Modeling and Simulation in Python

Chapter 9

Lab Author @ Joseph Simone

```
# Configure Jupyter to display the assigned value after an assignment
%config InteractiveShell.ast_node_interactivity='last_expr_or_assign'

# import everything from SymPy.
from sympy import *

# Set up Jupyter notebook to display math.
init_printing()
```

The following displays SymPy expressions and provides the option of showing results in LaTeX format.

```
from sympy.printing import latex

def show(expr, show_latex=False):
    """Display a SymPy expression.

    expr: SymPy expression
    show_latex: boolean
    """
    if show_latex:
        print(latex(expr))
    return expr
```

Analysis with SymPy

Create a symbol for time.

```
t = symbols('t')
```

t

If you combine symbols and numbers, you get symbolic expressions.

```
expr = t + 1
```

$t + 1$

The result is an `Add` object, which just represents the sum without trying to compute it.

```
type(expr)
```

```
sympy.core.add.Add
```

`subs` can be used to replace a symbol with a number, which allows the addition to proceed.

```
expr.subs(t, 2)
```

3

`f` is a special class of symbol that represents a function.

```
f = Function('f')
```

f

```

```

The type of `f` is `UndefinedFunction`

```
type(f)
```

sympy.core.function.UndefinedFunction

SymPy understands that `f(t)` means `f` evaluated at `t`, but it doesn't try to evaluate it yet.

```
f(t)
```

$f(t)$

`diff` returns a `Derivative` object that represents the time derivative of `f`

```
dfdt = diff(f(t), t)
```

$$\frac{d}{dt}f(t)$$

```
type(dfdt)
```

sympy.core.function.Derivative

We need a symbol for `alpha`

```
alpha = symbols('alpha')
```

α

Now we can write the differential equation for proportional growth.

```
eq1 = Eq(dfdt, alpha*f(t))
```

$$\frac{d}{dt}f(t) = \alpha f(t)$$

And use `dsolve` to solve it. The result is the general solution.

```
solution_eq = dsolve(eq1)
```

$f(t) = C_1 e^{\alpha t}$

We can tell it's a general solution because it contains an unspecified constant, `C1`.

In this example, finding the particular solution is easy: we just replace `C1` with `p_0`

```
C1, p_0 = symbols('C1 p_0')
```

```
particular = solution_eq.subs(C1, p_0)
```

$f(t) = p_0 e^{\alpha t}$

In the next example, we have to work a little harder to find the particular solution.

Solving the quadratic growth equation

We'll use the (r, K) parameterization, so we'll need two more symbols:

```
r, K = symbols('r K')
```

Now we can write the differential equation.

```
eq2 = Eq(diff(f(t), t), r * f(t) * (1 - f(t)/K))
```

$$\frac{d}{dt} f(t) = r \left(1 - \frac{f(t)}{K}\right) f(t)$$

And solve it.

```
solution_eq = dsolve(eq2)
```

$$f(t) = \frac{K e^{C_1 K + rt}}{e^{C_1 K + rt} - 1}$$

The result, `solution_eq`, contains `rhs`, which is the right-hand side of the solution.

```
general = solution_eq.rhs
```

$$\frac{K e^{C_1 K + rt}}{e^{C_1 K + rt} - 1}$$

We can evaluate the right-hand side at $t = 0$

```
at_0 = general.subs(t, 0)
```

$$\frac{K e^{C_1 K}}{e^{C_1 K} - 1}$$

Now we want to find the value of `C1` that makes `f(0) = p_0`.

So we'll create the equation `at_0 = p_0` and solve for `C1`. Because this is just an algebraic identity, not a differential equation, we use `solve`, not `dsolve`.

The result from `solve` is a list of solutions. In this case, we have reason to expect only one solution, but we still get a list, so we have to use the bracket operator, `[0]`, to select the first one.

```
solutions = solve(Eq(at_0, p_0), C1)
type(solutions), len(solutions)
```

```
(list, 1)
```

```
value_of_C1 = solutions[0]
```

$$\frac{\log\left(-\frac{p_0}{K-p_0}\right)}{K}$$

Now in the general solution, we want to replace `C1` with the value of `C1` we just figured out.

```
particular = general.subs(C1, value_of_C1)
```

$$-\frac{Kp_0e^{rt}}{(K-p_0)\left(-\frac{p_0e^{rt}}{K-p_0}-1\right)}$$

The result is complicated, but SymPy provides a method that tries to simplify it.

```
particular = simplify(particular)
```

$$\frac{Kp_0e^{rt}}{K+p_0e^{rt}-p_0}$$

Often simplicity is in the eye of the beholder, but that's about as simple as this expression gets.

Just to double-check, we can evaluate it at `t=0` and confirm that we get `p_0`

```
particular.subs(t, 0)
```

p_0

This solution is called the logistic function.

In some places you'll see it written in a different form:

$f(t) = \frac{K}{1+Ae^{-rt}}$

where $A = (K - p_0)/p_0$.

We can use SymPy to confirm that these two forms are equivalent. First we represent the alternative version of the logistic function:

```
A = (K - p_0) / p_0
```

$$\frac{K-p_0}{p_0}$$

```
logistic = K / (1 + A * exp(-r*t))
```

$$\frac{K}{1+\frac{(K-p_0)e^{-rt}}{p_0}}$$

To see whether two expressions are equivalent, we can check whether their difference simplifies to 0.

```
simplify(particular - logistic)
```

0

This test only works one way: if SymPy says the difference reduces to 0, the expressions are definitely equivalent (and not just numerically close).

But if SymPy can't find a way to simplify the result to 0, that doesn't necessarily mean there isn't one. Testing whether two expressions are equivalent is a surprisingly hard problem; in fact, there is no algorithm that can solve it in general.

Exercises

Exercise: Solve the quadratic growth equation using the alternative parameterization

$\frac{df(t)}{dt} = \alpha f(t) + \beta f^2(t)$

```
alpha, beta = symbols('alpha beta')
```

```
eq3 = Eq(diff(f(t), t), alpha*f(t) + beta*f(t)**2)
eq3
```

```
solution_eq = dsolve(eq3)
solution_eq
```

```
general = solution_eq.rhs
general
```

```
at_0 = general.subs(t, 0)
```

```
solutions = solve(Eq(at_0, p_0), C1)
value_of_C1 = solutions[0]
value_of_C1
```

```
solutions = solve(Eq(at_0, p_0), C1)
value_of_C1 = solutions[0]
value_of_C1
```

```
particular = general.subs(C1, value_of_C1)
particular.simplify()
```

Exercise: Use WolframAlpha to solve the quadratic growth model, using either or both forms of parameterization:

$$df(t)/dt = alpha f(t) + beta f(t)^2$$

or

$$df(t)/dt = r f(t)(1 - f(t)/K)$$

Find the general solution and also the particular solution where `f(0) = p_0`.

$\frac{df(t)}{dt} = \alpha f(t) + \beta f(t)^2$

$\frac{\frac{df(t)}{dt}}{\beta f(t)^2 + \alpha f(t)} = 1$

$\int \frac{\frac{df(t)}{dt}}{\beta f(t)^2 + \alpha f(t)}\, dt = \int 1 dt$

$\frac{\log(f(t))}{\alpha} - \frac{\log(\alpha + \beta f(t))}{\alpha} = t + c_1$

$f(t) = -\frac{\alpha e^{\alpha(t+c_1)}}{\beta e^{\alpha(t+c_1)} - 1}$