CS 3430: S19: SciComp with Py
Assignment 3
Theory of the Firm, Implicit Differentiation, Related Rates

Vladimir Kulyukin
Department of Computer Science
Utah State University

January 26, 2019

Learning Objectives

1. Product and Quotient Rules
2. Quantitative Theory of the Firm
3. Implicit Differentiation
4. Related Rates

Warmup

Practice the product and quotient rules of differentiation by taking the derivatives of the functions below. If you feel comfortable with these rules, move on to Problem 1. You can turn these into your unit tests for Problem 1.

1. $\frac{d}{dx}(x+1)(x^3+5x+2)$; answer: $4x^3+3x^2+10x+7$;
2. $\frac{d}{dx}(2x^4-x+1)(-x^5+1)$; answer: $-18x^8+6x^5-5x^4+8x^3-1$;
3. $\frac{d}{dx}\left(\frac{x^2-1}{x^2+1}\right)$; answer: $\frac{4x}{(x^2+1)^2}$;
4. $\frac{d}{dx}\left(\frac{x+11}{x-3}\right)^3$; answer: $\frac{-42(x+11)^2}{(x-3)^4}$.
5. $\frac{d}{dx}\frac{x^2}{(x^2+1)^2}$; answer: $\frac{-2x^3+2x}{(x^2+1)^3}$.

Problem 1: Product and Quotient Rules (2 points)

Extend your differentiation engine in `deriv.py` with the product and quotient rules of differentiation. Specifically, your implementation of `prod_deriv` should be able to handle not just products of constants and powers but also products of a sum and a sum, a sum and a power, a sum and a product, a sum and a quotient, a power and a sum, a power and a power, a power and a product, a power and a quotient, a product and a sum, a product and a power, a product and a product, a product and a quotient, a quotient and a sum, a quotient and a power, a quotient and a product, and a quotient and a quotient. This explicit enumeration is a rather windy way of what can be stated succinctly: your differentiation engine must be commutative. Use the implementation of the quotient class in `quot.py` and make appropriate changes in `tof.py` to convert the outputs of your `deriv` function to Python functions.

Test 01

Let's differentiate $\frac{d}{dx}(x+1)(x^3+5x+2)$.

```
def test_01():
    print('\n***** Test 01 ***********')
    e1 = make_plus(make_pwr('x', 1.0), make_const(1.0))
    e2 = make_pwr('x', 3.0)
    e3 = make_prod(make_const(5.0), make_pwr('x', 1.0))
    e4 = make_plus(e2, e3)
    e5 = make_plus(e4, make_const(2.0))
    e6 = make_prod(e1, e5)
    # 1) print the expression we just constructed
    print('-- function expression is:\n')
    print(e6)
    # 2) differentiate and make sure that it not None
    drv = deriv(e6)
    assert not drv is None
    print('-- derivative is:\n')
    print(e6)
    # 3) convert drv into a function
    e6f = tof(drv)
    assert not e6f is None
    # steps 2) and 3) can be combined into tof(deriv(e6)).
    # 4) construct the ground truth function
    gt = lambda x: 4.0*(x**3) + 3*(x**2) + 10.0*x + 7.0
    # 5) compare the ground gruth with what we got in
    # step 3) on an appropriate number range.
    print('\n Comparison with ground truth:\n')
    err = 0.00001
    for i in range(10):
```

```
        print(e6f(i), gt(i))
        assert abs(e6f(i) - gt(i)) <= err
    print('Test 01: pass')
```

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

```
***** Test 01 ***********
-- function expression is:

(((x^1.0)+1.0)*(((x^3.0)+(5.0*(x^1.0)))+2.0))

-- derivative is:

(((x^1.0)+1.0)*(((x^3.0)+(5.0*(x^1.0)))+2.0))

--comparison with ground truth:

(7.0, 7.0)
(24.0, 24.0)
(71.0, 71.0)
(172.0, 172.0)
(351.0, 351.0)
(632.0, 632.0)
(1039.0, 1039.0)
(1596.0, 1596.0)
(2327.0, 2327.0)
(3256.0, 3256.0)
Test 01: pass
```

Test 02

Let's do $\frac{d}{dx}(2x^4 - x + 1)(-x^5 + 1)$.

```
def test_02():
    print('\n***** Test 02 ***********')
    e1 = make_prod(make_const(2.0), make_pwr('x', 4.0))
    e2 = make_prod(make_const(-1.0), make_pwr('x', 1.0))
    e3 = make_plus(e1, e2)
    e4 = make_plus(e3, make_const(1.0))
    e5 = make_prod(make_const(-1.0), make_pwr('x', 5.0))
    e6 = make_plus(e5, make_const(1.0))
    e7 = make_prod(e4, e6)
    print('-- function expression is:\n')
    print(e7)
    drv = deriv(e7)
```

```
    assert not drv is None
    print('\n-- derivative is:\n')
    print(drv)
    e7f = tof(drv)
    assert not e7f is None
    gt = lambda x: -18.0*(x**8) + 6.0*(x**5) - 5.0*(x**4) +  8.0*(x**3) - 1.0
    err = 0.00001
    print('\n--comparison with ground truth:\n')
    for i in range(10):
        print(e7f(i), gt(i))
        assert abs(e7f(i) - gt(i)) <= err
    print('Test 02: pass')
```

