Generic Linear Algebra and Quotient Rings in Maple

Simon Lo, Michael Monagan, Roman Pearce[*]
Department of Mathematics, Simon Fraser University,
8888 University Drive,
Burnaby, BC, Canada V5A 1S6.
sclo@sfu.ca, mmonagan@cecm.sfu.ca, rpearcea@cecm.sfu.ca

Abstract

The algorithms for linear algebra in the Magma and Axiom computer algebra systems work over an arbitrary ring. For example, the implementation of Gaussian elimination for reducing a matrix to (reduced) row Echelon form works over any field that the user constructs. In contrast, Maple's facilities for linear algebra in its `LinearAlgebra` package only work for specific rings. If the input matrix contains general expressions, the algorithms may work incorrectly.

Motivated by a need to do linear algebra over quotient rings and finite fields in Maple, we have designed a simple to use facility that permits the Maple user to define a field, Euclidean domain, integral domain or ring so that our "generic" algorithms for linear algebra are immediately available. These algorithms comprise a package called `GenericLinearAlgebra` which is being integrated into Maple 11.

We have also implemented a package for computing in quotient rings. This package, called `QuotientRings`, exports all the necessary operations so that one can immediately do linear algebra over a quotient ring, for example, the trigonometric polynomial ring $\mathbb{Q}[s,c]/\langle s^2+c^2-1\rangle$. The package also includes a new algorithm for simplifying fractions over a quotient ring to canonical form that we discuss.

1 Introduction

The `GenericLinearAlgebra` package implements linear algebra operations over abstract commutative rings, fields, Euclidean domains, and integral domains. This means that if one can do arithmetic in a given ring R, such as a finite field $\mathrm{GF}(p^k)$, then one can multiply matrices, compute determinants, and solve linear systems over R as well. The package exports the following commands.

[*]This work was supported by NSERC and MITACS NCE of Canada.

```
> with(GenericLinearAlgebra);
```

$$[BareissAlgorithm, BerkowitzAlgorithm, CharacteristicPolynomial, Determinant, GaussianElimination, HermiteForm, HessenbergAlgorithm, HessenbergForm, LinearSolve, MatrixInverse, MatrixMatrixMultiply, MatrixVectorMultiply, MinorExpansion, NullSpace, RREF, ReducedRowEchelonForm, SmithForm, StronglyConnectedBlocks]$$

As a first example, we will construct the domain of rational numbers. The representation for a domain is either a table of Maple functions and constants, or a module that exports Maple functions and constants. Now the rational numbers are a field, so we must define $+$, $-$, $\times$, $\div$. We must also define $=$ for testing if two elements of a field are equal, and specify the constants 0 and 1. In our design, we require that the definitions for $+$ and $\times$ are n-ary, that is, we can add/multiply zero or more inputs. Here is code for defining the rationals using the table representation.[1]

```
> Q[`-`] := proc(a,b) a-b end:
> Q[`+`] := proc() local x; add(x, x=[args]) end;
> Q[`*`] := proc() local x; mul(x, x=[args]) end;
> Q[`/`] := proc(a,b) a/b end:
> Q[`=`] := proc(a,b) evalb(a=b) end proc:
> Q[`0`] := 0:
> Q[`1`] := 1:
```

It is now possible to use this 'domain' with the GenericLinearAlgebra package. The domain is passed as an index to the command, as shown below.

```
> A := Matrix([[1,2,3],[4,5,6],[7,8,9]]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
> GaussianElimination[Q](A);
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix}$$

```
> ReducedRowEchelonForm[Q](A);
```

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix}$$

For 'fraction-free' elimination over an integral domain, one must define an operation for exact division. The function Divide(a, b) should return *true* if b divides a, and Divide(a, b, q) should assign the quotient to q. The following Maple code works.

[1]Note, since '+', '−', '*', and '/' are built-in Maple functions with the required functionality, we could also use `Q[`+`] := `+`;`, `Q[`-`] := `-`;`, etc.

```
> Q[‘Divide‘] := proc(a,b,q::name) evalb(irem(a,b,q)=0) end:
```

We will run the Bareiss fraction-free algorithm over $\mathbb{Z}$. This algorithm reduces a matrix to row Echelon form using exact division. It is used by default by the `LinearSolve` command, although fractions are constructed during back substitution. The reason the Bareiss algorithm is used intead of Gaussian elimination when reducing a matrix to row Echelon form is that the size of an inverse of a matrix element may be very large. See section 3 for an example. Note that both algorithms perform the same *number* of arithmetic operations, namely, $O(n^3)$ where n is the dimension of the matrix.

