Math 317: Computer Lab 3

NAME:

Instructions.

- This assignment must be completed in the Carver 449 computer lab by 2pm on

  Friday March 25.

  Late submissions will not be accepted.

- When you have finished working or it is 2pm (whichever comes first):
  1. **Ask the professor to check your work.**
  2. Save your worksheet and name it Lab03.sws.
  3. **Submit your Lab03.sws file on Blackboard**.
  4. Stop your Sage worksheet (Action → Save and quit worksheet; or use Stop button).
  5. Sign out of your Sage account and log off the computer.

Part 1: Sage recap/review

The purpose of the first part of this lab is simply to recapitulate some of the things we learned about Sage in previous computer lab assignments, in case you are a bit rusty. Although this part will not be graded, you are strongly encouraged to try out all of the examples yourself (by entering them in a Sage worksheet). You need to be comfortable with the commands covered in Sections 2 and 3 in order to complete the second part of the lab.

1 Number systems

Sage knows a wide variety of sets of numbers. These are known as "rings" or "fields," but we may call them "number fields" or "scalar fields." (We need not worry about the differences for now.) Examples include the integers and the rational numbers—these sets are denoted by $\mathbb{Z}$ and $\mathbb{Q}$, respectively—as well as the real and complex number fields, denoted $\mathbb{R}$ and $\mathbb{C}$, respectively. Sage has special notation for each of these sets of numbers, and these are listed in the table below:

Set	**Description**	**Standard notation**	**Sage notation**
Integers	$\{\ldots, -1, 0, 1, 2, \ldots\}$	$\mathbb{Z}$	`ZZ`
Rational numbers	$\{\frac{p}{q} : p, q \in \mathbb{Z}, q \neq 0\}$	$\mathbb{Q}$	`QQ`
Real numbers	$(-\infty, \infty)$	$\mathbb{R}$	`RR`
Complex numbers	$\{a + ib : a, b \in \mathbb{R}\}$	$\mathbb{C}$	`CC`

Thus, `RR` denotes (Sage's representation of) the set of real numbers (up to "reasonable" precision), and `CC` is (Sage's representation of) the complex numbers (up to "reasonable" precision). In any computer system, there are complications surrounding the inability of digital arithmetic to accurately represent all real (or complex) numbers. In contrast, Sage can represent rational numbers exactly as the quotient of two (perhaps very large) integers. So, for now at least, we will use `QQ` as our main number system so we can concentrate on understanding the key concepts and ignore the fact that computers have trouble faithfully representing real and complex numbers.

2 Matrices in Sage

Sage is largely "object-oriented," which means many functions are associated with a specific "object" and are invoked using the "dot" notation. For example a matrix `A` is represented in Sage as a "matrix object," and the command `A.nrows()` returns the number of rows of the matrix `A`.

```
sage: A = matrix(QQ, 2, 3, [[1,2,3],[4,5,6]])
sage: print A
  [1 2 3]
  [4 5 6]

sage: A.nrows(), A.ncols()
  (2, 3)

sage: A.base_ring()
  Rational Field

sage: A.parent()
  Full MatrixSpace of 2 by 3 dense matrices over Rational Field
```

3 Vectors in Sage

Vectors in Sage are built, manipulated and interrogated in much the same way as matrices. However as simple lists, they are simpler to construct and manipulate. Sage will print a vector across the screen, even if we wish to interpret it as a column vector. It will be delimited by parentheses ((,)) which allows us to distinguish a vector from a matrix with just one row, if we look carefully. The number of slots in a vector is not referred to in Sage as rows or columns, but rather by "degree." Here are some examples:

```
sage: v = vector(QQ, 4, [1, 1/2, 1/3, 1/4])
sage: print v
  (1, 1/2, 1/3, 1/4)

sage: v.degree()
  4

sage: v.parent()
  Vector space of dimension 4 over Rational Field
```

```
sage: v[2]
  1/3

sage: w = vector([1, 2, 3, 4, 5, 6])
sage: print w
  (1, 2, 3, 4, 5, 6)

sage: w.degree()
  6

sage: w.parent()
  Ambient free module of rank 6 over the principal ideal domain Integer Ring

sage: w[3]
  4
```

Notice that if you use commands like `.parent()` you will sometimes see references to "free modules." This is a technical generalization of the notion of a vector, which is beyond the scope of this course, so just mentally convert to vectors when you see this term.

