CS 3430: S19: SciComp with Py
Assignment 4
Differentiation of $y = e^x$ and $y = lnx$ and Logarithmic Differentiation

Vladimir Kulyukin
Department of Computer Science
Utah State University

February 2, 2019

Learning Objectives

1. Differentiation of Exponential Functions
2. Differentiation of Logarithmic Functions
3. Logarithmic Differentiation

Warmup

Practice differentiating exponential and logarithmic functions and logarithmic differentiation. If you feel comfortable with this differentiation, move on to the coding problems. You can turn these examples into your unit tests later as you debug your solutions. Use the examples in the reading handouts in case you need more unit tests.

1. $\frac{d}{dx}e^{7x}$; **answer:** $7e^{7x}$;
2. $\frac{d}{dx}e^{(x^3+2x^2-10x+100)}$; **answer:** $e^{(x^3+2x^2-10x+100)}(3x^2+4x-10)$;
3. $\frac{d}{dx}4e^{3x}$; **answer:** $12e^{3x}$;
4. $\frac{d}{dx}(lnx)^7$; **answer:** $\frac{7(lnx)^6}{x}$.
5. $\frac{d}{dx}x^2lnx$; **answer:** $x(1+2lnx)$.
6. $\frac{d}{dx}(x+10)(x-5)(x+23)$; **answer:** $(x+10)(x-5)(x+23)\left(\frac{1}{x+10}+\frac{1}{x-5}+\frac{1}{x+23}\right)$;
7. $\frac{d}{dx}(x+1)^4(4x-1)^2$; **answer:** $(x+1)^4(4x-1)^2\left(\frac{4}{x+1}+\frac{8}{4x-1}\right)$.

Problem 1: Differentiation of Exponential and Logarithmic Functions (3 points)

Extend your differentiation engine in `deriv.py` with the rules for differentiating $y = e^x$ and $y = lnx$. Your extensions should handle such variants of e^x as Ce^{kx}, where C and k are reals, and $e^{g(x)}$, where $g(x)$ is a differentiable function. Your extensions should also cover such variants of lnx as $ln|x|$, $(lnx)^k$, where k is a real, and $ln(g(x))$, where $g(x)$ is a differentiable function. Use the implementations of the absolute value and natural logarithm classes in `absv.py` and `ln.py`, respectively, to represent function expressions and make appropriate changes to `tof.py` to convert the outputs of your `deriv` function to regular Python functions for unit testing.

Test 01

Let's do $\frac{d}{dx}e^{5x}$.

```
import math
from maker import make_prod, make_const, make_pwr, make_e_expr
from tof import tof
from deriv import deriv

def test_01():
    print('\n***** Test 01 ************')
    fex = make_e_expr(make_prod(make_const(5.0),
                                make_pwr('x', 1.0)))
    print(fex)
    drv = deriv(fex)
    assert not drv is None
    print(drv)
    drvf = tof(drv)
    assert not drvf is None
    gt = lambda x: 5.0*(math.e**(5.0*x))
    for i in range(10):
       print(drvf(i), gt(i))
       assert abs(gt(i) - drvf(i)) <= 0.001
    print('Test 01: pass')
```

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

```
***** Test 01 ************
(2.71828182846^(5.0*(x^1.0)))
((2.71828182846^(5.0*(x^1.0)))*(5.0*(1.0*(x^0.0))))
(5.0, 5.0)
(742.0657955128829, 742.0657955128829)
```

```
(110132.32897403352, 110132.32897403352)
(16345086.862360539, 16345086.862360539)
(2425825977.048949, 2425825977.048949)
(360024496686.92883, 360024496686.92883)
(53432372907622.23, 53432372907622.23)
(7930067261567139.0, 7930067261567139.0)
(1.1769263341850975e+18, 1.1769263341850975e+18)
(1.7467135528742506e+20, 1.7467135528742506e+20)
Test 01: pass
```

Test 02

Let's do $\frac{d}{dx}e^{x^2-1}$.

```
import math
from maker import make_prod, make_const, make_pwr, make_e_expr
from maker import make_plus
from tof import tof
from deriv import deriv

def test_02():
    print('\n***** Test 02 ************')
    fex = make_e_expr(make_plus(make_pwr('x', 2.0),
                                make_const(-1.0)))
    print(fex)
    drv = deriv(fex)
    assert not drv is None
    print(drv)
    drvf = tof(drv)
    assert not drvf is None
    gt = lambda x: 2*x*(math.e**(x**2 - 1.0))
    err = 0.0001
    for i in range(10):
       print(drvf(i), gt(i))
    assert abs(gt(i) - drvf(i)) <= err
    print('Test 01: pass')
```

