MAT 305: Mathematical Computing

Introduction to Sage

John Perry

University of Southern Mississippi

Spring 2016

Outline

1. What is Sage?
2. Getting started with Sage
3. Using computer memory
4. “Algebra”
5. Summary

Outline

1. What is Sage?
2. Getting started with Sage
3. Using computer memory
4. "Algebra"
5. Summary

Sage?

- Software for Algebra and Geometry Exploration
- Computer Algebra System "started" by William Stein

- Access to other CASs
  - Calculus: Maxima, SymPy, ...
  - Linear Algebra: M4RI, Linbox, PARI, ...
  - Commutative Algebra: SINGULAR, Macaulay, ...
  - Group theory: GAP, ...
  - etc.

Why Sage?

“Free” software

Why Sage?

“Free” software

- “Free as in beer”:
    - no cost to download
    - no cost to copy
    - no cost to upgrade

Why Sage?

"Free" software

- "Free as in beer":
  - no cost to download
  - no cost to copy
  - no cost to upgrade
- "Free as in speech":
  - no secret algorithms
  - can study implementation
  - can correct, improve, contribute

Analogy: “Free” Mathematics

Theorem

There are infinitely many primes.

Proof.

- Consider finite list of primes, $q_1, q_2, \ldots, q_n$.
- Let $p = q_1 q_2 \cdots q_n + 1$.
- **Fact:** since $p \neq 1$, divisible by at least one prime
- By Division Theorem, p not divisible by any q_i (remainder 1, not 0).
- p divisible by unlisted prime q_{n+1}!
- $\therefore$ no finite list, lists all primes.

□

Analogy: “Secret” mathematics

Theorem

There are infinitely many primes.

Proof.

“I have discovered a truly marvelous proof of this, which this margin is too narrow to contain.”[†] □

[†] Real quote, different theorem.

Analogy: "Proprietary" mathematics

Theorem

There are infinitely many primes.

Proof.

Trade Secret. □

But I prefer M—!

- Fine, buy your own copy
  - good reasons exist
  - student discount available
  - I will tell you the equivalent commands
- Be warned:
  - future versions not free
  - bug fixes not free
  - after you graduate, pay full price
  - not always backwards compatible (neither is Sage, but Sage is free)

Python

- "Sage" built on/with Python
  - interface between Sage and user

- Not all *components* of Sage in Python:
  - Maxima: LISP
  - SINGULAR: C/C++

Advantages of Python

- Modern
  - facilities for object-oriented, functional programming
- Wide distribution, usage
  - many employers use it
    (doing well in this class makes you more attractive!)
- Flexible
  - many good packages enhance it
- Can compile for efficiency using Cython

Python $\neq$ Sage

- Some Python commands don’t work in worksheet mode
    - `input()`
- Sage commands do not work in plain Python

Outline

1. What is Sage?
2. Getting started with Sage
3. Using computer memory
4. "Algebra"
5. Summary

How to get Sage

- Best: SageMathCloud
- `Download`, `install` to your computer
    - can tinker with/break the source code
    - Windows? need LiveCD or VirtualBox player: `www.virtualbox.org/wiki/Downloads`
    - ask nicely, & I might give you a DVD with Sage for Windows, Mac, Linux

First steps in Sage

- Log in to SageMathCloud
- Start a new project
  - name it “First Sage Assignment”
  - select type “SageMath Worksheet”
  - visit “Settings”, click “Project usage and quotas”
  - click “Network access” and “Member hosting”, then “Submit changes”
  - Now return to “First Sage Assignment” (near top)
- *If you like* (not always recommended)
  - Click “Modes”, then “Typeset output”

Some *symbols* of *symbolic* computation

- Variable's value is *indeterminate*
  - stands for arbitrary value
  - different from traditional languages, numeric packages
- only x pre-defined
  - undefined symbols give errors
- Need more? use `var()`
  - `var('y')` defines y
  - `var('a b c d')` defines a, b, c, d
- Use undefined variable?

```
sage: x+y+z
...
NameError: name 'z' is not defined
```

Arithmetic

operation	sage equivalent
add x, y	`x + y`
subtract y from x	`x - y`
multiply x, y	`x * y`
divide x by y	`x / y`
raise x to the yth power	`x ** y` or `x ^ y`

Arithmetic

operation	sage equivalent
add x, y	`x + y`
subtract y from x	`x - y`
multiply x, y	`x * y`
divide x by y	`x / y`
raise x to the yth power	`x ** y` or `x ^ y`

