MAT 305: Mathematical Computing

Solving equations in Sage

John Perry

University of Southern Mississippi

Fall 2011

Outline

Outline

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 equations (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
```

(translation: I dunno)

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

```
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

```
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)

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

(need to expand product)

Systems of linear equations

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

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]]
```

r1?!? What is r1?

`r1` is a *parameter* that can take infinitely many values

```
[[x == 13*r1 - 5, y == 10*r1 - 4, z == r1]]
```

corresponds to

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

Example

$t = 0$?

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

$$\begin{aligned} 3(-5) - 4(-4) + 0 &= 1 \\ 2(-5) - 3(-4) + 4(0) &= 2 \\ -6(-5) + 8(-4) - 2(0) &= -2. \end{aligned}$$

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
```

Outline

Why approximate?

- Exact solutions often… *complicated*
$$-\frac{1}{2}\cdot\sqrt[3]{\frac{\sqrt{3713}}{54}+\frac{7}{6}}\cdot\left(1+i\sqrt{3}\right)+\frac{-2+2i\sqrt{3}}{9}\cdot\sqrt[3]{\frac{\sqrt{3713}}{54}+\frac{7}{6}}$$
- 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
- complex roots: option `ring=CC`
    - approximate numbers in $\mathbb{C}$
- uses Scipy package

Example

```
sage:  p = x**5+2*x+1
sage:  p.roots()
```
...output cut...
```
RuntimeError:  no explicit roots found
sage:  p.roots(ring=CC)
[(-0.486389035934543, 1),
 (-0.701873568855862 - 0.879697197929823*I, 1),
 (-0.701873568855861 + 0.879697197929823*I, 1),
 (0.945068086823134 - 0.854517514439046*I, 1),
 (0.945068086823133 + 0.854517514439046*I, 1)]
```

notice: each root has multiplicity 1

Extract and use complex roots

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

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

```
sage: x0 = sols[0]
```
want first root

```
sage: x0
(-0.486389035934543, 1)
```

oops! I want only the root; I have 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
```

Outline

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=CC)` approximates real and complex roots
  - append to polynomial or equation