```
> b := Vector([5,3,1]);
```

$$b := \begin{bmatrix} 5 \\ 3 \\ 1 \end{bmatrix}$$

```
> BareissAlgorithm[Q](Matrix([A,b]));
```

$$\begin{bmatrix} 1 & 2 & 3 & 5 \\ 0 & -3 & 6 & -17 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

```
> LinearSolve[Q](A,b,output=[solution],free=[t]);
```

$$\begin{bmatrix} -\frac{19}{3} + t \\ \frac{17}{3} - 2t \\ t \end{bmatrix}$$

1.1 Coding Style

Coding algorithms for the generic linear algebra package is straight-forward. Below is the code for the `RREF` command which illustrates how one accesses operations from a domain, in this case from the field `F`. As remarked above, to avoid computing inverses, by default, the code uses the Bariess algorithm if exact division – if the `Divide` operation – is available, otherwise it uses ordinary Gaussian elimination.

```
proc(B::Matrix,rank::{name,equation},det::{name,equation})
local F,i,j,c,m,n,r,A,alg,opts;
  opts := select(type, [args[2..nargs]], 'equation');
  hasoption(opts,'method'={identical(GaussianElimination),
            identical('BareissAlgorithm')},'alg','opts');
  if opts<>[] then error "unrecognized option(s): %1", opts fi;
  if not assigned(alg) then
    if HasOperation(F,Divide) then alg := BareissAlgorithm
    else alg := GaussianElimination; end if;
  end if;

  F := GenericCheck( procname, FieldOperations );
  n,m := LinearAlgebra:-Dimensions(B);
```

```
    if alg=GaussianElimination then
      A := GaussianElimination[F](op(remove(type, [args], equation)));
      r := 1;
      for c to m while r <= n do
        for i from r to n while F[`=`](A[i,c],F[`0`]) do od;
        if i > n then next fi;
        for i to r-1 do
          if F[`=`](A[i,c],F[`0`]) then next fi;
          for j from c+1 to m do
            A[i,j] := F[`-`](A[i,j],F[`*`](A[i,c],A[r,j])) od;
          A[i,c] := F[`0`]
        od;
        r := r + 1      # go to next row
      od;         # go to next column
    else
      A := BareissAlgorithm[F](op(remove(type, [args], equation)));
      for r from n to 1 by -1 do
        for c from min(r,m) to m while F[`=`](A[r,c],F[`0`]) do od;
        if c > m then next end if;
        for j from c+1 to m do A[r,j] := F[`/`](A[r,j],A[r,c]) od;
        A[r,c] := F[`1`];
        for i to r-1 do
          for j from c+1 to m do
            A[i,j] := F[`-`](A[i,j], F[`*`](A[i,c], A[r,j])) od;
          A[i,c] := F[`0`]
        od;
      od;
    fi;
    A
  end:
```

Note, one must not use Maple's = operation to test for equality because the test in `if A[i,c]=F[`0`] then ...` assumes that the representation for the 0 in the domain is unique. Our design does not assume a unique representation for 0.

In the Maple library there is an older package called the `Domains` package that was designed by Monagan [6] in the early 1990s. The design of the representation for rings and fields, and the way one implements algorithms in our generic linear algebra package is similar to that used in the `Domains` package. What are the differences between the `Domains` package and our new package?

In `Domains`, a field has many operations. Everything that you might conceivably do with a field needs to be there. And since a field is a Euclidean domain, and a Euclidean domain is a unique factorization domain (UFD) which is an integral domain, which is a ring, everything that you might do in a ring, integral domain, UFD, and Euclidean domain is also there. The main difference between the Domains package and our generic linear algebra package is that we have deliberately kept the number of operations that the user needs to define for a domain to a minimum. The operations needed are largely determined by the algorithm. For example, in order to run the Bareiss algorithm over an integral domain, we do not want the user to have to define the characteristic of the domain, provide a test for whether an element is a unit or not, etc. etc.

Another advantage of the new design is that it is much easier to modify and repackage existing parts of the Maple library so that they can work with our generic linear algebra package. The next example illustrates this.

1.2 Finite Fields

Although Maple has had routines for linear algebra modulo n for quite some time, it was historically difficult to do linear algebra over GF(p^k) where $k > 1$. This is no longer the case, as Maple's GF package has been modified to work with `GenericLinearAlgebra` for Maple 11. The modification needed was almost trivial. The GF package already exported '+', '-', '*', '/' so we needed only to add '0', '1' and '='. Below we construct GF(2^4) as polynomials in $\mathbb{Z}_2[x]$ modulo $x^4 + x + 1$. Notice that this time the representation of the domain is a module with the required exports.