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

```
***** Test 02 ***********
-- function expression is:

((((2.0*(x^4.0))+(-1.0*(x^1.0)))+1.0)*((-1.0*(x^5.0))+1.0))

-- derivative is:

(((((2.0*(x^4.0))+(-1.0*(x^1.0)))+1.0)*((-1.0*(5.0*(x^4.0)))+0.0))
+(((-1.0*(x^5.0))+1.0)*(((2.0*(4.0*(x^3.0)))+
(-1.0*(1.0*(x^0.0))))+0.0)))

--comparison with ground truth:

(-1.0, -1.0)
(-10.0, -10.0)
(-4433.0, -4433.0)
(-116830.0, -116830.0)
(-1174273.0, -1174273.0)
(-7014626.0, -7014626.0)
(-30191185.0, -30191185.0)
(-103674838.0, -103674838.0)
(-301809665.0, -301809665.0)
(-774513658.0, -774513658.0)
Test 02: pass
```

Test 03

Let's do $\frac{d}{dx}\left(\frac{x+11}{x-3}\right)^3$.

```
def test_03():
```

```
    print('\n***** Test 03 ***********')
    q = make_quot(make_plus(make_pwr('x', 1.0),
                            make_const(11.0)),
                  make_plus(make_pwr('x', 1.0), make_const(-3.0)))
    pex = make_pwr_expr(q, 3.0)
    print('-- function expression is:\n')
    print(pex)
    pexdrv = deriv(pex)
    assert not pexdrv is None
    print('\n-- derivative is:\n')
    print(pexdrv)
    pexdrvf = tof(pexdrv)
    assert not pexdrvf is None
    gt = lambda x: -42.0*(((x + 11.0)**2)/((x - 3.0)**4))
    err = 0.00001
    print('\n--comparison with ground truth:\n')
    for i in range(10):
        if i != 3.0:
            print(pexdrvf(i), gt(i))
            assert abs(pexdrvf(i) - gt(i)) <= 0.001
    print('Test 03: pass')
```

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

```
***** Test 03 ***********
-- function expression is:

((((x^1.0)+11.0)/((x^1.0)+-3.0))^3.0)

-- derivative is:

((3.0*((((x^1.0)+11.0)/((x^1.0)+-3.0))^2.0))*(((((x^1.0)+-3.0)
*((1.0*(x^0.0))+0.0))+(-1.0*(((x^1.0)+11.0)*((1.0*(x^0.0))+0.0))))/
(((x^1.0)+-3.0)^2.0)))

--comparison with ground truth:

(-62.74074074074073, -62.74074074074074)
(-378.0, -378.0)
(-7098.0, -7098.0)
(-9450.0, -9450.0)
(-672.0, -672.0)
(-149.85185185185188, -149.85185185185185)
(-53.15625, -53.15625)
(-24.259200000000003, -24.2592)
(-12.962962962962964, -12.962962962962962)
```

```
Test 03: pass
```

Let's put to use our differentiation engine to solve some real world scientific computing problems. In each of the three problems below, you will need to abstract the statement of the problem into a Python function to solve a specific *type* of problem.

Problem 2: Quantitative Theory of the Firm: ($1\frac{1}{2}$ points)

The demand equation for `X Logistics`, a small trucking company, is

$$p = \frac{1}{12}x^2 - 10x + 300,\ 0 \leq x \leq 60,$$

where x is the number of rides per day. The constraint says that the company cannot do more than 60 rides per day (e.g., it does not have enough trucks). Find the value of x and the corresponding price the company must charge to maximize its daily revenue.

