Section LC Linear Combinations

Sage LC Linear Combinations

Linear Combinations We can redo Example TLC with Sage. First we build the relevant vectors and then do the computation.

```
u1 = vector(QQ,  [ 2, 4, -3,  1,  2, 9])
u2 = vector(QQ,  [ 6, 3,  0, -2,  1, 4])
u3 = vector(QQ,  [-5, 2,  1,  1, -3, 0])
u4 = vector(QQ,  [ 3, 2, -5,  7,  1, 3])
1*u1 + (-4)*u2 + 2*u3 +(-1)*u4
```

```
(-35, -6, 4, 4, -9, -10)
```

With a linear combination combining many vectors, we sometimes will use more compact ways of forming a linear combination. So we will redo the second linear combination of $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$, $\mathbf{u}_4$ using a list comprehension and the `sum()` function.

```
vectors = [u1, u2, u3, u4]
scalars = [3, 0, 5, -1]
multiples = [scalars[i]*vectors[i] for i in range(4)]
multiples
```

```
[(6, 12, -9, 3, 6, 27), (0, 0, 0, 0, 0, 0),
 (-25, 10, 5, 5, -15, 0), (-3, -2, 5, -7, -1, -3)]
```

We have constructed two lists and used a list comprehension to just form the scalar multiple of each vector as part of the list `multiples`. Now we use the `sum()` function to add them all together.

```
sum(multiples)
```

```
(-22, 20, 1, 1, -10, 24)
```

We can improve on this in two ways. First, we can determine the number of elements in any list with the `len()` function. So we do not have to count up that we have 4 vectors (not that it is very hard to count!). Second, we can combine this all into one line, once we have defined the list of vectors and the list of scalars.

```
sum([scalars[i]*vectors[i] for i in range(len(vectors))])
```

```
(-22, 20, 1, 1, -10, 24)
```

The corresponding expression in mathematical notation, after a change of names and with counting starting from 1, would roughly be:

$$\sum_{i=1}^{4} a_i \mathbf{u}_i$$

Using `sum()` and a list comprehension might be overkill in this example, but we will find it very useful in just a minute.

Sage SLC Solutions and Linear Combinations

Solutions and Linear Combinations We can easily illustrate Theorem SLSLC with Sage. We will use Archetype F as an example.

```
coeff = matrix(QQ, [[33, -16,  10,-2],
                    [99, -47,  27,-7],
                    [78, -36,  17,-6],
                    [-9,   2,   3, 4]])
const = vector(QQ, [-27, -77, -52, 5])
```

A solution to this system is $x_1 = 1$, $x_2 = 2$, $x_3 = -2$, $x_4 = 4$. So we will use these four values as scalars in a linear combination of the columns of the coefficient matrix. However, we do not have to type in the columns individually, we can have Sage extract them all for us into a list with the matrix method `.columns()`.

```
cols = coeff.columns()
cols
```

```
[(33, 99, 78, -9), (-16, -47, -36, 2),
 (10, 27, 17, 3), (-2, -7, -6, 4)]
```

With our scalars also in a list, we can compute the linear combination of the columns, like we did in Sage LC.

```
soln = [1, 2, -2, 4]
sum([soln[i]*cols[i] for i in range(len(cols))])
```

```
(-27, -77, -52, 5)
```

So we see that the solution gives us scalars that yield the vector of constants as a linear combination of the columns of the coefficient matrix. Exactly as predicted by Theorem SLSLC. We can duplicate this observation with just one line:

```
const == sum([soln[i]*cols[i] for i in range(len(cols))])
```

```
True
```

In a similar fashion we can test other potential solutions. With theory we will develop later, we will be able to determine that Archetype F has only one solution. Since Theorem SLSLC is an equivalence (Technique E), any other choice for the scalars should not create the vector of constants as a linear combination.

```
alt_soln = [-3, 2, 4, 1]
const == sum([alt_soln[i]*cols[i] for i in range(len(cols))])
```

```
False
```

Now would be a good time to find another system of equations, perhaps one with infinitely many solutions, and practice the techniques above.

Sage SS2 Solving Systems, Part 2

Solving Systems, Part 2 We can now resolve a bit of the mystery around Sage's `.solve_right()` method. Recall from Sage SS1 that if a linear system has solutions, Sage only provides one solution, even in the case when there are infinitely many solutions. In our previous discussion, we used the system from Example ISSI.

```
coeff = matrix(QQ, [[ 1,  4,  0, -1,  0,   7, -9],
                    [ 2,  8, -1,  3,  9, -13,  7],
                    [ 0,  0,  2, -3, -4,  12, -8],
                    [-1, -4,  2,  4,  8, -31, 37]])
const = vector(QQ, [3, 9, 1, 4])
coeff.solve_right(const)
```

```
(4, 0, 2, 1, 0, 0, 0)
```

