What is Sage?
"Computer algebra"
Why Sage?
Sage and Python

Getting started with Sage

Using computer memory

Summary

MAT 305: Mathematical Computing

Introduction to Sage

John Perry

University of Southern Mississippi

Spring 2019

Outline

1. What is Sage?
    - "Computer algebra"
    - Why Sage?
    - Sage and Python
2. Getting started with Sage
3. Using computer memory
4. Summary

Outline

1. What is Sage?
   - "Computer algebra"
   - Why Sage?
   - Sage and Python
2. Getting started with Sage
3. Using computer memory
4. Summary

Outline

1. What is Sage?
   - "Computer algebra"
   - Why Sage?
   - Sage and Python
2. Getting started with Sage
3. Using computer memory
4. Summary

Sage?

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

"Computer algebra system"?

Numerical computing

goal approximate computation, "accurate estimation"

Symbolic computing

goal exact computation

"Computer algebra system"?

Numerical computing

goal approximate computation, "accurate estimation"

tools floating-point numbers, vectors, matrices

Symbolic computing

goal exact computation

tools exact numbers, sets, abstract structures

"Computer algebra system"?

Numerical computing

goal approximate computation, "accurate estimation"

tools floating-point numbers, vectors, matrices

challenge overflow

- division by a small number

Symbolic computing

goal exact computation

tools exact numbers, sets, abstract structures

challenge complexity

- adding many fractions

“Computer algebra system”?

Numerical computing

goal approximate computation, “accurate estimation”

tools floating-point numbers, vectors, matrices

challenge overflow

- division by a small number

analogy telling you an “accurate” lie

Symbolic computing

goal exact computation

tools exact numbers, sets, abstract structures

challenge complexity

- adding many fractions

analogy telling you the truth… once we figure it out…

Pros & cons: symbolic

$$\frac{1}{2}+\frac{1}{3}+\frac{1}{5}+\frac{1}{7}=\frac{247}{210}$$

- summands: two digits each, *but*
- sum: 6 digits
- imagine this done thousands or millions of times

"Expression swell"

Pros & cons: symbolic

$$\frac{1}{2}+\frac{1}{3}+\frac{1}{5}+\frac{1}{7}=\frac{247}{210}$$

- summands: two digits each, *but*
- sum: 6 digits
- imagine this done thousands or millions of times

"Expression swell"

```
sage: 1 + 10^(-5) - 1
1/100000
```

...not bad!

Pros & cons: numeric

$$0.5000 + 0.3333 + 0.2000 + 0.1429 \approx 1.176$$

- start and end with four digits, *but*
- small loss in precision

Pros & cons: numeric

$$0.5000 + 0.3333 + 0.2000 + 0.1429 \approx 1.176$$

- start and end with four digits, *but*
- small loss in precision

```
sage: 1.0 + 10.0^(-5.0) - 1.0
0.0000100000000000655
sage: 1.0 + 10.0^(-15.0) - 1.0
1.11022302462516e-15
sage: 1.0 + 10.0^(-20.0) - 1.0
0.000000000000000
```

More cons: numeric

$$\begin{pmatrix} \frac{1001}{2001} & -\frac{1000}{2001} \\ -\frac{1000}{2001} & \frac{1001}{2001} \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

but

$$\begin{pmatrix} \frac{1001}{2001} & -\frac{1000}{2001} \\ -\frac{1000}{2001} & \frac{1001}{2001} \end{pmatrix} \begin{pmatrix} 1.1 \\ -0.9 \end{pmatrix} = \begin{pmatrix} 201.1 \\ 199.1 \end{pmatrix}$$

- small change in input, *but*
- large change in output
- consider the effect of roundoff error…

More cons: numeric

$$\begin{pmatrix} \frac{1001}{2001} & -\frac{1000}{2001} \\ -\frac{1000}{2001} & \frac{1001}{2001} \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

but

$$\begin{pmatrix} \frac{1001}{2001} & -\frac{1000}{2001} \\ -\frac{1000}{2001} & \frac{1001}{2001} \end{pmatrix} \begin{pmatrix} 1.1 \\ -0.9 \end{pmatrix} = \begin{pmatrix} 201.1 \\ 199.1 \end{pmatrix}$$

- small change in input, *but*
- large change in output
- consider the effect of roundoff error…

"It makes me nervous to fly an airplane since I know they are designed using floating-point arithmetic."

— Alston Householder

Outline

1. What is Sage?
   - "Computer algebra"
   - Why Sage?
   - Sage and Python
2. Getting started with Sage
3. Using computer memory
4. Summary

Practical reasons

- Free
- Cutting edge
- Access to other CAS's
  - Calculus: Maxima, SymPy, ...
  - Linear Algebra: M4RI, Linbox, PARI, ...
  - Commutative Algebra: SINGULAR, Macaulay, ...
  - Group theory: GAP, ...
  - etc.

