is 6, which is the change of argument $\div \pi$. Hence there are all 3 roots within the first contour. Now bisectors are drawn and the roots enclosed in the respective quadrants are counted. The second bisector contains one of the roots and is replaced by one lying an imaginary unit lower. Again, the students are encouraged "to play" with the package in order to gain mathematical insight. They can design their own contours and count the argument change for different polynomials and they can construct invalid (e.g., root on contour) and boundary cases (e.g., endpoint on real axis). Finally, we show the Mathematica code that performed the initial computation shown.

```
ComplexRoots[f_, z_, precision_]:=
Block[{b, SW, SE, NW, NE, ssS, ssE, ssN, ssW,
       lineS, lineE, lineN, lineW, Darg, t=Global`t},

  b = Polynom`RootBound[f, z];

  SW = -b-I*b; SE = b-I*b; NE = b+I*b; NW = -b+I*b;

  ssS = Routh`SturmSequence[ SeparateFunctionOnLine[f, z, SW, SE, t], t];
  lineS = List[ssS, t, {0, Variation[ssS, t, 0]}, {1, Variation[ssS, t, 1]}];
  RTrace[ StringForm["lineS = ``", lineS] ];

  ssE = Routh`SturmSequence[ SeparateFunctionOnLine[f, z, SE, NE, t], t];
  lineE = List[ssE, t, {0, Variation[ssE, t, 0]}, {1, Variation[ssE, t, 1]}];
  RTrace[ StringForm["lineE = ``", lineE] ];

  ssN = Routh`SturmSequence[ SeparateFunctionOnLine[f, z, NE, NW, t], t];
  lineN = List[ssN, t, {0, Variation[ssN, t, 0]}, {1, Variation[ssN, t, 1]}];
  RTrace[ StringForm["lineN = ``", lineN] ];

  ssW = Routh`SturmSequence[ SeparateFunctionOnLine[f, z, NW, SW, t], t];
  lineW = List[ssW, t, {0, Variation[ssW, t, 0]}, {1, Variation[ssW, t, 1]}];
  RTrace[ StringForm["lineW = ``", lineW] ];
  Darg = (vStart[lineS] - vEnd[lineS] +
          vStart[lineE] - vEnd[lineE] +
          vStart[lineN] - vEnd[lineN] +
```

```
          vStart[lineW] - vEnd[lineW]
        ) / 2;

  If[Darg != Exponent[f, z],
     Print["Inconsistency in Routh\‘ComplexRoots"];
     Return["Error in Routh\‘ComplexRoots"]
    ];

  Return[ BoxRoots[f, z, {SW, SE, NE, NW}, Darg,
                   {lineS, lineE, lineN, lineW}, precision]
        ]
] (* end ComplexRoots *)
```

We have chosen the Routh-Hurwitz approach to root finding as a demonstration of applying abstract algebraic tools such as the Euclidean algorithm for polynomials to a concrete computational problem. A main point is that the generic code written for continued fraction expansion of a rational function in a Euclidean domain is used directly in our application. Therefore, the concepts learned from abstract algebra require no translation for the analytical application. Moreover, the graphical tools in modern symbolic mathematical systems allow for unusual visualizations of complex polynomials.

5. Assessment

In general, the students reacted quite positive to the course. Judging from the end-of-semester course evaluations, however, the course format was not uniformly endorsed by the students. Surprisingly, most students suggested to incorporate Mathematica to a greater extent in the course, perhaps even have a Mathematica laboratory, while at the same time de-emphasising mathematical proofs. In terms of subject selection, the students felt that the material covered was too vast, and that some basic topics should be covered to greater depth. On the question "which topics were most interesting/least interesting to you?" individual responses were

most interesting	least interesting
GCDs, Hermite forms	complex root finding
Euclid's algorithm	Nullstellensatz
integer factorization	ring theory (ideals, UFDs)
Gröbner basis	integer factorization
integer factoring, root finding	Hermite forms
	complex root finding
root finding, Peano arithmetic	integer factorization

Such mixed reaction can be viewed positively in that no subject was disliked by a large subset of the students. The selection of topics needs further consideration. Clearly, group theory could be incorporated in a computational manner by use of a special purpose system such as GAP or MAGMA (Bosma *et al.* 1994). Alternately, Mathematica code for permutation groups could be developed. In turn the topic of complex root finding or polynomial factorization could be dropped without the loss of coherence.

The usage of a computer algebra system in a course of abstract algebra was an unqualified success of this educational experiment. As mentioned above, the students would have liked even more computer involvement in the course. We realize that since the course was

offered only once five years ago with twelve students enrolled such a conclusion might be premature. At Rensselaer Polytechnic Institute, a graduate course entitled *Symbolic Mathematical Computation* is offered bi-annually. We have been using tools as the ones described here in the graduate course in the subsequent years. A main difference is that in the Symbolic Mathematical Computation course algorithm and system design issues become the subject matter.

