**Doing Linear Algebra in Sage – Part 2 – Simple Matrix Calculations**

**Creating matrices**

**Setting the underlying field of numbers**

When you work with matrices (or many other things) in Sage you need to tell it what kinds of numbers you will be using. The basic choices are:

* **QQ** for rational numbers
* **RR** for real numbers
* **CC** for complex numbers.

In this course we will always work with RR (or sometimes QQ to avoid rounding).

Next we move to the discussion of linear algebra operations. This material is drawn from Section 2.6 of the Sage Tutorial.

**Setting the size of the matrix and entering its values**

Suppose we want to create the set of 3x3 matrices over **Q.**

The command is

sage:M = MatrixSpace(QQ,3)

You could have gotten the same result by typing

sage:M = MatrixSpace(QQ,3,3)

and, as you can guess, MatrixSpace(QQ,3,2) will give you 3x2 matrices (3 rows and 2 columns).

You are not limited to matrices with entries in QQ…. RR or CC works as expected.

Sage also has a matrix( ) function for creating matrices and has corresponding vector( ) function and VectorSpace(QQ, 7) objects. (Reference Manual Chapter 31).

Once we have created the space M we may enter a matrix in it. For example, (see Constructions Chapter 5 at http://sagemath.org/doc/html/const/index.html) we enter

sage: A=M([1, 2, 3, 4, 5, 6, 7, 8, 9])

 **NOTE the format: A = M( enter inside square brackets)**

 **You need both the parentheses and square brackets.**

If you now ask Sage for A, you will have the following:

sage: A

[1 2 3]

[4 5 6]

[7 8 9]

It is also possible to get this matrix A specifying the matrix space ‘on the fly’ or anonymously:

**Matrix arithmetic and basic manipulations:**

If you have defined two matrices A and B then 2\*A multiplies A by 2, A+B is the matrix sum, A\*B is the matrix product and A^3 will give you A\*A\*A.

Sage comes with the many built-in **functions** or **methods**.

NOTE: When you are using a function named trace() or something else you must have the **parentheses** at the end. Also note that there is a **period** between the name of the matrix (A in this example) and the name of the function.

When you look at the code for Sage you will see code like:

 B=A.inverse(); B

The first part of this finds the inverse for A and assigns it to B. The semicolon separates the two statements. The second statement (B) asks for B to be displayed.

Some of the most basic functions are:

A.trace()

A.determinant()

 (There are other functions which use different algorithms for this
 computation.)

A.transpose()

**Row reducing, inverses and solving systems of equations**

If you have defined sage: A = matrix(QQ, 3, [1,2,3,4,5,6,7,8,9]

we can ask Sage to give us the reduced row echelon form of A.

NOTE: You are using a function named echelon\_form() and you must have the parentheses at the end. Also note that there is a period between the name of the matrix (A in this example) and the name of the function.

sage: A.echelon\_form()

[1 0 -1]

[0 1 2]

[0 0 0]

The method A.echelonize() will replace A by its row-echelon form.

The method A.echelon\_classical() **returns** the eceholon form of A and
A.echelon\_in\_place\_classical() **replaces** A by its echelon form.

If you wish to solve a system of equations A**x**=**b**, where the matrix A and the vector **b** have been defined, you may do it in any one of the following ways:

 X = A.solve\_right(b)

 X = A\b

Here is what Sage’s matrix.py file says about solving equations, and an explanation:

def \_backslash\_(self, B):

 """

 Used to compute $A \\ B$, i.e., the backslash solver operator.

 EXAMPLES:

 sage: A = matrix(QQ, 3, [1,2,4,2,3,1,0,1,2])

 sage: B = matrix(QQ, 3, 2, [1,7,5, 2,1,3])

 sage: C = A.\_backslash\_(B); C

 [ -1 1]

 [13/5 -3/5]

 [-4/5 9/5]

 sage: A\*C == B

 True

 sage: A.\_backslash\_(B) == A \ B

 True

 sage: A.\_backslash\_(B) == A.solve\_right(B)

 True

 """

 return self.solve\_right(B)

First the matrix A is defined as a 3x3 matrix over the rationals.

1. 2 4)
2. 3 1)
3. 1 2)

Notice that if you give only one dimension for the matrix size, Sage assumes it is a square matrix. Notice also that the definition of A is embedded in the statement which specifies it as a 3x3 matrix.

Next B is defined as a 3x2 (3 rows, 2 columns) matrix over the rationals. B is

1. 7)
2. 2)

(1 3)

We can solve two systems of equations at once:

 A **x** = first column of B

 A **y** = second column of B

**X** gives the first column of C and **y** gives the second column of C.

You can verify that A**x** = (1 2 4) ( -1 ) = (1)

 (2 3 1) (13/5) (5)

 (0 1 2) ( -4/5) (1)

and the right hand side is the first column of B. Ditto for A**y** being the second column of B.

To repeat, there are 3 easy ways to solve AX=b:

X= A \ b

X= A.\_blackslash\_(b)

X=A. solve\_right(b)

For MATLAB/Octave users, this is the backslash operator in that platform.

**Sage also has a method to augment matrices:**

 A.augment(C)

For example, if A is a 3x3 matrix and I3 is the 3x3 identity matrix you might write

 B=A.augment(I3)

 C=B.echelon\_form()

This will give you an augmented matrix ( I3 | A-1 )

If you have a matrix A and vector b then C = A.augment(b) will give you the usual augmented matrix and you may then ask for C.echelon\_form() to get the usual row-reduced echelon form.

Not surprisingly, you may also partition a matrix. Here is Sage’s example (The first line is just to generate some entries):

EXAMPLE:

 sage: M = matrix(3, 4, range(12))

 sage: M.subdivide(1,2); M

 [ 0 1| 2 3]

 [-----+-----]

 [ 4 5| 6 7]

 [ 8 9|10 11]

 sage: M.subdivision(0,0)

 [0 1]

 sage: M.subdivision(0,1)

 [2 3]

 sage: M.subdivision(1,0)

 [4 5]

 [8 9]

Eigenvalues and vectors and characteristic polynomials

A.charpoly()

 Sage’s documentation in matrix.py says:

 EXAMPLES:

 sage: a = matrix(QQ, 2,2, [1,2,3,4]); a

 [1 2]

 [3 4]

 sage: a.characteristic\_polynomial('T')

 T^2 - 5\*T - 2

 Notice that you may specify what to use as your indeterminant (here ‘T’).

A.fcp() will find the characteristic polynomial of A and factor it. The determinant will be ‘x’ unless you specify otherwise – e.g. A.fcp(‘z’)