```
> GF16 := GF(2,4,x^4+x+1);
```

$$GF16 := \mathbb{Z}_2[x]/\langle x^4 + x + 1\rangle$$

```
> print(GF16);
```

```
module()
export `+`, `-`, `*`, `/`, `^`, `0`, `1`, `=`, input, output,
inverse, extension, variable, factors, norm, trace, order, random,
size, isPrimitiveElement, PrimitiveElement, ConvertIn, ConvertOut,
zero, one, init;
end module
```

The GF package uses a special representation for the elements of the field. It encodes the polynomials as 'modp1' polynomials. The modp1 facility was added to the Maple kernel by Monagan [5] in the early 1990s to support efficient computation in $\mathbb{Z}_m[x]$. See `?modp1` in Maple for details of this representation. So it is necessary to first convert our matrix entries into the desired format.

```
> A := Matrix([[x^3, x^2+1, x], [x, x+1, 0], [x+1, 0, x^2+1]]):
> A := map(GF16:-ConvertIn, A);
```

$$A := \begin{bmatrix} x^3 \mod 2 & (x^2+1) \mod 2 & x \mod 2 \\ x \mod 2 & (x+1) \mod 2 & 0 \mod 2 \\ (x+1) \mod 2 & 0 \mod 2 & (x^2+1) \mod 2 \end{bmatrix}$$

```
> B := MatrixInverse[GF16](A); # from GenericLinearAlgebra package
```

$$B := \begin{bmatrix} 1 \mod 2 & (x+1) \mod 2 & (x^2+1) \mod 2 \\ (x^3+x^2+x+1) \mod 2 & (x^3+x^2) \mod 2 & (x^2+x) \mod 2 \\ (x^3+x^2+x) \mod 2 & 1 \mod 2 & x^3 \mod 2 \end{bmatrix}$$

```
> MatrixMatrixMultiply[GF16](A,B);
```

$$\begin{bmatrix} 1 \mod 2 & 0 \mod 2 & 0 \mod 2 \\ 0 \mod 2 & 1 \mod 2 & 0 \mod 2 \\ 0 \mod 2 & 0 \mod 2 & 1 \mod 2 \end{bmatrix}$$

2 Algorithms and Commands

Many of the known algorithms for linear algebra are generic in the class of rings they work for. For example, the Hessenberg algorithm (see [3]) computes the characteristic polynomial of a matrix of dimension n over any field F in $O(n^3)$ arithmetic operations. The Berkowitz algorithm (see [1]) computes the characteristic polynomial over any ring R. It is division free, and so can be applied to a matrix over any ring R. The Berkowitz algorithm does $O(n^4)$ multiplications in R.

In our design of the GenericLinearAlgebra package, we have separated algorithms like `BerkowitzAlgorithm` from commands like `CharacteristicPolynomial` cleanly; we export both algorithms and commands. The code for the algorithms is unembellished, it executes the pure algorithm. The code for commands can be smart. It may inspect the matrix (vector), do something clever, and select the appropriate algorithm to use. Consider, for example, the computation of the characteristic polynomial of the following matrix over the integers.

```
> A := Matrix([[1,1,3,2],[0,6,0,1],[3,2,1,2],[0,7,0,2]]);
```

$$A := \begin{bmatrix} 1 & 1 & 3 & 2 \\ 0 & 6 & 0 & 1 \\ 3 & 2 & 1 & 2 \\ 0 & 7 & 0 & 2 \end{bmatrix}$$

We compute the characteristic polynomial using the Berkowitz algorithm. The output is a vector of coefficients. The reason for this output format, instead of outputting a Maple polynomial $x^4-10x^3+13x^2+54x-40$ is firstly, of necessity; we permit the user of GenericLinearAlgebra to use any data representation for the elements of the ring that the user chooses, for example, the user could use hash tables as the data representation for polynomials. Secondly, it is more convenient for programming purposes to get the output as a vector of coefficients.