The vector **c** described in the statement of Theorem VFSLS is precisely the solution returned from Sage's `.solve_right()` method. This is the solution where we choose the α_i, $1 \leq i \leq n-r$ to all be zero, in other words, each free variable is set to zero (how convenient!). Free variables correspond to columns of the row-reduced augmented matrix that are not pivot columns. So we can profitably employ the `.nonpivots()` matrix method. Lets put this all together.

```
aug = coeff.augment(const)
reduced = aug.rref()
reduced
```

```
[ 1  4  0  0  2  1 -3  4]
[ 0  0  1  0  1 -3  5  2]
[ 0  0  0  1  2 -6  6  1]
[ 0  0  0  0  0  0  0  0]
```

```
aug.nonpivots()
```

```
(1, 4, 5, 6, 7)
```

Since the eighth column (numbered 7) of the reduced row-echelon form is not a pivot column, we know by Theorem RCLS that the system is consistent. We can use the indices of the remaining non-pivot columns to place zeros into the vector **c** in those locations. The remaining entries of **c** are the entries of the reduced row-echelon form in the last column, inserted in that order. Boom! So we have three ways to get to the same solution: (a) row-reduce the augmented matrix and set the free variables all to zero, (b) row-reduce the augmented matrix and use the formula from Theorem VFSLS to construct **c**, and (c) use Sage's `.solve_right()` method. One mystery left to resolve. How can we get Sage to give us infinitely many solutions in the case of systems with an infinite solution set? This is best handled in the next section, Section SS, specifically in Sage SS3.

Sage PSHS Particular Solutions, Homogeneous Solutions

Particular Solutions, Homogeneous Solutions Again, Sage is useful for illustrating a theorem, in this case Theorem PSPHS. We will illustrate both "directions" of this equivalence with the system from Example ISSI.

```
coeff = matrix(QQ,[[ 1,  4,  0, -1,  0,   7, -9],
                   [ 2,  8, -1,  3,  9, -13,  7],
                   [ 0,  0,  2, -3, -4,  12, -8],
                   [-1, -4,  2,  4,  8, -31, 37]])
n = coeff.ncols()
const = vector(QQ, [3, 9, 1, 4])
```

First we will build solutions to this system. Theorem PSPHS says we need a particular solution, i.e. one solution to the system, **w**. We can get this from Sage's `.solve_right()` matrix method. Then for any vector **z** from the null space of the coefficient matrix, the new vector $\mathbf{y} = \mathbf{w} + \mathbf{z}$ should be a solution. We walk through this construction in the next few cells, where we have employed a specific element of the null space, `z`, along with a check that it is really in the null space.

```
w = coeff.solve_right(const)
nsp = coeff.right_kernel(basis='pivot')
z = vector(QQ, [42, 0, 84, 28, -50, -47, -35])
z in nsp
```

```
True
```

```
y = w + z
y
```

```
(46, 0, 86, 29, -50, -47, -35)
```

```
const == sum([y[i]*coeff.column(i) for i in range(n)])
```

```
True
```

You can create solutions repeatedly via the creation of random elements of the null space. Be sure you have executed the cells above, so that `coeff`, `n`, `const`, `nsp` and `w` are all defined. Try executing the cell below repeatedly to test infinitely many solutions to the system. You can use the subsequent compute cell to peek at any of the solutions you create.

```
z = nsp.random_element()
y = w + z
const == sum([y[i]*coeff.column(i) for i in range(n)])
```

```
True
```

```
y     # random
```

```
(-11/2, 0, 45/2, 34, 0, 7/2, -2)
```

For the other direction, we present (and verify) two solutions to the linear system of equations. The condition that $\mathbf{y} = \mathbf{w} + \mathbf{z}$ can be rewritten as $\mathbf{y} - \mathbf{w} = \mathbf{z}$, where $\mathbf{z}$ is in the null space of the coefficient matrix. which of our two solutions is the "particular" solution and which is "some other" solution? It does not matter, it is all sematics at this point. What is important is that their difference is an element of the null space (in either order). So we define the solutions, along with checks that they are really solutions, then examine their difference.

```
soln1 = vector(QQ,[4, 0, -96, 29, 46, 76, 56])
const == sum([soln1[i]*coeff.column(i) for i in range(n)])
```

```
True
```

```
soln2 = vector(QQ,[-108, -84, 86, 589, 240, 283, 105])
const == sum([soln2[i]*coeff.column(i) for i in range(n)])
```

```
True
```

```
(soln1 - soln2) in nsp
```

```
True
```

Section SS Spanning Sets

Sage SS Spanning Sets

Spanning Sets A strength of Sage is the ability to create infinite sets, such as the span of a set of vectors, from finite descriptions. In other words, we can take a finite set with just a handful of vectors and Sage will create the set that is the span of these vectors, which is an infinite set. Here we will show you how to do this, and show how you can use the results. The key command is the vector space method `.span()`.

```
V = QQ^4
v1 = vector(QQ, [1,1,2,-1])
v2 = vector(QQ, [2,3,5,-4])
W = V.span([v1, v2])
W
```

```
Vector space of degree 4 and dimension 2 over Rational Field
Basis matrix:
[ 1  0  1  1]
[ 0  1  1 -2]
```

```
x = 2*v1 + (-3)*v2
x
```

```
(-4, -7, -11, 10)
```

```
x in W
```

```
True
```

```
y = vector(QQ, [3, -1, 2, 2])
y in W
```

```
False
```