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

```
***** Test 02 ************
(2.71828182846^((x^2.0)+-1.0))
((2.71828182846^((x^2.0)+-1.0))*((2.0*(x^1.0))+0.0))
(0.0, 0.0)
(2.0, 2.0)
(80.34214769275066, 80.34214769275066)
```

```
(17885.74792225036, 17885.74792225036)
(26152138.979776863, 26152138.979776863)
(264891221298.4344, 264891221298.4344)
(1.9032161427761132e+16, 1.9032161427761132e+16)
(9.82343027693666e+21, 9.82343027693666e+21)
(3.6700530551513636e+28, 3.6700530551513636e+28)
(9.973120291908276e+35, 9.973120291908276e+35)
Test 02: pass
```

Test 03

Let's do $\frac{d}{dx}e^{x-\frac{1}{x}}$.

```
import math
from maker import make_prod, make_const, make_pwr, make_e_expr
from maker import make_plus, make_quot
from tof import tof
from deriv import deriv

def test_03(self):
    print('\n***** Test 03 ************')
    fex1 = make_quot(make_const(-1.0), make_pwr('x', 1.0))
    fex2 = make_e_expr(make_plus(make_pwr('x', 1.0), fex1))
    print(fex2)
    drv = deriv(fex2)
    assert not drv is None
    print(drv)
    drvf = tof(drv)
    assert not drvf is None
    def gt_drvf(x):
        d = (x - 1.0/x)
        return (math.e**d)*(1.0 + 1.0/(x**2))
    err = 0.0001
    for i in range(1, 10):
        print drvf(i), gt_drvf(i)
        assert abs(gt_drvf(i) - drvf(i)) <= err
    print('Test 03: pass')
```

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

```
(2.71828182846^((x^1.0)+(-1.0/(x^1.0))))
((2.71828182846^((x^1.0)+(-1.0/(x^1.0))))*((1.0*(x^0.0))+((((x^1.0)*0.0)+
(-1.0*(-1.0*(1.0*(x^0.0)))))/((x^1.0)^2.0))))
2.0 2.0
5.60211133792 5.60211133792
```

```
15.9910178835 15.9910178835
45.1786496251 45.1786496251
126.370834219 126.370834219
350.981076014 350.981076014
970.048009362 970.048009362
2671.79066202 2671.79066202
7340.4760869 7340.4760869
Test 03: pass
```

Test 04

Let's do $\frac{d}{dx}\left(\frac{3e^{2x}}{1+x^2}\right)$.

```
import math
from maker import make_prod, make_const, make_pwr, make_e_expr
from maker import make_plus, make_quot
from tof import tof
from deriv import deriv

def test_04():
    print('\n***** Test 04 ************')
    n = make_prod(make_const(3.0),
                  make_e_expr(make_prod(make_const(2.0),
                                        make_pwr('x', 1.0))))
    d = make_plus(make_const(1.0), make_pwr('x', 2.0))
    fex = make_quot(n, d)
    print(fex)
    drv = deriv(fex)
    assert not drv is None
    print(drv)
    drvf = tof(drv)
    assert not drvf is None
    def gt_drvf(x):
        n = 6.0*(math.e**(2.0*x))*(x**2 - x + 1.0)
        d = (1 + x**2)**2
        return n/d
    for i in range(-10, 10):
        print drvf(i), gt_drvf(i)
        assert abs(gt_drvf(i) - drvf(i)) <= 0.001
    print('Test 04: pass')
```

Here is the output of `test_04` in the Py shell.

```
***** Test 04 ************
((3.0*(2.71828182846^(2.0*(x^1.0))))/(1.0+(x^2.0)))
```

```
((((1.0+(x^2.0))*(3.0*((2.71828182846^(2.0*(x^1.0)))*(2.0*(1.0*(x^0.0))))))
+(-1.0*((3.0*(2.71828182846^(2.0*(x^1.0))))*(0.0+(2.0*(x^1.0))))))/
((1.0+(x^2.0))^2.0))
1.3456801417e-10 1.3456801417e-10
1.23669972347e-09 1.23669972347e-09
1.16663684088e-08 1.16663684088e-08
1.13753128773e-07 1.13753128773e-07
1.15793045081e-06 1.15793045081e-06
1.24916966506e-05 1.24916966506e-05
0.000146257062684 0.000146257062684
0.0019334266978 0.0019334266978
0.0307702733331 0.0307702733331
0.609008774565 0.609008774565
6.0 6.0
11.0835841484 11.0835841484
39.3106680239 39.3106680239
169.440093267 169.440093267
804.549214496 804.549214496
4105.52468956 4105.52468956
22112.7766281 22112.7766281
124108.762126 124108.762126
719301.727341 719301.727341
4277077.10918 4277077.10918
Test 04: pass
```