```
> BerkowitzAlgorithm[Q](A);
```

$$\begin{bmatrix} 1 \\ -10 \\ 13 \\ 54 \\ -40 \end{bmatrix}$$

When computing determinants using the `Determinant` command or characteristic polynomials using the `CharacteristicPolynomial` command, an algorithm is first applied to search for 'strongly connected' blocks in the input matrix. Recall that the directed graph G corresponding to a matrix A has the directed edge $i \to j$ if and only if the matrix entry $A_{i,j} \neq 0$. The strongly connected blocks in A correspond to the strongly connected components in G. The algorithm for computing the strongly connected components of a graph, originally due to Tarjan [10], is linear time in the number of edges in the graph, that is, linear time in the number of non-zero entries in the matrix.

These components would appear as blocks along the diagonal of the matrix if rows and columns were swapped to put the matrix into block upper triangular form. Observe that if you swap rows 2 and 3 and swap columns 2 and 3, the matrix A

becomes

$$\begin{bmatrix} 1 & 3 & 1 & 2 \\ 3 & 1 & 2 & 2 \\ 0 & 0 & 6 & 1 \\ 0 & 0 & 7 & 2 \end{bmatrix}$$

The determinant of A is the product of the determinants of the blocks. The same holds for characteristic polynomials. This block decomposition catches upper and lower triangular matrices as special cases. The command also outputs an integer denoting the number of zero blocks that it found. The block decomposition has been implemented as part of the **LinearAlgebra** package as the **StronglyConnectedBlocks** function.

```
> k, B := StronglyConnectedBlocks[Q](A);
```

$$k, B := 0, \left[\begin{bmatrix} 6 & 1 \\ 7 & 2 \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 3 & 1 \end{bmatrix} \right]$$

```
> CharacteristicPolynomial[Q](A, output=factored);
```

$$0, \left[\begin{bmatrix} 1 \\ -8 \\ 5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ -8 \end{bmatrix} \right]$$

```
> CharacteristicPolynomial[Q](A, lambda, output=factored);
```

$$(\lambda^2 - 8\lambda + 5)(\lambda^2 - 2\lambda - 8)$$

3 Quotient Rings

We have implemented a Maple package called `QuotientRings` for computing in polynomial quotient rings or affine algebras [2, 4]. It exports the following commands.

```
> with(QuotientRings);
```

$$[Associate, Coefficients, Divide, Inverse, IsField, IsIntegralDomain, IsUnit, IsZeroDivisor, KrullDimension, MonomialBasis, MultiplicationMatrix, Normal, QuotientRing, Reduce, RingPolynomial, VectorSpaceDimension]$$

Below we construct the affine algebra $\mathbb{Q}[x, y, z]/\langle x^2+y, z^3-x, xy^2-2\rangle$ and compute it's dimension as a vector space over $\mathbb{Q}$. We will also test whether this domain is a field.

```
> Q := QuotientRing(x^2+y, z^3-x, x*y^2-2);
```

$$Q := Q[x, y, z]/\langle x^2 + y, z^3 - x, xy^2 - 2\rangle$$

```
> VectorSpaceDimension(Q);
```

$$15$$

```
> IsField(Q);
```

$$true$$

Because this domain is a field we can do linear algebra over it. The monomials which appear in the polynomials are the following.

```
> MonomialBasis(Q);
```

$$[xyz^2, y^2z^2, xyz, y^2z, xz^2, yz^2, xy, y^2, xz, yz, z^2, x, y, z, 1]$$

```
> A := Matrix([[x*z^2 + 1, y^2 + x], [x*y-2, x+z+5]]);
```

$$A := \begin{bmatrix} xz^2+1 & y^2+x \\ xy-2 & x+z+5 \end{bmatrix}$$

For solving linear systems over affine algebras, it is usually a good idea to avoid inverting elements until the very end, that is, instead of using Gaussian elimination, one should use the Bareiss algorithm. The reason is that inverses are typically dense polynomials with very large coefficients.

```
> d := Determinant[Q](A);
```

$$5 + 3x + z - z^2y + 5z^2x - 3y + 3y^2$$

```
> Q:-Inverse(d);
```

$$\frac{718434477151}{44610774984795} + \frac{1630819510808}{44610774984795}x - \frac{690116075912}{44610774984795}xy + \frac{1494832352321}{44610774984795}y + \ldots$$

The `QuotientRings` package also supports computations over integral domains and their fields of fractions. In the example below, we construct the trigonometric polynomial ring $\mathbb{Q}[s,c]/\langle s^2+c^2-1\rangle$ and compute the characteristic polynomial of a matrix.

