Lab: Symbolic and Numeric Math with Sympy

Copyright Oliver Serang, 2018
University of Montana Department of Computer Science

Day 1

We can do basic symbolic math using the sympy package:

```python
# make it so we don’t need to do sympy.this and sympy.that
# all over the place
from sympy import *

# create a symbolic variable named x which has symbol ’x’:
x = symbols(’x’)

fibonacci_characteristic_poly = x**2 - x - 1

# solve fibonacci characteristic polynomial = 0:
solutions = solve(fibonacci_characteristic_poly, x)

# print each solution:
for sol in solutions:
    print ’Exact solution:’, sol
```

This outputs a sympy formatted forms of the symbolic expressions $\frac{1+\sqrt{5}}{2}$ and $\frac{1-\sqrt{5}}{2}$.

We can make these solutions numeric by adding the following evalf code to the listing above:

```python
for sol in solutions:
    print ’Approximate solution:', sol.evalf()
```

This will output numeric approximations of the symbolic expressions above: 1.61803398874989, -0.618033988749895.

We can get more precision by providing the number of decimal points to evalf:

```python
for sol in solutions:
    print ’Approximate solution (80 decimal points):’, sol.evalf(80)
```
Given an expression, we can substitute values in for variables using a dictionary. Note that we do not use quotes on the variables, because they are the keys, not a string.

```python
# first 6 terms of Taylor series:
e_taylor_series = 1 + x + x**2/2 + x**3/6 + x**4/24 + x**5/120

# compute exp(1)
print 'e:', exp(1).evalf()
# approximate e
print 'Taylor approximation:', e_taylor_series.subs({x:1}).evalf()
```

We can solve multivariate symbolic equations as follows: Given an expression, we can substitute values in for variables using a dictionary. Note that we do not use quotes on the variables, because they are the keys, not a string.

```python
y,z = symbols('y z')

# list of all solutions:
print solve(x**y + z, y)
```

Recall from algebra that we need an equal number of independent equations and variables to compute actual values. We can do this by providing more equations as a list and sending in all the free variables:

```python
# solve simultaneous equations:
# x*y - y - 1 = 0
# x - 2*y = 0
print solve([x*y-y-1, x-2*y], x,y)
```

Each solution in the printed list above will be an (x,y) tuple pair. Note that when we run `print solve([x*y-y-1, x-2*y], y,x)` instead, we print the tuples in the (y,x) order instead. If we are worried we might forget whether we did `solve(..., x,y)` or `solve(..., y,x)`, we can ask solve to return our solutions as dictionaries:

```python
# solve simultaneous equations:
# x*y - y - 1 = 0
# x - 2*y = 0
solutions = solve([x*y-y-1, x-2*y], x,y, dict=True)
for sol in solutions:
```

```
We can conveniently use these dictionary solutions with the `subs` function to substitute these values into different expressions.

```python
# solve simultaneous equations:
# x*y - y - 1 = 0
# x - 2*y = 0
solutions = solve([x*y-y-1, x-2*y], x,y, dict=True)
some_expr = x**y
for sol in solutions:
    print some_expr.subs(sol)
```

Note that we can receive imaginary solutions \( I = \sqrt{-1} \). Verify that `print I**2` prints out \(-1\). 

`sympy` is not able to solve every kind of symbolic equation:

```python
print solve(cos(x**x) - sin(exp(x)), x)
```

prints `NotImplementedError: and No algorithms are implemented to solve equation.`

In these cases, we can use a numerical solution with `nsolve`:

```python
print nsolve(cos(x**x) - sin(exp(x)), 1)
```

The above prints 0.912557797675771. Note that we do not give the variable \( x \) as the final argument, because `nsolve` only works for univariate equations (unless all other variables are constants). Instead, we provide the starting value of \( x \) to the solver. Tweaking this variable can discover alternate solutions: `print nsolve(cos(x**x) - sin(exp(x)), -1)` prints alternate solution \(-0.0200280642803447 - 0.258093186348457*I\).

1. Find and print all \( x \) (symbolically) for which \( x^3 - 2*x = 1 \) by using the `solve` function.
2. Find and print all \( x \) (numerically) for which \( x^3 - 2*x = 1 \) by using the `solve` and `evalf` functions.
3. Using `nsolve`, numerically find values of \( x \) for which \( x^3 - 2*x = 1 \). Find starting values of \( x \) so that you find all solutions found in the question above.
4. Find all solutions to the simultaneous equations

\[
\begin{align*}
\sin(x) + 2\sin(y) &= 0 \\
2\cos(x) - \sin(y) &= 0.
\end{align*}
\]

Use `dict=True` to get \{x:..., y:...\} dictionaries for each solution.

5. Loop through the solutions to the question above and lookup the \(x\) and \(y\) solutions via \[x\] and \[y\]. Print out the \(x\) and \(y\) values numerically using `evalf`.

6. We can cast `sympy` objects as `str` types. Use `str(pi.evalf())` and the `[i]` operator to lookup and print the 4th digit place of \(\pi\) after the decimal point.

7. Use `str` and `evalf` to lookup the 100th digit of \(\pi\) after the decimal point.

---

**Day 2**

We can also use `sympy` to compute derivatives.

```python
some_expr = x**3 - 3*x + 10
print some_expr.diff(x)
```

This will compute \(\frac{\partial}{\partial x} (x^3 - 3x + 10) = 3x^2 - 3\) and print the result.

Likewise, we can use `sympy` to compute integrals (\(i.e.,\) areas under curves):

```python
some_expr = x**3 - 3*x + 10
print some_expr.integrate(x)
```

Both derivatives and integrals generalize to the multivariate case:

```python
multivariate_expr = x**3*y - y**3*x
print multivariate_expr.diff(x)
print multivariate_expr.diff(y)
```

1. Recall that the maximum value of a function may occur where its derivative is 0. Find the values of \(x\) for which the following derivative achieves 0:

\[
\frac{\partial}{\partial x} \left( -\frac{x^4}{4} + \frac{2x^3}{3} + \frac{5x^2}{2} - 6x \right) = 0.
\]
2. Repeat the above using `dict=True`. Then iterate through those solutions and substitute them into the expression \(-\frac{x^4}{4} + \frac{2x^3}{3} + \frac{5x^2}{2} - 6x\). At which of these \(x\) values in the question above does the function attain a maximum?

3. Using `pylab`, plot \(-\frac{x^4}{4} + \frac{2x^3}{3} + \frac{5x^2}{2} - 6x\) from \(x = -4\) to \(x = 6\) (inclusive). At what values of \(x\) in the plot does \(-\frac{x^4}{4} + \frac{2x^3}{3} + \frac{5x^2}{2} - 6x\) achieve a maximum?