Test 05

Let's do $\frac{d}{dx}(lnx)^5$.

```
import math
from maker import make_prod, make_const, make_pwr, make_e_expr
from maker import make_plus, make_quot, make_pwr_expr, make_ln
from tof import tof
from deriv import deriv

def test_05():
    print('\n***** Test 05 ************')
    fex = make_pwr_expr(make_ln(make_pwr('x', 1.0)), 5.0)
    print(fex)
    drv = deriv(fex)
    assert not drv is None
    print(drv)
    drvf = tof(drv)
    assert not drvf is None
    gt = lambda x: (5.0*(math.log(x, math.e)**4))/x
    err = 0.0001
```

```
    for i in range(1, 5):
        print(drvf(i), gt(i))
        assert abs(gt(i) - drvf(i)) <= err
    print('Test 05: pass')
```

Here is the output of `test_05` in the Py shell.

```
***** Test 05 ************
(ln(x^1.0)^5.0)
((5.0*(ln(x^1.0)^4.0))*((1.0/(x^1.0))*(1.0*(x^0.0))))
(0.0, 0.0)
(0.5770877464577086, 0.5770877464577086)
(2.427876323344287, 2.427876323344287)
(4.616701971661668, 4.616701971661668)
Test 05: pass
```

Test 06

Let's do $\frac{d}{dx}xlnx$.

```
import math
from maker import make_prod, make_const, make_pwr, make_e_expr
from maker import make_plus, make_quot, make_pwr_expr, make_ln
from tof import tof
from deriv import deriv

def test_06():
    print('\n***** Test 06 ************')
    fex = make_prod(make_pwr('x', 1.0),
                    make_ln(make_pwr('x', 1.0)))
    print(fex)
    drv = deriv(fex)
    assert not drv is None
    print(drv)
    drvf = tof(drv)
    assert not drvf is None
    gt = lambda x: 1.0 + math.log(x, math.e)
    err = 0.0001
    for i in range(1, 10):
        print(drvf(i), gt(i))
    assert abs(gt(i) - drvf(i)) <= err
    print('Test 06: pass')
```

Here is the output of `test_06` in the Py shell.

```
***** Test 06 ************
((x^1.0)*ln(x^1.0))
(((x^1.0)*((1.0/(x^1.0))*(1.0*(x^0.0))))+(ln(x^1.0)*(1.0*(x^0.0))))
(1.0, 1.0)
(1.6931471805599454, 1.6931471805599454)
(2.09861228866811, 2.09861228866811)
(2.386294361119891, 2.386294361119891)
(2.6094379124341005, 2.6094379124341005)
(2.791759469228055, 2.791759469228055)
(2.9459101490553135, 2.9459101490553135)
(3.0794415416798357, 3.0794415416798357)
(3.1972245773362196, 3.1972245773362196)
Test 06: pass
```

Test 07

Let's do $\frac{d}{dx} ln(xe^x)$.

```
import math
from maker import make_prod, make_const, make_pwr, make_e_expr
from maker import make_plus, make_quot, make_pwr_expr, make_ln
from tof import tof
from deriv import deriv

def test_07():
    print('\n***** Test 07 ************')
    fex0 = make_prod(make_pwr('x', 1.0),
                     make_e_expr(make_pwr('x', 1.0)))
    fex = make_ln(fex0)
    print(fex)
    drv = deriv(fex)
    assert not drv is None
    print(drv)
    drvf = tof(drv)
    assert not drvf is None
    gt = lambda x: (x + 1.0)/x
    err = 0.0001
    for i in range(1, 10):
        print(drvf(i), gt(i))
        assert abs(gt(i) - drvf(i)) <= err
    for i in range(-10, -1):
        print(drvf(i), gt(i))
        assert abs(gt(i) - drvf(i)) <= 0.001
    print('Test 07: pass')
```