Generalize the above problem into a function `maximize_revenue` that takes an expression of the demand equation and a keyword that specifies the constraint on the values of the variable in the demand equation and returns a tuple of three constants: the number of units that results in maximum revenue, i.e., the value of x, the value of maximum revenue, and the unit price when the revenue is maximum. The constraint should be specified either as a lambda or a one-parameter named function. For example, in the above problem, the constraint is `lambda x: 0 <= x <= 60`. These constraints are needed to weed out negative values, because in the quantitative theory of the firm, as we do function optimization, we typically ignore negative numbers of units. Who is interested in -2 rides per day anyway? What does it even mean? Save your implementation of `maximize_revenue` in `hw03.py`. Here is a test that applies `maximize_revenue` to the above problem.

```
from hw03 import maximize_revenue
def max_rev_test():
    print('\n***** Max Revenue Test ************')
    e1 = make_prod(make_const(1.0/12.0), make_pwr('x', 2.0))
    e2 = make_prod(make_const(-10.0), make_pwr('x', 1.0))
    sum1 = make_plus(e1, e2)
    dmndf_expr = make_plus(sum1, make_const(300.0))
    num_units, rev, price = \
              maximize_revenue(dmndf_expr,
                               constraint=lambda x: 0 <= x <= 60)
    print('x = ', num_units.get_val())
    print('rev = ', rev.get_val())
```

```
    print('price = ', price.get_val())
    print('Max Revenue Test: pass')
```

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

```
***** Max Revenue Test ************
('x = ', 20.0)
('rev = ', 2666.6666666666665)
('price = ', 133.33333333333331)
Max Revenue Test: pass
```

This output has a straightforward interpretation: to maximize its daily revenue, `X Logistics` must do 20 rides per day, at a price of $\approx$ \$133.33 per ride for a maximum revenue of $\approx$ \$2,666.67 per day.

Develop a couple, at least two, of your own unit tests for this type of problem from the problems we worked out on the board in class and the week 3 reading handouts.

Problem 3: Science and Geology: (1 point)

Due to an oil tanker accident, the leaking oil is forming an approximately circular disk on the surface of the ocean. The response team has determined with its measurement equipment that when the radius of the disk r is equal to ≈ 150 meters, the radius is increasing at a rate of ≈ 20 meters per hour. The volume of the disk is $V \approx 0.02\pi r^2$. How fast is the volume of the disk changing when the response team measures the disk's radius?

Recall that we worked out this problem on the board at the end of lecture 6. Generalize the above problem into a function `dydt_given_x_dxdt(yt, x, dxdt)`, where `yt` is a function expression that relates `y` to `x`, and `x` and `dxdt` are constant objects. Notationally, `dydt` denotes the rate of change of `y` with respect to `t`, i.e., $\frac{dy}{dt}$, and `dxdt` – the rate of change of `x` with respect to `t`, i.e., $\frac{dx}{dt}$.

Also, note that `x` is just a variable name. In the above problem, for example, `x` is the radius `r` of the oil disk and `dxdt` is the same as $\frac{dr}{dt}$, i.e., 20 meters per hour, and `yt` is specified by $V \approx 0.02\pi r^2$.

The function `dydt_given_x_dxdt(yt, x, dxdt)` returns the value of $\frac{dy}{dt}$ when the value of `x` is specified by its second parameter and the value of $\frac{dx}{dt}$ is given by its third parameter, i.e., `dxdt`.

Let's write a test for the above problem.

```
from hw03 import dydt_given_x_dxdt
def oil_disk_test():
    yt = make_prod(make_const(0.02*math.pi),
```

```
                  make_pwr('r', 2.0))
    print(yt)
    dydt = dydt_given_x_dxdt(yt, make_const(150.0),
                             make_const(20.0))
    assert not dydt is None
    assert isinstance(dydt, const)
    print(dydt)
```

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

```
(0.0628318530718*(r^2.0))
376.991118431
```

Thus, the volume of the oil disk is increasing (because the value is positive) at ≈ 377 m^3 per hour.

Save your implementation of `dydt_given_x_dxdt(yt, x, dxdt)` in `hw03.py`. A word of caution: your implementation of `dydt_given_x_dxdt` does not have to solve the generic implicit differentiation problem; it just needs to be powerful enough to handle only *this* type of problem.

Problem 4: Radiology and Health Care: ($\frac{1}{2}$ point)

A patient is receiving radiation treatment for a spherical tumor on her arm. The doctor has determined that when the radius of the tumor is ≈ 10.3 mm, the radius is decreasing at a rate of ≈ 1.75 mm per week. What is the rate at which the volume of the tumor is changing when the doctor measures its radius if the volume of the tumor is given by $V \approx 0.003\pi r^3$?

Use your implementation of `dydt_given_x_dxdt` from Problem 3 to solve this problem by writing a function `arm_tumor_test` in `hw03.py` that, similarly to `oil_disk_test`, prints out the rate at which the patient's tumor is changing.

What to Submit

You definitely need to submit `deriv.py`, `tof.py`, and `hw03.py`. But you should also submit all the files needed to run your code. After grading Assignment 1, the grader told me that some students did not submit all the files that they had modified and/or written needed for running their code. Zip all your files in `hw03.zip` and submit it via 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 unit tests that I write for the grader every week depend on these names remaining the same.

Happy Hacking!