

MAT 305: Mathematical Computing

Solving equations in Sage

John Perry

University of Southern Mississippi

Fall 2013

Outline

① Exact solutions to equations

Exact solutions

Extracting solutions

Systems of linear equations

② Approximate solutions to equations

③ Summary

Exact solutions
to equations

Exact solutions

Extracting solutions

Systems of linear
equations

Approximate
solutions to
equations

Summary

Outline

1 Exact solutions to equations

Exact solutions

Extracting solutions

Systems of linear equations

2 Approximate solutions to equations

3 Summary

Exact solutions

- Many equations can be solved without rounding
 - *exact solutions*
 - Solving by radicals: old, important problem
 - Niels Abel, Evariste Galois, Joseph Lagrange, Paolo Ruffini, ...
 - Special methods
- Others require approximate solutions

The `solve()` command

`solve(eqs, vars)` where

- *eqs* is a list of equations
 - an equation contains the symbol `==`, “equals”
 - the symbol `=` means “assign”
- *vars* is a list of variables to solve for
 - variables not listed are treated as constants
 - if only one variable, do not use list
- returns a list of solutions *if* Sage can solve *eqs* exactly

$= \neq ==$

FACT OF PYTHON

- `=` (single)
 - assignment of a value to a symbol
 - `f = x**2 - 4` assigns the value $x^2 - 4$ to `f`
 - “let $f = x^2 - 4$ ”
- `==` (double)
 - two quantities are equal
 - `16==4**2` is *true*
 - `16==5**2` is *false*
 - `16==x**2` is *conditional*; it depends on the value of `x`
- Confuse the two? *naughty user*

Example

```
sage: 16==4**2
```

```
True
```

```
sage: 16==5**2
```

```
False
```

```
sage: 16==x**2
```

```
16 == x^2
```

(cannot simplify the expression)

Univariate polynomials

```
sage: solve([3*x+1==4*(x-2)+3],x)
[x == 6]
```

```
sage: solve([x**2==-1],x)
[x == -I, x == I]
```

(**I** represents $\sqrt{-1}$)

```
sage: solve([x**5+2*x+1==0],x)
[0 == x^5 + 2*x + 1]
```

(Sage cannot find exact solution)

Unknown constants

```
sage: var('a b c')  
(a, b, c)
```

```
sage: solve([a*x**2+b*x+c==0],x)  
[x == -1/2*(b + sqrt(-4*a*c + b^2))/a,  
 x == -1/2*(b - sqrt(-4*a*c + b^2))/a]
```

(quadratic formula!)

Copying solutions not always a good idea

Exact solutions
to equations

Exact solutions

Extracting solutions

Systems of linear
equations

Approximate
solutions to
equations

Summary

```
sage: solve([3*x**3-4*x==7],x)
[x == -1/2*(1/54*sqrt(3713) + 7/6)^(1/3)*(I*sqrt(3)
+ 1) + 1/9*(2*I*sqrt(3) - 2)/(1/54*sqrt(3713) +
7/6)^(1/3), x == -1/2*(1/54*sqrt(3713) +
7/6)^(1/3)*(-I*sqrt(3) + 1) + 1/9*(-2*I*sqrt(3) -
2)/(1/54*sqrt(3713) + 7/6)^(1/3), x ==
(1/54*sqrt(3713) + 7/6)^(1/3) + 4/9/(1/54*sqrt(3713)
+ 7/6)^(1/3)]
```

ouch!

Assign, use []

To extract values from solutions, assign and use []

Example

```
sage: sols = solve([x**4-1==0], x)
```

```
sage: sols
```

```
[x == I, x == -1, x == -I, x == 1]
```

```
sage: sols[0]
```

```
x == I
```

```
sage: sols[1]
```

```
x == -1
```

```
sage: sols[3]
```

```
x == 1
```

[] ranges from 0 to (*length*-1)

FACT OF PYTHON

Suppose L is a list or tuple of length n

- first element: $L[0]$
- last element: $L[n-1]$
- $L[n]$? *naughty user*

Example

```
sage: sols = solve([x**4-1==0],x)
```

```
sage: sols
```

```
[x == I, x == -1, x == -I, x == 1]
```

```
sage: sols[4]
```

... *output cut* ...

```
IndexError: list index out of range
```

But I want only the number...!

- Every equation has a right hand side
- Use `.rhs()` command
 - “dot” command: *append* to object

Example

Exact solutions
to equations

Exact solutions

Extracting solutions

Systems of linear
equations

Approximate
solutions to
equations

Summary

```
sage: eq = 4*x**2 - 3*x + 1 == 0
```

```
sage: sols = solve([eq],x)
```

```
sage: len(sols)
```

2

(len() gives length of a collection)

```
sage: x1 = sols[0]
```

```
sage: x1
```

```
x == -1/8*I*sqrt(7) + 3/8
```

(oops! want only solution)

```
sage: x1 = sols[0].rhs()
```

```
sage: x1
```

```
-1/8*I*sqrt(7) + 3/8
```

(better)

What if I get complex solutions?

