CS 3430: S19: SciComp with Py
Assignment 5
Partial Differential Equations, Exponential Growth and Decay, Carbon Dating, Elasticity of Demand

Vladimir Kulyukin
Department of Computer Science
Utah State University

February 9, 2019

Learning Objectives

1. Partial Differential Equations
2. Exponential Growth and Decay Models
3. Carbon Dating
4. Quantitative Theory of the Firm: Elasticity of Demand

Problem 1: Partial Differential Equations (1 point)

Write a function `solve_pdeq(k1, k2)` that takes the values of k_1 and k_2 in the partial differential equation $k_1 y' = k_2 y$ and returns a function representation of a solution to this partial differential equation.

Write a function `solve_pdeq_with_init_cond(y0, k)` that takes two constants `y0` and `k` such that $y' = \mathtt{k}y$ and $\mathtt{y0} = y(0)$ and returns a function representation of a solution to the partial differential equation with the specified initial condition.

Save your code in `hw05.py`

Test 01

Solve $y' = y$.

```
def test_01(self):
    print('\n***** Unit Test 01 *****')
    eq = solve_pdeq(make_const(1.0), make_const(1.0))
    assert not eq is None
    print(eq)
    eqf = tof(eq)
    assert not eqf is None
    err = 0.0001
    gt = lambda t: math.e**t
    for t in range(100):
        assert abs(gt(t) - eqf(t)) <= err
    print('Unit Test 01: pass')
```

Here is the output of `test_01` in the Py shell.

```
***** Test 01 ************
1.0*(2.71828182846^(1.0*(t^1.0))))
Test 01: pass
```

Test 02

Solve $4y' = \frac{1}{3}y$.

```
def test_02():
    print('\n***** Unit Test 02 *****')
    eq = solve_pdeq(make_const(4.0), make_const(1.0/3.0))
    assert not eq is None
    print(eq)
    eqf = tof(eq)
    assert not eqf is None
    gt = lambda t: math.e**((1.0/12.0)*t)
    for t in range(100):
        assert abs(gt(t) - eqf(t)) <= 0.0001
    print('Test 02: pass')
```

Here is the output of `test_02` in the Py shell.

```
***** Test 02 ************
(1.0*(2.71828182846^(0.0833333333333*(t^1.0))))
Test 02: pass
```

Test 03

Solve $y' = 3y$; $y(0) = 1$.

```
def test_03():
    print('\n***** Unit Test 03 *****')
    eq = solve_pdeq_with_init_cond(make_const(1.0),
                                   make_const(3.0))
    assert not eq is None
    print(eq)
    eqf = tof(eq)
    assert not eqf is None
    def gt(t): return math.e**(3.0*t)
    err = 0.0001
    for t in range(100):
        assert abs(gt(t) - eqf(t)) <= err
    print('Test 03: pass')
```

Here is the output of `test_03` in the Py shell.

```
***** Test 03 ************
1.0*(2.71828182846^(3.0*(t^1.0))))
Test 03 pass
```

Use the examples we worked out in lecture 9 and in the reading handout to write more unit tests.

Problem 2: Growth Models for Bacteria Cultures (1 point)

Write a function `find_growth_model(p0, t, n)` that takes three constants and determines growth models for various bacteria cultures. Specifically, it returns a function representation of the exponential growth model for a bacteria culture described by the given parameters. You may assume that each culture is modeled with $P(t) = P(0)e^{kt}$, where $P(0) =$ `p0`, t is a number of hours, and n is a constant specifying how fast the initial population $P(0)$ has grown after t hours. For example, if $P(0)$ doubles in 3 hours, then $P(3) = 2P(0)$.

Use the examples we worked out in lecture 9 and in the reading handout to write your unit tests. Save your code in `hw05.py`

After `find_growth_model` works to your satisfaction, use it to solve the following problem.

A bacteria culture that exhibits exponential growth quadruples in size in 2 days. If the initial size of the culture is 20,000, what is its size after 12 hours?

Problem 3: Radioactive Decay (1/2 point)

Write a function `radioactive_decay(lmbda, p0, t)` that takes three constants and determines decay models for various radioactive materials. Specifically, it returns a function representation of the decay model for a material described by the given parameters, where `lmbda` specifies λ, `p0` – the initial amount of radioactive material in grams, and t – a number of years. You may assume that each material is modeled with $P(t) = P(0)e^{-\lambda t}$, where $P(0) =$ `p0`.

Use the examples we worked out in lecture 10 and in the reading handout to write your unit tests. Save your code in `hw05.py`.

After `radioactive_decay` works to your satisfaction, use it to solve the following problem.

A sample of 8 grams of radioactive material is placed in a vault. If $\lambda = 0.021$, how much of the material will remain in the vault after 10 years?

Problem 4: Carbon Dating (1/2 point)

Write a function `c14_carbon_dating(c14_percentage)` that takes a constant describing the percentage of ^{14}C found in an object and returns an approximate age of the object according to the carbon dating model. The object's age is a constant whose float value is rounded up with `math.ceil`.

Use the examples we worked out in lecture 10 and in the reading handout to write your unit tests. Save your code in `hw05.py`.

After `c14_carbon_dating(c14_percentage)` works to your satisfaction, use it to solve the following two problems.

Problem A: In 1947, a cave with beautiful prehistoric wall paintings was discovered in Lascaux, France. Some charcoal found in the cave contained 20% of ^{14}C expected in living trees. How old are the Lascaux cave paintings?

Problem B: A wooden chest was found in the tomb of the 25th century B.C. Chaldean king Meskalumdug of Ur. The chest contained 58.3% of ^{14}C expected in living trees. How old is the chest?

Problem 5: Elasticity of Demand (2 points)

Write a function `demand_elasticity(demand_eq, price)` that takes a function representation of a demand equation (as a function of price p) and a price constant and returns a constant whose float value describes the elasticity of demand at the specified price.

Write a boolean function `is_demand_elastic(demand_eq, price)` that takes a function representation of a demand equation (as a function of price p) and a price constant and returns True if demand is elastic at the specified price and False otherwise.

Write a boolean function `expected_rev_dir(demand_eq, price, price_dir)` that takes a function representation of a demand equation (as a function of price p), a price constant, and a price direction constant (i.e., +1 – the price will be increased; -1 – the price will be lowered) and returns a constant that specifies the expected direction of revenue after the price action is taken (i.e., +1 - revenue will go up; -1 – revenue will go down).

Use the examples we worked out in lecture 10 and in the reading handout to write your unit tests. Save your code in `hw05.py`.

After these functions work to your satisfaction, use them to solve the following problem.

The number of people attending a show in a movie theater at a price of p dollars per ticket is $q = (18,000/p) - 1500$. Is demand elastic at $p = \$6$? If the price is lowered, will revenue increase or decrease?

What to Submit

Submit your code in `hw05.py`. You should also submit all the files needed to run your code. The easiest and safest thing for you to do is to zip your whole directory into `hw05.zip` and upload it to Canvas.

Do not change the names of the files that were given to you or the names of the functions you are asked to implement. The weekly unit tests that I write for the grader depend on these names remaining the same.

Happy Hacking!