- Do not omit muliplication symbol
  - `2*x` $\longrightarrow 2x$
  - `2x` $\longrightarrow$ `SyntaxError: invalid syntax`
  - possible, but dangerous, to get around this using `implicit_multiplication(True)`
- Do not neglect parentheses
  - `e**(2*x)` $\neq$ `e**2*x`
- Prefer `**` to `^` for various sordid reasons (scripting)

Example

- Sage simplifies (of course)

```
sage: 5 + 3
8
sage: (x + 3*x**2) - (2*x - x**2)
4*x^2 - x
```

Transcendental numbers, functions

number	sage symbol
e	`e`
π	`pi`

operation	sage equivalent
e^x	`e**x`
$\ln x$	`ln(x)`
$\sin x$, $\cos x$, etc.	`sin(x)`, `cos(x)`, etc.

Transcendental numbers, functions

number	sage symbol
e	`e`
π	`pi`

operation	sage equivalent
e^x	`e**x`
$\ln x$	`ln(x)`
$\sin x$, $\cos x$, etc.	`sin(x)`, `cos(x)`, etc.

- `log(x)` $= \ln x \neq \log_{10} x$

Some useful operations

operation	sage equivalent
factor *expr*	`factor(`*expr*`)`
simplify *expr*	`simplify(`*expr*`)`
expand *expr*	`expand(`*expr*`)`
round *expr* to n decimal places	`round(`*expr*`,` n`)`

Examples

- Some expressions simplify automatically; many need hints

```
sage:  (x**2 - 1) / (x - 1)
(x^2 - 1)/(x - 1)
sage:  (factor(x**2 - 1)) / (x - 1)
x + 1
```

  (good reason this isn't automatic: what?)

- Expand $(x-1)(x^3+x^2+x+1)$

```
sage:  expand((x-1)*(x**3+x**2+x+1))
x^4 - 1
```

- Round e to 5 decimal places

```
sage:  round(e,5)
2.71828
```

Getting help

- Online Sage documentation (tutorial, manual, etc.) at `http://www.sagemath.org/doc/`
- These notes: `www.math.usm.edu/perry/mat305ssyy/` (ssyy? semester and year: `sp13`, `sp14`, `sm14`, ...)
- In-Sage help: command, question mark, `<Enter>`

```
sage: round?
```

[output omitted]

- Email: `john.perry@usm.edu`

Outline

1. What is Sage?
2. Getting started with Sage
3. Using computer memory
4. “Algebra”
5. Summary

Expressions

- Use a computer's memory by defining *expressions* with the *assignment symbol* =

```
sage: f = x**2 - 1
```

Sage does not answer when you define an expression

- Expressions are remembered until you terminate Sage

```
sage: f
x^2 - 1
```

- You can remember a "structure" as well as an expression

```
sage: R = GF(7) # I'll tell you what
this is later
```

Valid names

Names for expressions ("*identifiers*") can

- contain letters (A–Z), digits (0–9), or the underscore (_) *but*
- must begin with a letter or the underscore *and*
- may not contain other character (space, tab, !@#$%^, etc.)

Using expressions

- Manipulate expression in the same way as the mathematical object it represents

```
sage: factor(f)
(x - 1)*(x + 1)
sage: f - 3
x^2 - 4
```

- Avoid repeating computations: substitute!

```
sage: f(x=3)
8
sage: f(x=-1)
0
sage: f(x=4)
15
```

Alternate method of substitution

Sometimes you should use the **dictionary** method of substitution. An example would be when an identifier stands for a variable.

```
sage:  f = x**2 + y**2
sage:  f(x=3)
9 + y^2
sage:  f({x:3})
9 + y^2
```

This also means replace x by 3 in f

Alternate method of substitution

Sometimes you should use the **dictionary** method of substitution. An example would be when an identifier stands for a variable.

```
sage:  f = x**2 + y**2
sage:  f(x=3)
9 + y^2
sage:  f({x:3})
9 + y^2
sage:  z = x
sage:  f(z=3)
x^2 + y^2
```

This also means replace x by 3 in f

Here we let z stand in place of x

We want to replace x by 3, but…

Alternate method of substitution

Sometimes you should use the **dictionary** method of substitution. An example would be when an identifier stands for a variable.

```
sage:  f = x**2 + y**2
sage:  f(x=3)
9 + y^2
sage:  f({x:3})
9 + y^2
sage:  z = x
sage:  f(z=3)
x^2 + y^2
sage:  f({z:3})
9 + y^2
```