The zero vector is super easy to build, but be sure to specify what number system your zero is from.

```
sage: z = zero_vector(QQ, 5)
sage: print z
  (0, 0, 0, 0, 0)
```

Exercise 1. Login to your Sage account, open a new worksheet and name it Lab03. Then try out all of the examples above. (Enter what appears after the `sage:` prompt in a worksheet cell and click evaluate ; alternatively, you can evaluate a worksheet cell with the keyboard shortcut shift + enter .) Also note, you can put multiple lines in a single worksheet cell and evaluate them all at once.

Part 2: Spanning Sets & the Four Fundamental Subspaces

4 Spanning Sets

One of Sage's strengths 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 learn how to do this for the example vector space $\mathbb{Q}^4 = \{(x_1, x_2, x_3, x_4) : x_i \in \mathbb{Q}\}$, which Sage denotes by `QQ^4`. The key command is the vector space method `.span()`.

Exercise 2. Continuing with the worksheet (named Lab03) that you created in Exercise 1 above, try out each example below.

```
sage: V = QQ^4
sage: v1 = vector(QQ, [1,1,2,-1])
sage: v2 = vector(QQ, [2,3,5,-4])
sage: W = V.span([v1, v2])
sage: print W
  Vector space of degree 4 and dimension 2 over Rational Field
  Basis matrix:
  [ 1 0 1 1]
  [ 0 1 1 -2]

sage: x = 2*v1 + (-3)*v2
sage: print x
  (-4, -7, -11, 10)

sage: x in W
  True

sage: y = vector(QQ, [3, -1, 2, 2])
sage: y in W
  False

sage: u = vector(QQ, [3, -1, 2, 5])
sage: u in W
  True

sage: W <= V
  True
```

Most of the above should be fairly self-explanatory, but a few comments are in order. The span, `W`, is constructed with the `.span()` method, which accepts a list of vectors and is called using the "dot" syntax (since it is a method of a vector space object). You can see the number system (Rational Field) and the number of entries in each vector (degree 4).

Notice that the span (and the `.span()` command) is a function that takes as input a finite set of vectors and generates as output the span of these vectors. As we know, the result in an infinite collection of vectors. Of course, Sage does not literally construct all of the vectors in the span at once, since it's impossible to fit all of them in the computers finite physical memory. However, Sage provides us with a means of accessing any (finite) number of vectors in the span, and, importantly, Sage can check whether any given vector belongs to the span.[1]

In the example above, we know the vector `x` will be in the span `W` since we built it as a linear combination of `v1` and `v2`. The vectors `y` and `u` are a bit more mysterious, but Sage can answer the membership question easily for both of these vectors as well. In fact, from your experience in class and previous computer labs, you already know how to use Sage to prove that `y` and `u` belong to `W`.

[1] If you have studied computer science or programming, then you have probably heard of recursion, streams, and the like, and you should not be surprised that a computer can *represent* and *compute with* infinite sets, despite the fact that a computer is a finite entity with finite amount of physical memory. For example, a "stream" is basically just an infinite list that gives us access to its elements as needed... as many as we need, the stream will generate for us... but we are not allowed to ask for the entire list at once!

Exercise 3.
If possible, use Sage to write `y` as a linear combination of the vectors `v1` and `v2`. Do the same for `u`. That is, find the coefficients c_1, c_2, d_1, d_2, so that $\mathbf{y} = c_1\mathbf{v}_1 + c_2\mathbf{v}_2$ and $\mathbf{u} = d_1\mathbf{v}_1 + d_2\mathbf{v}_2$. Write these coefficients in the blanks spaces provided below. If it's not possible, explain why not.

$$\mathbf{y} = \begin{bmatrix} 3 \\ -1 \\ 2 \\ 2 \end{bmatrix} = \_\_\_\_\mathbf{v}_1 + \_\_\_\_\mathbf{v}_2, \qquad \mathbf{u} = \begin{bmatrix} 3 \\ -1 \\ 2 \\ 5 \end{bmatrix} = \_\_\_\_\mathbf{v}_1 + \_\_\_\_\mathbf{v}_2.$$

5 The Four Fundamental Subspaces

5.1 The Null Space $\mathrm{N}(A)$

Sage can compute the null space of a matrix. Again, this may seem rather remarkable, since the null space is very often an infinite set! The null space command is quite powerful since, as we have learned, there is a lot of theory associated with the null space; e.g., if we know the null space of a given matrix, then we know a lot about (among other things) systems of linear equations associated with that matrix.