Philosophical reasons

"Free" software

Philosophical reasons

"Free" software

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

Philosophical reasons

"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 (Euclid)

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
- p not divisible by any q_i (remainder 1, not 0).
- p must be divisible by an unlisted prime
- $\therefore$ no finite list, lists all primes.

□

Analogy: "Secret" mathematics

Theorem (Fermat)

If $n > 2$, the equation $a^n + b^n = c^n$ has no solution with integers $a, b, c \geq 1$.

Proof.

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

[†] Real quote. (to be fair: in private notes, not letter, article)

Analogy: "Proprietary" mathematics

Theorem (Mersenne)

The number

$$2^n - 1$$

is prime for $n = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257.$

Proof.

□

Analogy: "Proprietary" mathematics

Theorem (Mersenne)

The number

$$2^n - 1$$

is prime for $n = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257$.

Proof.

Trade Secret.[†] □

Analogy: "Proprietary" mathematics

Theorem (Mersenne)

The number

$$2^n - 1$$

is prime for $n = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257.$

Proof.

□

†In fact, the "theorem" is false.

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)

Outline

1. What is Sage?
   - "Computer algebra"
   - Why Sage?
   - Sage and Python
2. Getting started with Sage
3. Using computer memory
4. Summary

Python

- Major computer language
  - easy to use
  - elegantly designed
  - unlike *Cough* the Coding Convention they presCribe in CSC 101/L
- Modern
  - facilities for object-oriented, functional programming
- Wide distribution, usage
  - many employers use it
    (doing well in this class makes you more attractive!)
  - I checked 4 websites that listed top in-demand languages & salaries
- Flexible
  - many good packages enhance it
- Can compile for efficiency using Cython

Kinds of computer languages

- Interpreted
  - BASIC, Python, Perl
  - computer reads source, repeats following:
    - translate symbols until full command formed
    - execute command
  - no translation saved

Kinds of computer languages

- Interpreted
  - BASIC, Python, Perl
  - computer reads source, repeats following:
    - translate symbols until full command formed
    - execute command
  - no translation saved
- Compiled
  - C/C++, FORTRAN, Go
  - reads source, translates and saves **machine** code
  - translation works on same architecture (OS, CPU, ...)

Kinds of computer languages

- Interpreted
  - BASIC, Python, Perl
  - computer reads source, repeats following:
    - translate symbols until full command formed
    - execute command
  - no translation saved
- Compiled
  - C/C++, FORTRAN, Go
  - reads source, translates and saves **machine** code
  - translation works on same architecture (OS, CPU, ...)
- Mixed ("bytecode")
  - C#(.NET), Java
  - reads source, translate into **byte**code, saves
  - translation works in "virtual machine" (JVM, .Net, ...)

Sage and Python

- "Sage" built on/with Python
  - interface between Sage and user
- Not all *components* of Sage in Python:
  - Maxima: LISP
  - SINGULAR: C/C++
  - "kernel" "compiled" for efficiency's sake

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?
   - "Computer algebra"
   - Why Sage?
   - Sage and Python
2. Getting started with Sage
3. Using computer memory
4. Summary

How to get Sage

- Best: links on the website
- Alternate: SageMathCloud
- Linux users:[1] `sudo dnf install sagemath`
- People with delusions of grandeur: `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

[1] Because Fedora is the One True Linux. People who want to `apt-get` stuff can go ask Debian or Ubuntu users how to do it.

First steps in Sage

- Log in to Bagheera (use links on class web page, have I mentioned that yet?)
- Start a new worksheet
    - rename it "First Sage Assignment"
- *If you like* (not always recommended)
    - Click "Typeset"

Working with variables

Variable: symbol that represents another value

Example

```
sage:  a = 7
```

Until you change it, a represents 7

Symbols of symbolic computation

Indeterminate: symbol with no specific value ("unknown")

- special kind of variable
- x pre-defined
  - if value assigned, no longer indeterminate
- 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 constants, 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 constants, 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/old_classes/mat305ssyy/` (ssyy? semester and year: `sp13`, `sp14`, `sm14`, ...)
- Textbook: www.math.usm.edu/dont_panic
- In-Sage help: command, question mark, `<Enter>`

```
sage: round?
```

[output omitted]

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

Outline

1. What is Sage?
   - "Computer algebra"
   - Why Sage?
   - Sage and Python
2. Getting started with Sage
3. Using computer memory
4. Summary

Expressions

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

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

  Sage does not answer when you define an expression

- Expressions remembered until you terminate Sage

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

- Can remember "structures" as well as expressions

```
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 just like the 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?
   - "Computer algebra"
   - Why Sage?
   - Sage and Python
2. Getting started with Sage
3. Using computer memory
4. Summary

Summary

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