This also means replace x by 3 in f

Here we let z stand in place of x

We want to replace x by 3, but…

This works where `f(z=3)` did not

Expressions as functions

Define function using natural notation

```
sage: f(x) = x**2
sage: f(2)
4
sage: f
x |--> x^2
```

Expressions as functions

Define function using natural notation

```
sage:  f(x) = x**2
sage:  f(2)
4
sage:  f
x |--> x^2
```

Automatically defines variables!

```
sage:  f(w,z) = 4*w**2-4*z**2
sage:  f(3,2)
20
sage:  f(1,z)/z
-4*(z**2 - 1)/z
sage:  f(3,2)/z
20/z
```

Expressions as functions

Define function using natural notation

```
sage:  f(x) = x**2
sage:  f(2)
4
sage:  f
x |--> x^2
```

Functions really expressions

```
sage:  factor(f)
4*(w - z)*(w + z)
sage:  type(f)
<type 'sage.symbolic.expression.Expression'>
```

Outline

1. What is Sage?
2. Getting started with Sage
3. Using computer memory
4. “Algebra”
5. Summary

Structure

- Mathematical operations take place in well-defined structures
- In this class, we primarily use rings and fields

"Ring"?!? *"Field"?!?*

Ring: ordinary arithmetic guaranteed, *except* division

- $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ (integers)
- $\mathbb{Q} = \{a/b : a, b \in \mathbb{Z}, b \neq 0\}$ (rationals, "quotients")
- $\mathbb{R} = \left\{\pm a_0 a_1 \ldots a_m . a_{m+1} a_{m+1} \ldots\right\}$ (reals, "measurements")
- $\mathbb{C} = \left\{a + bi : a, b \in \mathbb{R}, i^2 = -1\right\}$ (complex, "complete")

"Ring"?!? *"Field"?!?*

Ring: ordinary arithmetic guaranteed, *except* division

- $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ (integers)
- $\mathbb{Q} = \{a/b \ : \ a, b \in \mathbb{Z}, b \neq 0\}$ (rationals, "quotients")
- $\mathbb{R} = \left\{\pm a_0 a_1 \ldots a_m . a_{m+1} a_{m+1} \ldots\right\}$ (reals, "measurements")
- $\mathbb{C} = \left\{a + bi : a, b \in \mathbb{R}, i^2 = -1\right\}$ (complex, "complete")

Field: division guaranteed, too

- $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$
- *not* $\mathbb{Z}$
- don't worry about 0

“Ring”?!? *“Field”?!?*

Ring: ordinary arithmetic guaranteed, *except* division

- $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ (integers)
- $\mathbb{Q} = \{a/b : a, b \in \mathbb{Z}, b \neq 0\}$ (rationals, “quotients”)
- $\mathbb{R} = \{\pm a_0 a_1 \ldots a_m . a_{m+1} a_{m+1} \ldots\}$ (reals, “measurements”)
- $\mathbb{C} = \{a + bi : a, b \in \mathbb{R}, i^2 = -1\}$ (complex, “complete”)

Field: division guaranteed, too

- $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$
- *not* $\mathbb{Z}$
- don’t worry about 0

(Intuitive descriptions, not formal definitions)

Sage notation for common rings

- Integers: `ZZ` $\mathbb{Z}$
- Rationals: `QQ` $\mathbb{Q}$
- Reals: `RR` $\mathbb{R}$
  (Sage *approximates* w/53 bits precision)
- Complex: `CC` $\mathbb{C}$
  (Sage *approximates* w/53 bits precision)

Advanced rings

- Algebraic reals: `AA` $\overline{\mathbb{Q}}$
  (algebraic closure of $\mathbb{Q}$)

- Finite fields: `GF`(n) $\mathbb{Z}_n$
  (n power of prime; if not first power, specify string as name for generator)

- Finite rings: `ZZ.quo`(n) $\mathbb{Z}_p$
  (n must be an integer)

- Symbolic: `SR`
  (can use expressions with symbols as entries)

Using expressions

We sometimes work in uncommon rings

```
sage: a, b = R(4), R(6)
sage: a + b
3
sage: 4 + 6
10
sage: 2*a + 3*b
5
sage: a**(-1)
2
```

Recall R is $\mathbb{Z}_7$

$4+6=10$, remainder by 7

ordinary arithmetic still holds

$2\times4+3\times6=26$, remainder by 7

$4\times2=8$, remainder by 7 is 1

Outline

1 What is Sage?

2 Getting started with Sage

3 Using computer memory

4 "Algebra"

5 Summary

Summary

- Basic, intuitive facilities for arithmetic
- Create variables to your heart’s content
- Define expressions to avoid repeating computations