Sage and Linear Algebra Worksheet

FCLA Section LT

Robert Beezer
Department of Mathematics and Computer Science
University of Puget Sound

Spring 2017

Sage has very capable linear transformations from $\mathbb{Q}^n$ to $\mathbb{Q}^m$.

1 Creation via Symbolic Functions

Start with a symbolic function.

```
var('x1␣x2␣x3␣x4')
f(x1, x2, x3, x4) = (x1 + 2*x2 + x3 + 5*x4, x1 + 5*x2 + 4*x3
    + 8*x4, -x2 - x3 - x4)
```

Then specify the domain and codomain. We need to be careful about how `T` prints, Sage likes rows.

```
T = linear_transformation(QQ^4, QQ^3, f)
T
```

At a most basic level, `T` behaves as a function.

```
u = random_vector(ZZ, 4, x=-9, y=9).change_ring(QQ)
u, T(u)
```

We can check Theorem LTTZZ, zero goes to zero.

```
z4 = zero_vector(QQ, 4)
z3 = zero_vector(QQ, 3)
z4, T(z4), T(z4) == z3
```

2 Creation via Matrices

We can also create a linear transformation from a matrix, as in Theorem MBLT, with one caveat. For a matrix A, the default is to create the function $T(\mathbf{v}) = \mathbf{v}A$. The keywords option `side='right'` will indicate that we want to put the vector on the right of the matrix.

```
A = matrix(QQ, [[1, 2, 1, 5], [1, 5, 4, 8], [0, -1, -1, -1]])
S = linear_transformation(A, side='right')
```

Notice that we do not have to specify the domain or codomain, these are inferred from the size and type of the matrix. `S` is not new, it is exactly the linear transformation `T` above.

```
S == T
```

Again, notice how `S` prints — the matrix representation is the transpose of what we like to use. This does not *change* the linear transformation as a function, it just changes how we think about it (we like linear combinations of columns, Sage likes linear combinations of rows).

```
A, S
```

3 Creation via Values on a Basis

Starting with a domain and a codomain, we can provide a list of the images of basis vectors for the domain.

```
v1 = vector(QQ, [1, 1, 0])
v2 = vector(QQ, [2, 5, -1])
v3 = vector(QQ, [1, 4, -1])
v4 = vector(QQ, [5, 8, -1])
R = linear_transformation(QQ^4, QQ^3, [v1, v2, v3, v4])
```

That's right — same function again.

```
R == T
```

We can check how this construction works.

```
d3 = R.domain().basis()[2]
R(d3); R(d3) == v3
```

We can give the domain an alternate basis and create a different linear transformation, despite seemingly having the same construction. First we build the domain with a different user basis.

```
u1 = vector(QQ, [1, 0, 0, 0])
u2 = vector(QQ, [1, 1, 0, 0])
u3 = vector(QQ, [1, 1, 1, 0])
u4 = vector(QQ, [1, 1, 1, 1])
dom4 = (QQ^4).subspace_with_basis([u1, u2, u3, u4])
dom4
```

```
L = linear_transformation(dom4, QQ^3, [v1, v2, v3, v4])
L
```

Even though the matrix representation prints the same, this is not the same function, we will need ideas from Chapter R before we can understand the difference.

```
R(u3), L(u3)
```

This code should consistently return `False`.

```
v = random_vector(QQ, 4)
R(v) == L(v)
```

4 Basic Properties

Illustrations with `T`.

```
T.domain()
```

```
T.codomain()
```

A defining property, so always `True`.

```
u = random_vector(QQ, 4)
v = random_vector(QQ, 4)
u, v, T(u+v) == T(u) + T(v)
```

A defining property, so also always `True`.

```
alpha = (QQ).random_element()
u = random_vector(QQ, 4)
alpha, u, T(alpha*u) == alpha*T(u)
```

We can do "arithmetic" with linear transformations, though the addition seems to allow bad things to happen.

```
R+S
```

Scalar multiples also, and they are well-behaved.

```
12*T
```

The following is wrong. (In other words, there is a bug in Sage.)

```
P = R + L
P
```

As we can see...

```
Q = L + R
Q.is_equal_function(P)
```

The problem is that Sage is simply adding the matrices representing the linear transformations, without checking that they are defined using domains with the *same* basis. We will understand the subtlety better in Chapter R.