- 1 `.real_part()`, `imag_part()` separate real, imaginary parts
- 2 Can `round()` if desired

```
sage: sols = solve([x**5-3==0], x)
```

```
sage: sols
```

```
[x == 3^(1/5)*e^(2/5*I*pi), x ==
```

```
3^(1/5)*e^(4/5*I*pi), x == 3^(1/5)*e^(-4/5*I*pi), x  
== 3^(1/5)*e^(-2/5*I*pi), x == 3^(1/5)]
```

```
sage: sols[0].rhs().real_part()
```

```
1/4*sqrt(5)*3^(1/5) - 1/4*3^(1/5)
```

```
sage: sols[0].rhs().imag_part()
```

```
3^(1/5)*sin(2/5*pi)
```

```
sage: a, b = sols[0].rhs().real_part(),
```

```
sols[0].rhs().imag_part()
```

```
sage: round(a,5), round(b, 5)
```

```
(0.38495, 1.18476)
```

Let's test solutions

Extract second solution; substitute into equation

```
sage: x2 = sols[1].rhs()
```

```
sage: x2  
1/8*I*sqrt(7) + 3/8
```

```
sage: eq(x=x2)  
4*(1/8*I*sqrt(7) + 3/8)^2  
- 3/8*I*sqrt(7) - 1/8 == 0
```

(need to expand product)

```
sage: expand(eq(x=x2))  
0 == 0
```


Systems of linear equations

Exact solutions to equations

Exact solutions

Extracting solutions

Systems of linear
equations

Approximate solutions to equations

Summary

- system of linear, multivariate equations
- can always be solved *exactly*
- zero, one, or infinitely many solutions
- solution is a list of solutions

No solution

```
sage: var('y')  
(y)
```

```
sage: solve([x + y == 1,  
            x + y == 0],  
            [x,y])
```

... output cut...

```
[]
```

One solution

```
sage: var('z')
```

```
(z)
```

```
sage: solve([3*x - 4*y + z == 1,  
            2*x - 3*y + 4*z == 2,  
            7*x + 10*y - 39*z == 1],  
            [x,y,z])
```

```
[[x == (3/2), y == 1, z == (1/2)]]
```

Infinitely many solutions

```
sage: solve([3*x - 4*y + z == 1,  
            2*x - 3*y + 4*z == 2,  
            -6*x + 8*y - 2*z == -2],  
            [x,y,z])  
[[x == 13*r1 - 5, y == 10*r1 - 4, z == r1]]
```

r_1 ?!?! What is r_1 ?

r_1 is a *parameter* that can take infinitely many values

$$[[x == 13*r_1 - 5, y == 10*r_1 - 4, z == r_1]]$$

corresponds to

$$x = 13t - 5, \quad y = 10t - 4, \quad z = t.$$

Example

$$t = 0?$$

- $x = -5, y = -4, z = 0$
- Substitute into system:

$$3(-5) - 4(-4) + 0 = 1$$

$$2(-5) - 3(-4) + 4(0) = 2$$

$$-6(-5) + 8(-4) - 2(0) = -2.$$

Extract and test

```
sage: eq1 = 3*x - 4*y + z == 1
sage: eq2 = 2*x - 3*y + 4*z == 2
sage: eq3 = -6*x + 8*y - 2*z == -2
sage: sols = solve([eq1, eq2, eq3], [x,y,z])
```

sols is a list of lists...

```
sage: sol1 = sols[0]
sage: x1 = sol1[0].rhs()
sage: y1 = sol1[1].rhs()
sage: z1 = sol1[2].rhs()
sage: x1,y1,z1
(13*r2 - 5, 10*r2 - 4, r2)
sage: eq1(x=x1,y=y1,z=z1)
1 == 1
```

Exact solutions
to equations

Exact solutions

Extracting solutions

Systems of linear
equations

Approximate
solutions to
equations

Summary

Outline

1 Exact solutions to equations

Exact solutions

Extracting solutions

Systems of linear equations

2 Approximate solutions to equations

3 Summary

Why approximate?

- Exact solutions often... *complicated*

$$-\frac{1}{2} \cdot \sqrt[3]{\frac{\sqrt{3713}}{54} + \frac{7}{6}} + \frac{-2 + 2i\sqrt{3}}{9} \cdot \sqrt[3]{\frac{\sqrt{3713}}{54} + \frac{7}{6}}$$

- Approximate solutions easier to look at, manipulate
 $-0.8280018073 - 0.8505454986i$
- Approximation often *much, much* faster!

- except when approximation fails
 - bad condition numbers
 - rounding errors
 - inappropriate algorithm (real solver, complex roots)

The `find_root()` command

`find_root(equation, xmin, xmax)` where

- *equation* has a root between real numbers *xmin* and *xmax*
- reports an error if no root exists
- this is a *real solver*: looks for *real* roots
- uses Scipy package

Example

```
sage: find_root(x**5+2*x+1==0,-10,0)
-0.48638903593454297
```

```
sage: find_root(x**5+2*x+1==0,0,10)
```

... output cut...

```
RuntimeError: f appears to have no zero on the
interval
```

The `.roots()` command

polynomial.`roots()` ordinarily finds exact roots of a polynomial, along with multiplicities

- reports error if cannot find explicit roots
- approximate real roots: option `ring=RR`
- approximate complex roots: option `ring=CC`
- uses Scipy package
- “multiplicity” = “shape” of root
 - linear, quadratic, cubic, ...

