PIC 16, Spring 2018

Lecture 6M: Sympy

Monday, May 7, 2018

Matt Haberland

Announcements

- Assignment 5W due

Intended Learning Outcomes

By the end of lecture, students are intended to be able to:

- use SymPy to:
  - perform mathematically exact algebraic manipulations,
  - solve equations,
  - differentiate functions, and
  - perform indefinite and definite integrals.

Activities

- Finish assignment 5W
- Start assignment 6M
  - Option 1: practice GUIs, make a silly calculator
  - Option 2: do meaningful symbolic manipulation

Gotchas

- SymPy is just a module. It doesn't change the rules of the Python language.
  - `1/2` is still `0`

    Use `Rational(1,2)`
  -
```
x = symbols('x')
expr = x + 1
x = 2
print(expr)
```

`x + 1`

    If you want to substitute the value 2 for `x`, use: `expr.subs(x, 2)`
  - `==` is still an operator that *tests* for equality, and SymPy's definition of equality may be different from yours
    - If you want to check mathematical equality, simplify the difference between expressions and check whether it's 0.
    - If you want evidence that two expressions are mathematically equal, you can use the `equals` method (e.g. `lhs.equals(rhs)`). It substitutes random floating point values and compares the results.
    - If you want to represent an (equality) equation as an object (say, to pass into the `solve` function), there's a class for that: `Eq`.

Operators

- Could Sympy have changed ^ to mean exponentiation?

e.g.:
```
x = symbols('x')
display(x^2)
```

x^2

Yes.
Challenge: subclass `Symbol` to make this possible

- Could Sympy have changed / to always create a `Rational`?

e.g.
```
x = symbols('x')
display(x + 1/2)
```

$x + \frac{1}{2}$

No. Not at all.

- Could Sympy have made = perform substitution?

e.g.
```
x = symbols('x')
e = x + 1
x = 2
display(e)
```

3

Sort of. But not cleanly, IMO.
Challenge: subclass `Symbol` and `Add` to make this possible. It only has to work properly for addition.

evalf vs subs

- evalf is for performing *numerical* substitutions and evaluating the result as a floating point value:

e.g.:

```
x = symbols('x')
y = x + 1
y.evalf(subs={x:pi})
```

→ 4.14159265358979

```
x = symbols('x')
y = x + 1
y.evalf(subs={x:x**2})
```

→ (error)

- subs is for performing symbolic substitutions

e.g.:

```
x = symbols('x')
y = x + 1
y.subs(x, pi)
```

→ $1 + \pi$

```
x = symbols('x')
y = x + 1
y.subs(x, x**2)
```

→ $1 + x^2$

lambdify

- `lambdify` is for converting an expression into a Python function (so that you can evaluate numerical results conveniently)

e.g.:

```
x = symbols('x')
y = x + 1
z = lambdify(x, y)
```

essentially defines z like:

```
z = lambda x: x + 1
```

It is very useful when you want to symbolically derive (large, complicated) equations, but then turn them automatically into Python functions for doing numerical computations with them.

?

```
equa = exp(x*pi.evalf()*sqrt(-1))
print equa.subs(x, 1.0/2)

f = lambdify(x, equa, "numpy")
print f(1.0/2)
```

The results are:

```
exp(1.5707963267949*I)
(1.61554457443e-15+1j)
```