Here is the output of `test_07` in the Py shell.

```
***** Test 07 ************
ln((x^1.0)*(2.71828182846^(x^1.0)))
(((1.0/(x^1.0))*(1.0*(x^0.0)))+((1.0/(2.71828182846^(x^1.0)))*
x‘((2.71828182846^(x^1.0))*(1.0*(x^0.0)))))
(2.0, 2.0)
(1.5, 1.5)
(1.3333333333333333, 1.3333333333333333)
(1.25, 1.25)
(1.2, 1.2)
(1.1666666666666667, 1.1666666666666667)
(1.1428571428571428, 1.1428571428571428)
(1.125, 1.125)
(1.1111111111111112, 1.1111111111111112)
(0.9, 0.9)
(0.8888888888888888, 0.8888888888888888)
(0.875, 0.875)
(0.8571428571428572, 0.8571428571428571)
(0.8333333333333334, 0.8333333333333334)
(0.8, 0.8)
(0.75, 0.75)
(0.6666666666666667, 0.6666666666666666)
(0.5, 0.5)
Test 07: pass
```

Test 08

Let's do $\frac{d}{dx} ln|x|$.

```
import math
from maker import make_prod, make_const, make_pwr, make_e_expr
from maker import make_plus, make_quot, make_pwr_expr, make_ln
from maker import make_absv
from tof import tof
from deriv import deriv

def test_08():
    print('\n***** Test 08 ************')
    fex = make_ln(make_absv(make_pwr('x', 1.0)))
    print(fex)
    drv = deriv(fex)
    assert not drv is None
    print(drv)
    drvf = tof(drv)
    assert not drvf is None
    gt = lambda x: 1.0/x
    err = 0.0001
```

```
    for i in range(1, 10):
        print(drvf(i), gt(i))
        assert abs(gt(i) - drvf(i)) <= err
    print('Test 08: pass')
```

Here is the output of `test_08` in the Py shell.

```
***** Test 08 ************
ln|(x^1.0)|
((x^1.0)^-1.0)
(1.0, 1.0)
(0.5, 0.5)
(0.3333333333333333, 0.3333333333333333)
(0.25, 0.25)
(0.2, 0.2)
(0.16666666666666666, 0.16666666666666666)
(0.14285714285714285, 0.14285714285714285)
(0.125, 0.125)
(0.1111111111111111, 0.1111111111111111)
Test 08: pass
```

Problem 2: Logarithmic Differentiation (2 points)

Implement logarithmic differentiation. Specifically, write a function `logdiff(expr)` and save it in `deriv.py`. This function takes a function representation of a product and computes its derivative.

Test 09

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

```
import math
from maker import make_prod, make_const, make_pwr, make_e_expr
from maker import make_plus, make_quot, make_pwr_expr, make_ln
from maker import make_absv
from tof import tof
from deriv import deriv
from deriv import logdiff

def test_09(self):
    print('\n***** Test 09 ************')
    fex = make_prod(make_pwr('x', 1.0),
                    make_prod(make_plus(make_pwr('x', 1.0),
                                        make_const(1.0)),
                              make_plus(make_pwr('x', 1.0),
```

```
                                        make_const(2.0))))
    drv = logdiff(fex)
    assert not drv is None
    print(drv)
    drvf = tof(drv)
    assert not drvf is None
    def gt_drvf(x):
        z = x*(x + 1.0)*(x + 2.0)
        z2 = (1.0/x + 1.0/(x + 1.0) + 1.0/(x + 2.0))
        return z * z2
    err = 0.0001
    for i in range(1, 10):
        print(drvf(i), gt_drvf(i))
        assert abs(gt_drvf(i) - drvf(i)) <= err
    for i in range(-10, -1):
        if i == -1 or i == -2:
             continue
        print(drvf(i), gt_drvf(i))
        assert abs(gt_drvf(i) - drvf(i)) <= err
    print('Test 09: pass')
```

Here is the output of `test_09` in the Py shell. Note that the output of logarithmic differentiation typically has variables in denominators. So be careful to specify the appropriate ranges in your unit tests.

```
***** Test 09 ************
(((x^1.0)*(((x^1.0)+1.0)*((x^1.0)+2.0)))*(((1.0/(x^1.0))*(1.0*(x^0.0)))+
(((1.0/((x^1.0)+1.0))*((1.0*(x^0.0))+0.0))+((1.0/((x^1.0)+2.0))*
((1.0*(x^0.0))+0.0)))))
(11.0, 11.0)
(26.0, 26.0)
(47.0, 46.99999999999999)
(74.0, 74.0)
(107.0, 107.0)
(146.0, 146.0)
(191.0, 191.0)
(242.00000000000003, 242.00000000000003)
(299.0, 299.0)
(242.00000000000003, 242.00000000000003)
(191.0, 191.0)
(146.0, 146.0)
(107.0, 107.0)
(74.0, 74.0)
(46.99999999999999, 47.0)
(26.0, 26.0)
(11.0, 11.0)
```

```
Test 09: pass
```