One way to get Sage to build the null space of a matrix is with the `.right_kernel()` command. We experiment with this command below and, to ensure that the presentation of results is consistent with our previous work and the textbook, we will always use the option `basis="pivot"`. (Also, for reasons that are not important right now, we must continue to define matrices over the rational numbers.)

```
sage: A = 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]])

sage: NA = A.right_kernel(basis="pivot")  # (name this whatever you want...
                                          # ...but NA seems reasonable)
sage: print NA
  Vector space of degree 7 and dimension 4 over Rational Field
  User basis matrix:
  [-4 1 0 0 0 0 0]
  [-2 0 -1 -2 1 0 0]
  [-1 0 3 6 0 1 0]
  [ 3 0 -5 -6 0 0 1]
```

(Stuff that appears after a hashtag symbol `#` is called a comment; Sage ignores all comments.)

We can test membership in `NA`,

```
sage: x = vector(QQ, [3, 0, -5, -6, 0, 0, 1])
sage: x in NA
```

```
  True

sage: vector(QQ, [-4, 1, -3, -2, 1, 1, 1]) in NA
  True     # (we needn't name a vector to test its membership in NA)

sage: vector(QQ, [1, 0, 0, 0, 0, 0, 2]) in NA
  False
```

We can also verify that `NA` is an infinite set,

```
sage: NA.is_finite()
  False
```

As we also know, *sometimes* the null space is finite. When does this occur?

Exercise 4. Build your own matrix (call it `F`) such that the following command returns `True`.

```
sage: F.right_kernel(basis="pivot").is_finite()
```

If you're having trouble with Exercise 4, here's a hint: Since the null space of `F` is finite, we can list its vectors. For example, if our matrix happens to have two columns, then we will observe

```
sage: F.right_kernel(basis="pivot").list()
  [(0, 0)]
```

5.2 The Column Space $\mathrm{C}(A)$

Using span, we can expand our techniques for checking the consistency of a linear system $A\mathbf{x} = \mathbf{b}$. Recall that the system is consistent if and only if $\mathbf{b}$ is a linear combination of the columns of A. So consistency of a system is equivalent to the membership of $\mathbf{b}$ in the span of the columns of A.

We make use of the matrix method `columns()` to get all of the columns into a list at once.

```
sage: A = matrix(QQ, [[33, -16, 10, -2],
                      [99, -47, 27, -7],
                      [78, -36, 17, -6],
                      [-9,   2,  3,  4]])
sage: column_list = A.columns()
```

Exercise 5. Let `b = vector(QQ, [-27, -77, -52, 5])`. Use the Sage commands `columns` and `(QQ^4).span` to verify that the system `Ax = b` is consistent.

[*Hint:* write a line in Sage that tests for membership of `b` in the column space of `A` and returns `True`.]

Alternatives

Although the above works perfectly well, Sage has its own column space command, called `.column_space()`. It's very convenient, so let's try it out.

```
sage: A = matrix(QQ, [[ 1,-1, 0],
                      [ 2, 3, 9],
                      [ 0, -3, -4],
                      [-1, 4, 8]])
sage: CA = A.column_space()
sage: CA == (QQ^4).span(A.columns())  # check we get the same answer either way
  True
```

That the equality test in the last line above returns `True` provides evidence that our means of computing the column space of `A` is equivalent to Sage's build-in `column_space()` command.

Now if, say, `b = vector(QQ, [-16, 43, -10, 126])`, then we can check that the system `Ax = b` is consistent as follows:

```
sage: b = vector(QQ, [-16, 43, -10, 126])
sage: b in A.column_space()
  True
```

Knowing that `b` is in the column space of `A`, we can now ask Sage to solve for `x` in the system `Ax = b`, without fear of producing an error.

```
sage: A.solve_right(b)
  (-46, -30, 25)
```

Where to go from here?

To gain more familiarity with Sage, continue exploring on your own. Here are some suggestions: construct a right-hand side vector for which the system above is inconsistent, then look up what Sage commands are available for least squares solutions and/or projections, and apply these to the example you just constructed to find the vector in the column space of A that is closest to your right-hand side vector.