```
> Q := QuotientRing(s^2+c^2-1);
```

$$Q[s,c]/\langle s^2+c^2-1\rangle$$

```
> IsField(Q);
```

$$false$$

```
> IsIntegralDomain(Q);
```

$$true$$

```
> A := Matrix([[s+1,c*s-1,s],[c*s+c,s^2,c],[1,c+s,c-s]]);
```

$$A := \begin{bmatrix} s+1 & cs-1 & s \\ cs+c & s^2 & c \\ 1 & c+s & c-s \end{bmatrix}$$

```
> CharacteristicPolynomial[Q](A, x, method=BerkowitzAlgorithm);
```

$$x^3 + (-2+c^2-c)x^2 + (-2s+sc-2c^2+3c-sc^2-c^3+c^4)x$$
$$+1+2s+c+sc^4-5sc^2-c^5-4c^2+c^3+3sc^3+3c^4.$$

There is also a nice algorithm for reducing fractions to a canonical form.

```
> f := (-c^4+c^2+s*c)/(1+s*c^4-s*c^2+c^3-c+s*c);
```

$$f := \frac{-c^4 + c^2 + sc}{1 + sc^4 - sc^2 + c^3 - c + sc}$$

```
> Q:-Normal(f)
```

$$\frac{sc}{c^3 - c + 1}$$

The algorithm, see [7, 9], works over any quotient ring that is an integral domain. It constructs a Gröbner basis for a module of dimension 2 over the polynomial ideal ring. This results in a canonical representation for the simplified fraction that is unique up to the monomial order used for the polynomial ideal. We discovered that, depending on the ideal, the output from this algorithm may or may not have a common factor present in the numerator and denominator. For the trigonometric polynomial ring shown in the above example, it is true that there can be no common factor between the numerator and denominator in the output. This is because in the trigonometric polynomial ring, we have the following 'degree sum property' [8].

Lemma: Let $R = \mathbb{Q}[s, c]/\langle s^2 + c^2 - 1\rangle$. Let $\phi : R \to R$ with $\phi(x)$ replacing s^2 by $1 - c^2$ in x. Then for all $a, b \in R$ satisfying $\phi(a) \neq 0$ and $\phi(b) \neq 0$ we have

$$\deg \phi(ab) = \deg \phi(a) + \deg \phi(b).$$

However, for some quotient rings, it can happen that the 'simplified' result has a common factor in the numerator and denominator. Consider the following example.

```
> R := QuotientRing(x*y^5-x-y):
> f := (y^5+x+y)/(x-y);
```

$$f := \frac{y^5 + x + y}{x - y}$$

```
> g := Normal[R](f);
```

$$g := \frac{x + y + x^2 + yx}{x^2 - yx}$$

```
> IsUnit[R](x);
```

$$false$$

What has happened is that the numerator and denominator have both been multiplied by x, which is not a unit in R. The result g is nevertheless simpler than f in that the total degree of the fraction f is 6 and of g is 4.

References

[1] J. Abdeljaoued. *The Berkowitz Algorithm, Maple and Computing the Characteristic Polynomial in an Arbitrary Commutative Ring.* MapleTech, **5**(1), pp. 21–32, Birkhauser, 1997.

[2] T. Becker and V. Weispfenning. *Gröbner Bases.* Springer-Verlag, 1993.

[3] H. Cohen. *A Course in Computational Algebraic Number Theory.* Graduate texts in mathematics, **138**, Springer-Verlag, 1995.

[4] D. Cox, J. Little, D. O'Shea. *Ideals, Varieties, and Algorithms.* Second Edition. Springer-Verlag, 1996.

[5] M. B. Monagan. *In-place arithmetic for polynomials over* $\mathbf{Z}_n$. Proceedings of DISCO '92, Springer-Verlag LNCS, **721**, pp. 22–34, 1993.

[6] M. B. Monagan. *Gauss: a Parameterized Domain of Computation System with Support for Signature Functions.* Proceedings of DISCO '93, Springer-Verlag LNCS, **722**, pp. 81–94, 1993.

[7] M.B. Monagan, R. Pearce. *Rational Simplification Modulo a Polynomial Ideal.* To appear in the 2006 proceedings of ISSAC, ACM Press, 2006.

[8] J. Mulholland, M.B. Monagan. *Algorithms for Trigonometric Polynomials.* Proceedings of ISSAC '2001, ACM Press, pp. 245–252, 2001.

[9] R. Pearce. *Rational Expression Simplification with Polynomial Side Relations.* M.Sc. Thesis, Simon Fraser University, 2005.

[10] R. Tarjan. *Depth-First Search and Linear Graph Algorithms.* SIAM Journal on Computing, **1**(2) pp. 146–160, 1972.