Test 10

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

```
import math
from maker import make_prod, make_const, make_pwr, make_e_expr
from maker import make_plus, make_quot, make_pwr_expr, make_ln
from maker import make_absv
from tof import tof
from deriv import deriv
from deriv import logdiff

def test_10():
    print('\n***** Test 10 ************')
    fex1 = make_plus(make_pwr('x', 2.0), make_const(1.0))
    fex2 = make_plus(make_pwr('x', 3.0), make_const(-3.0))
    fex3 = make_plus(make_prod(make_const(2.0),
                               make_pwr('x', 1.0)),
                     make_const(5.0))
    fex = make_prod(fex1, make_prod(fex2, fex3))
    print(fex)
    drv = logdiff(fex)
    assert not drv is None
    print(drv)
    drvf = tof(drv)
    assert not drvf is None
    def gt_drvf(x):
        z = (x**2 + 1.0)*(x**3 - 3.0)*(2*x + 5.0)
        z2 = ((2.0*x)/(x**2 + 1) + (3.0*(x**2))/(x**3 - 3.0) \
              + 2.0/(2*x + 5.0))
        return z * z2
    for i in range(1, 10):
        print(drvf(i), gt_drvf(i))
        assert abs(gt_drvf(i) - drvf(i)) <= 0.001
    print('Test 10: pass')
```

Here is the output of `test_10` in the Py shell.

```
***** Test 10 ************
(((x^2.0)+1.0)*(((x^3.0)+-3.0)*((2.0*(x^1.0))+5.0)))
((((x^2.0)+1.0)*(((x^3.0)+-3.0)*((2.0*(x^1.0))+5.0)))*(((1.0/((x^2.0)+1.0))*((2.0*(x^1.0))+0
(6.0000000000000036, 6.0)
(770.0000000000001, 770.0000000000001)
```

```
(5034.0, 5034.0)
(19026.0, 19026.0)
(53894.0, 53894.0)
(127146.0, 127146.0)
(264090.0, 264090.0)
(499273.99999999994, 499274.00000000006)
(877926.0, 877926.0)
Test 10: pass
```

Test 11

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

```
import math
from maker import make_prod, make_const, make_pwr, make_e_expr
from maker import make_plus, make_quot, make_pwr_expr, make_ln
from maker import make_absv
from tof import tof
from deriv import deriv
from deriv import logdiff

def test_11():
    print('\n***** Test 11 ************')
    fex1 = make_pwr_expr(make_plus(make_pwr('x', 1.0),
                                   make_const(1.0)),
                         4.0)
    fex2 = make_pwr_expr(make_plus(make_prod(make_const(4.0),
                                             make_pwr('x', 1.0)),
                                   make_const(-1.0)),
                         2.0)
    fex = make_prod(fex1, fex2)
    print(fex)
    drv = logdiff(fex)
    assert not drv is None
    print(drv)
    drvf = tof(drv)
    def gt_drvf(x):
        z1 = ((x + 1.0)**4.0) * ((4*x - 1.0)**2.0)
        z2 = (4.0/(x + 1.0)) + ( 8.0/(4*x - 1.0))
        return z1 * z2
    for i in range(0, 20):
        print(drvf(i), gt_drvf(i))
        assert abs(gt_drvf(i) - drvf(i)) <= 0.001
    print('Test 11: pass')
```

Here is the output of `test_11` in the Py shell.

```
***** Test 11 ************
((((x^1.0)+1.0)^4.0)*(((4.0*(x^1.0))+-1.0)^2.0))
(((((x^1.0)+1.0)^4.0)*(((4.0*(x^1.0))+-1.0)^2.0))*((4.0*((1.0/((x^1.0)+1.0))*((1.0*(x^0.0))+
(-4.0, -4.0)
(671.9999999999999, 671.9999999999999)
(9828.0, 9828.0)
(53504.0, 53504.0)
(187500.00000000003, 187500.00000000003)
(508895.99999999994, 508895.99999999994)
(1167572.0, 1167572.0)
(2377728.0, 2377728.0)
(4429404.0, 4429404.0)
(7700000.0, 7700000.0)
(12665796.000000002, 12665796.000000002)
(19913472.0, 19913472.0)
(30151628.000000004, 30151628.000000004)
(44222304.0, 44222304.0)
(63112500.0, 63112500.0)
(87965696.0, 87965696.0)
(120093372.0, 120093372.0)
(160986528.0, 160986528.0)
(212327204.0, 212327204.0)
(276000000.0, 276000000.0)
Test 11: pass
```

What to Submit

You definitely need to submit `deriv.py` and `tof.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 `hw04.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!