Computer visualization of mathematical facts such as the argument principle or a reduction in a Gröbner basis appear to be very powerful teaching tools. In particular, the abstract algebra subject matter, a significant part of which comes from the 19th century, is transformed into a modern mathematical problem solving tool by the computer algorithmic approach. It never was the author's intention to provide the best-known computer solutions to the problems considered, but instead provide educational programs that reflect the mathematics taught in class most clearly. For instance, the fastest infallible complex root finders do not use Sturm sequences (Schönhage 1982), although they are still based on Cauchy integration. It was pointed out to the students that the Mathematica library routines in many instances use more sophisticated algorithms, but that in many cases they are based on the same underlying mathematical principles.

The course was also taken by computer science students. For them, the demonstration of software design using parametric types is an added pedagogical bonus. Indeed, computer algebra systems like AXIOM (Jenks and Sutor 1992) are specifically designed to facilitate such generic algebraic programming. Surely, the course packages could have as easily been written in AXIOM or the GAUSS subsystem of MAPLE (Monagan 1993). Although the development effort for such software may appear high at first, the freely available Mathematica code developed by us should help in the computer approach to algebra teaching elsewhere.

A surprising aspect of reform in higher mathematical education is that systems like Mathematica and Maple have had major impact on the teaching of calculus. I believe that they are equally—if not better—suited to modernize instruction in linear algebra and abstract algebra. To some extent, the algorithms present in such systems are more advanced in those subjects and mathematical abstraction has been incorporated in their design out of a necessity of covering a wider range of mathematics. Clearly, the author's attempt in squeezing modern algebra in a single semester course caused me to omit traditional subjects, such as group and Galois theory. The author's subject selection was intended to fit the needs of non-mathematics students, as modern symbolic mathematical software makes algebra a powerful problem solving tool for general scientists and engineers.

Acknowlegdement

I would like to express my gratitude to the two unnamed referees, whose suggestions have improved the final version of the paper.

References

Bosma, W., Cannon, J., Matthews, G. (1994). Programming with algebraic structures: design of the magma language. In von zur Gathen, J., Giesbrecht, M., editors, *Proc. Internat. Symp. Symbolic Algebraic Comput. ISSAC '94*, pages 52–57, New York, N. Y. ACM Press.

Buchberger, B. (1985). Gröbner bases: An algorithmic method in polynomial ideal theory. In Bose, N. K., editor, *Recent Trends in Multidimensional Systems Theory*, pages 184–232. D. Reidel Publ. Comp., Dordrecht (Holland).

Collins, G. E. (1977). Infallible calculation of polynomial zeros to specified precision. In Rice, J. R., editor, *Mathematical Software III*, pages 35–68, New York. Academic Press.

Collins, G. E., Krandick, W. (1992). An efficient algorithm for infallible polynomial complex root isolation. In Wang, P. S., editor, *Proc. Internat. Symp. Symbolic Algebraic Comput. ISSAC '92*, pages 189–194, New York, N. Y. ACM Press.

Collins, G. E., Loos, R. (1982). Real zeros of polynomials. In et al, B. B., editor, *Computer Algebra, 2nd ed.*, pages 83–94, Vienna. Springer Verlag.

Gantmacher, F. R. (1960). *The Theory of Matrices, Vol. 1.* Chelsea Publ. Co., New York, N. Y.

Jacobson, N. (1974). *Basic Algebra I.* W. H. Freeman & Co., San Francisco.

Jenks, R. D., Sutor, R. S. (1992). *axiom The Scientific Computing System*. Springer Verlag, New York.

Knuth, D. E. (1981). *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, Ed. 2*. Addison Wesley, Reading, MA.

Lipson, J. (1981). *Elements of Algebra and Algebraic Computing*. Addison-Wesley Publ., Reading, Mass.

Monagan, M. B. (1993). Gauss: A parameterized domain of computation system with support for signature functions. In Miola, A., editor, *Proc. DISCO '93*, volume 722 of *Springer Lect. Notes Comput. Sci.*, pages 81–94.

Musser, D. R., Stepanov, A. A. (1989). *The Ada Generic Library: List Processing Packages*. Springer Verlag, New York, NY.

Pinkert, J. R. (1976). An exact method for finding roots of a complex polynomial. *ACM Trans. Math. Software*, 2(4):351–363.

Schönhage, A. (1982). The fundamental theorem of algebra in terms of computational complexity. Tech. report, Univ. Tübingen.

Sims, C. C. (1984). *Abstract Algebra, A Computational Approach*. Wiley, New York.

van der Waerden, B. L. (1940). *Moderne Algebra*. Springer Verlag, Berlin. English transl. publ. under the title "Modern algebra" by F. Ungar Publ. Co., New York, 1953.

Wilf, H. S. (1978). A global bisection algorithm for computing the zeros of polynomials in the complex plane. *J. ACM*, 25(3):415–420.

Wolfram, S. (1988 and 1991). *Mathematica A System for Doing Mathematics by Computer*. Addison-Wesley, Redwood City, California.