Coding Theory for Storage and Networks

Tutorial: Introduction of SageMath

This tutorial gives an introduction on *SageMath*(*Sage*, in short) with a focus on computations in finite fields and algebraic coding.

Usage on Your Own Computers

You can

- download Sage from `http://www.sagemath.org/download.html` and install it in your own computer;
- use the online *CoCalc (SageMathCloud)*.

Usage after Installation on Your Own Computers

How to start SageMath

If you want to start SageMath the first time:

1. Open a new Konsole (terminal)
2. Type (in the terminal) `sage` to start SageMath

3* *You might need the following extra step to set* `sage` *as the alias to start SageMath in terminal.*

   a) Type (in the terminal)

   ```
   gedit ~/.bash_alias
   ```

   b) The editor gedit should open (it can be empty or show some text). Copy and paste the following line into the editor

   ```
   alias sage81="/nas/ei/share/lnt/students/SageMath/sage-8.1-Ubuntu_16.04-x86_64/SageMath/sage"
   ```

   c) Save your changes and close the editor gedit.

   d) Restart the terminal or type (in the terminal):

   ```
   source ~/.bash_alias
   ```

How to run the prepared script

1. Create a folder where you want to store the file. E.g., by opening a new terminal and typing:

   ```
   mkdir ~/CTSN_Lab/Lab_0 -p
   ```

2. Download the `.sage` files from moodle and save it in the created folder (e.g., in `~/CTSN_Lab/Lab_0`)

3. Type in SageMath

   ```
   sage: load("~/CTSN_Lab/Lab_0/Sage_assignment.sage")
   ```

You will probably get the message: ‘SyntaxError: invalid syntax’. This is expected since some parts in the file are missing and it is your task to fill them.

There is a short video in Moodle (Link to Panopto) explaining how to run a Sage script in Terminal or Cocalc.

How to edit the prepared script:

1. Open the downloaded file with an editor (e.g. Atom etc.) or by typing in the terminal

   ```
   gedit ~/CTSN_Lab/Lab_0/SageTutorial.sage
   ```

2. Fill in the missing parts and save your changes. Then, run your script again.

Basic Programming

Sage is based on Python and the syntax is essentially the same.

Common Structures

The most common structures used for linear algebra are lists, vectors and matrices.

```
sage: Lst = [2,1,3] # list
sage: Vec = vector([6,5,4]) # vector
sage: Mat = matrix([[1,0],[2,1],[0,1]]) # matrix
```

Sage handles finite fields very well.

```
# Defining a finite integer field:
sage: F11 = GF(11)
# An extension field can be defined by
sage: F.<a> = GF(16, modulus=x^4+x+1)
# a is a primitive element of the field
```

Instead of an explicit polynomial you can use **modulus=‘primitive’**. The integer 11 or 16 gives the cardinality of the field and must be a prime or a power of a prime.

* For more examples, please see `SageExamples.sage`.

Useful Functions

As sage is based on python, it is class based. Each data type has its own functions. For example, type Lst. and press *Tab* to see the functions defined for lists.

Table 1 gives some useful functions

Typing **help(function_name)** or **function_name?** can access to the documentation of functions.

List	len(Lst)	# Get the length of Lst
	del Lst[*pos*]	# Delete the element at position *pos* from Lst
	.pop(*pos*)	# Delete the element at position *pos* from a list
	.insert(*pos*,*ele*)	# Insert *ele* into position *pos*
Vector	.list()	
	len(Vec) or Vec.length()	# Get the length of Vec
	.base_ring()	# Get the field of a vector
Matrix	matrix(F,n,m)	# Create an $n \times m$ all-zero matrix in F
	matrix.random(F,n,m)	
	matrix.identity(F,n,m)	
	.base_ring()	# Get the field of a matrix
	.nrows()	
	.ncols()	
	.rank()	
	.transpose()	
	.echelon_form()	
	.solve_right()	
	.apply_map(Φ)	# Apply the map Φ to all the enries
Finite Field	.primitive_element()	
	.polynomial or .modulus()	# Get the primitive polynomial of a finite field
	.list()	# Get a list of all element in a finite field
	.order()	# Get the cardinality of a finite field

Table 1: Table of useful functions

Task 1: Defining field elements and performing basic operations

1. Define a field $\mathbb{F}_{2^4}$ with the primitive element named a.
2. Define two elements **ele_1** $= a^2 + 1$ and **ele_2** $= a^2 + a$ in $\mathbb{F}_{2^4}$.
    - Find the logarithm for each element w.r.t. a.
    *Hint: The function **log(element,base)** works for elements in finite field.*
    - Check if the elements are primitive elements of the field.
    Hint: Check if the powers of the elements generate $\mathbb{F}_{2^4}/\{0\}$.

Task 2: Polynomials over finite fields

In Sage it is very easy to represent a polynomial $g(x) \in \mathbb{F}_{q^m}[x]$. x is defined as the variable of a polynomial. There are several possible ways to define x. We give 4 possibilities in *Sage_assignment.sage.*

1. Represent the following polynomial

$$f(x) = x^5 + \alpha^2 x^4 + (\alpha^3 + \alpha^2 + \alpha)x^3 + (\alpha^3 + \alpha^2 + \alpha)x^2 + \alpha^3 x + \alpha + 1 \quad \in \mathbb{F}_{2^4}[x]. \tag{1}$$

   Get the coefficients of the polynomial.

   *Hint: Use **.list()** or **.coefficients(sparse=false)**.*

2. Find the roots of $f(x)$.

   *Hint: Use **.roots()**.*