# Sage and Linear Algebra Worksheet

FCLA Section CB

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

Spring 2017

## 1 A Linear Transformation, Two Vector Spaces, Four Bases

In this section we define a linear transformation from $\mathbb{C}^3$ to $\mathbb{C}^7$ using a randomly selected matrix. The definition is a $7 \times 3$ matrix of rank 3 that we will use to multiply input vectors with a matrix-vector product. It is not important if the linear transformation is injective and/or surjective.

We will build two representations, using a total of four bases — two for the domain and two for the codomain.

```
m, n = 7, 3
A = random_matrix(QQ, m, n, algorithm='echelonizable',
    rank=n, upper_bound=9)
A
```

```
T = linear_transformation(A, side='right')
T
```

The four bases, associated with the two vector spaces.

```
D1mat = random_matrix(QQ, n, n, algorithm='unimodular',
    upper_bound=9)
D1 = D1mat.columns()
D1
VD1 = (QQ^n).subspace_with_basis(D1)
#
D2mat = random_matrix(QQ, n, n, algorithm='unimodular',
    upper_bound=9)
D2 = D2mat.columns()
D2
VD2 = (QQ^n).subspace_with_basis(D2)
#
C1mat = random_matrix(QQ, m, m, algorithm='unimodular',
    upper_bound=9)
C1 = C1mat.columns()
C1
VC1 = (QQ^m).subspace_with_basis(C1)
#
C2mat = random_matrix(QQ, m, m, algorithm='unimodular',
    upper_bound=9)
C2 = C2mat.columns()
```

```
C2
VC2 = (QQ^m).subspace_with_basis(C2)
```

**Exercise 1.** Check out a few of these bases by just asking Sage to display them.

```
D1
```

Now we build two *different* representations.

```
rep1 = T.restrict_domain(VD1).restrict_codomain(VC1)
rep1.matrix(side='right')
```

```
rep2 = T.restrict_domain(VD2).restrict_codomain(VC2)
rep2.matrix(side='right')
```

## 2 Change of Basis Matrices

A natural way to build a change-of-basis matrix in Sage is to adjust the bases for domain and range of the identity linear transformation by supplying an identity matrix to the linear tansformation constructor.

```
identity_domain = linear_transformation(QQ^n, QQ^n,
    identity_matrix(n))
identity_domain
```

```
CBdom =
    identity_domain.restrict_domain(VD1).restrict_codomain(VD2)
CBdom_mat = CBdom.matrix(side='right')
CBdom_mat
```

This matrix should convert between the two bases for the domain. Here's a check of Theorem CB.

```
u = random_vector(QQ, n)
u1 = VD1.coordinate_vector(u)
u2 = VD2.coordinate_vector(u)
u, u1, u2, u2 == CBdom_mat*u1
```

Same drill in the codomain.

```
identity_codomain = linear_transformation(QQ^m, QQ^m,
    identity_matrix(m))
identity_codomain
```

```
CBcodom =
    identity_codomain.restrict_domain(VC1).restrict_codomain(VC2)
CBcodom_mat = CBcodom.matrix(side='right')
CBcodom_mat
```

And here is the check on Theorem MRCB. Convert from domain basis 1 to domain basis 2, use the second representation, then convert back from codomain basis 2 to codomain basis 1 and get as a result the representation relative to the first bases.

```
rep1.matrix(side='right') ==
    CBcodom_mat.inverse()*rep2.matrix(side='right')*CBdom_mat
```

## 3 A Diagonal Representation

We specialize to linear transformations with equal domain and codomain.

First a matrix representation using a square matrix.

```
A = matrix(QQ,
[[ -2,   3, -20,  15,   1,  30,  -5,   17],
 [-27, -38, -30, -50, 268, -73, 210, -273],
 [-12, -24,  -7, -30, 142, -48, 100, -160],
 [ -3, -15,  35, -32,  35, -57,  20,  -71],
 [ -9,  -9, -10, -10,  65, -11,  50,  -59],
 [ -3,  -6, -20,   0,  58,   1,  40,  -55],
 [  3,   0,   5,   0, -10,  -3, -12,    1],
 [  0,   3,   0,   5, -19,  10, -15,   25]])
T = linear_transformation(A, side='right')
T
```

A basis of $\mathbb{C}^8$. And a vector space with this basis.

```
v1 = vector(QQ, [-4,  -6, -1,  7, -2, -4, 1, 0])
v2 = vector(QQ, [ 3, -10, -6, -6, -2,  0, 0, 1])
v3 = vector(QQ, [ 0,  -9, -4, -1, -3, -1, 1, 0])
v4 = vector(QQ, [ 1, -12, -8, -5, -3, -2, 0, 1])
v5 = vector(QQ, [ 0,   3,  2,  2,  1,  0, 0, 0])
v6 = vector(QQ, [ 1,   0,  0, -2,  0,  1, 0, 0])
v7 = vector(QQ, [ 0,   0,  2,  3,  0,  0, 1, 0])
v8 = vector(QQ, [ 3,  -4, -2, -3,  0,  0, 0, 1])
B = [v1, v2, v3, v4, v5, v6, v7, v8]
V = (QQ^8).subspace_with_basis(B)
```

```
S = T.restrict_domain(V).restrict_codomain(V)
S.matrix(side='right')
```

That's a nice representation! Where did the basis come from?

```
A.eigenvalues()
```

Some (right) eigenvectors.

```
(A - 3).right_kernel(basis='pivot').basis()
```

Eigenvalues are a property of the linear transformation.

```
T.eigenvalues()
```

Bases for the eigenspaces depend on the representation, but the actual eigenvectors are also a property of the linear transformation.

```
S.eigenvectors()
```

```
T.eigenvectors()
```

We could do the same thing, but in the style of Section SD, using a change-of-basis matrix.

```
identity = linear_transformation(QQ^8, QQ^8,
    identity_matrix(8))
identity
```

```
CB = identity.restrict_domain(V).restrict_codomain(QQ^8)
CB
```

Here is similarity, in disguise.

```
CBmat = CB.matrix(side='right')
CBmat.inverse()*T.matrix(side='right')*CBmat
```