Exact solutions
to equations

Exact solutions

Extracting solutions

Systems of linear
equations

Approximate
solutions to
equations

Summary

field addition, multiplication as in rational, real,
complex numbers

Ring?!?

field addition, multiplication as in rational, real,
complex numbers

ring addition, multiplication common to integers,
matrices, and fields

+ as usual

× weird sometimes

- $ab \neq ba$ matrices
- no $1/a$ even if $a \neq 0$ integers, matrices
- $ab = 0$ but $a, b \neq 0$ matrices

Exact example

```
sage: p = x**3 + 2*x**2 - 4*x - 8
```

```
sage: p.roots()
```

```
[(2, 1), (-2, 2)]
```

roots are 2 (mult. 1) and -2 (mult. 2)

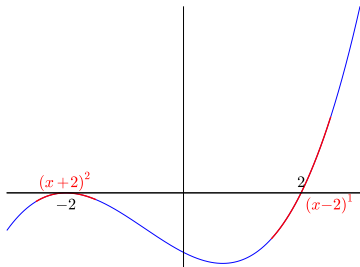
Exact example

```
sage: p = x**3 + 2*x**2 - 4*x - 8
```

```
sage: p.roots()
```

```
[(2, 1), (-2, 2)]
```

roots are 2 (mult. 1) and -2 (mult. 2)



see if you can make Sage produce this image!

Approximate example

```
sage: p = x**5 + 2*x + 1
```

```
sage: p.roots()
```

... output cut ...

```
RuntimeError: no explicit roots found
```


Approximate example

```
sage: p = x**5 + 2*x + 1
```

```
sage: p.roots()
```

... output cut...

```
RuntimeError: no explicit roots found
```

```
sage: p.roots(ring=RR)
```

```
[(-0.486389035934543, 1)]
```

root approximately -0.486389 w/multiplicity 1

Approximate example

```
sage: p = x**5 + 2*x + 1
```

```
sage: p.roots()
```

... output cut...

```
RuntimeError: no explicit roots found
```

```
sage: p.roots(ring=RR)
```

```
[(-0.486389035934543, 1)]
```

root approximately -0.486389 w/multiplicity 1

Fundamental Theorem of Algebra

Every polynomial of degree n has n complex roots.

Where are the other 4 roots?

Extract and use complex roots

```
sage: sols = p.roots(ring=CC)
```

How can we extract roots?

Exact solutions
to equations

Exact solutions

Extracting solutions

Systems of linear
equations

Approximate
solutions to
equations

Summary

Extract and use complex roots

Exact solutions
to equations

Exact solutions
Extracting solutions
Systems of linear
equations

Approximate
solutions to
equations

Summary

```
sage: sols = p.roots(ring=CC)
```

How can we extract roots?

`sols` is a list of tuples (*root*, *multiplicity*):
need to extract tuple, *then* root

```
sage: x0 = sols[0]
```

want first root

```
sage: x0
```

```
(-0.486389035934543, 1)
```

oops! this is the tuple!

```
sage: x0 = sols[0][0]
```

root is first element of tuple

```
sage: x0
```

```
-0.486389035934543
```

```
sage: x1 = sols[1][0]
```

want second root

```
sage: x1
```

```
-0.701873568855862 - 0.879697197929823*I
```

What is going on here?

Exact solutions
to equations

Exact solutions
Extracting solutions
Systems of linear
equations

Approximate
solutions to
equations

Summary

sols

0	0	$-0.486389\dots$	(approximation)
	1	1	(multiplicity)
1	0	$-0.701873\dots - 0.879697\dots i$	(approximation)
	1	1	(multiplicity)
⋮		⋮	

What is going on here?

Exact solutions
to equations

Exact solutions
Extracting solutions
Systems of linear
equations

Approximate
solutions to
equations

Summary

sols

0	0	$-0.486389\dots$	(approximation)
	1	1	(multiplicity)
1	0	$-0.701873\dots - 0.879697\dots i$	(approximation)
	1	1	(multiplicity)
⋮		⋮	

- first bracket: gets solution
- each solution is a tuple
 - second bracket: gets information about solution

[0] approximation

[1] multiplicity

Outline

1 Exact solutions to equations

Exact solutions

Extracting solutions

Systems of linear equations

2 Approximate solutions to equations

3 Summary

Summary

- distinguish `=` (assignment) and `==` (equality)
- Sage can find *exact* or *approximate* roots
- `solve()` finds exact solutions
 - not all equations can be solved exactly
 - systems of linear equations always exact
 - extract using `[]` and `.rhs()`
- `find_root()` approximates real roots on an interval
 - error if no roots on interval
- `.roots(ring=...)` approximates roots
 - RR for real roots only; CC for all complex roots
 - append to polynomial or equation