# CS 3430: S19: SciComp with Py
# Assignment 7
# Antidifferentiation and Basic Image Processing

Vladimir Kulyukin
Department of Computer Science
Utah State University

February 23, 2019

## Learning Objectives

1. Antidifferentiation
2. Splitting, Merging, and Amplifying Images

## Warmup

Practice antidifferentiation on the problems below. For those of you who are familiar with integration, the term **antidifferentiation** is synonymous with **indefinite integration** and **definite antidifferentiation** – with **definite integration**. I use the term **antidifferentiation**, because it is common in scientific computing and numerical analysis, but the terms are completely interchangeable. If and when you feel comfortable with antidifferentiation, move on to Problem 1. We'll turn these problems into your unit tests for Problem 1.

1. $\int x^2 dx$; **answer:** $\frac{1}{3}x^3 + C$, $C$ is real;
2. $\int e^{-2x} dx$; **answer:** $-\frac{1}{2}e^{-2k} + C$, $C$ is real;
3. $\int \sqrt{x} dx$; **answer:** $\frac{2}{3}x^{\frac{3}{2}} + C$, $C$ is real;
4. $\int \frac{1}{x^2} dx$; **answer:** $-\frac{1}{x} + C$, $C$ is real;
5. $\int \frac{1}{x} dx$; **answer:** $ln|x| + C$, $C$ is real;
6. $\int \left(\frac{1}{x^3} + 7e^{5x} + \frac{4}{x}\right) dx$; **answer:** $-\frac{1}{2x^2} + \frac{7}{5}e^{5x} + 4ln|x| + C$, $C$ is real;
7. $\int 4x^3 dx$; **answer:** $x^4 + C$, $C$ is real;
8. $\int (5x-7)^{-2} dx$; **answer:** $-\frac{1}{5}(5x-7)^{-1} + C$, $C$ is real.

9. $\int \frac{3}{2+x} dx$; **answer:** $3ln|2+x| + C$, $C$ is real.

10. $\int (3x+2)^4 \, dx$; **answer:** $\frac{1}{15}(3x+2)^5 + C$, $C$ is real.

# Problem 1: Antidifferentiation (3 points)

Implement the function `antideriv(i)` in the file `antideriv.py` that takes an integrand $i$ represented as a function expression and outputs an antiderivative of $i$ also represented as a function expression. Let's make a simplifying assumption that the value of $C$ in the returned antiderivative is always 0.

Your implementation should handle the following four main cases: 1) $i$ is a constant; 2) $i$ is a power; 3) $i$ is a sum; 4) $i$ is a product in which the first element is a constant. In case 2, let $b$ be the base of $i$ and $d$ be its degree. Then, case 2, in turn, splits into the following three subcases: 2.1) $b$ is a variable and $d$ is a constant; 2.2) $b = e$; 2.3) $b$ is a sum. I gave you the code outline in `antideriv.py` where you should implement and save your code.

## Test 01

Let's do $\int x^2 dx$.

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

def test_01():
    print('\n***** Test 01 ***********')
    fex = make_pwr('x', 2.0)
    print(fex)
    afex = antideriv(fex)
    assert not afex is None
    def gt(x): return (1.0/3.0)*(x**3.0)
    afexf = tof(afex)
    assert not afexf is None
    err = 0.0001
    for i in range(1, 101):
        assert abs(afexf(i) - gt(i)) <= err
    print(afex)
    print('Test 01: pass')
```

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

```
***** Test 01 ***********
(x^2.0)
(((1.0/3.0)*(x^3.0))+0.0)
Test 01: pass
```

## Test 02

Let's do $\int e^{-2x} dx$.

```
def test_02():
    print('\n***** Test 02 ***********')
    fex = make_e_expr(make_prod(make_const(-2.0),
                                make_pwr('x', 1.0)))
    print(fex)
    afex = antideriv(fex)
    assert not afex is None
    def gt(x): return (-0.5)*(math.e**(-2.0*x))
    afexf = tof(afex)
    assert not afexf is None
    err = 0.0001
    for i in range(0, 101):
        assert abs(afexf(i) - gt(i)) <= err
    print(afex)
    print('Test 02: pass')
```

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

```
***** Test 02 ***********
(2.71828182846^(-2.0*(x^1.0)))
(((1.0/-2.0)*(2.71828182846^(-2.0*(x^1.0))))+0.0)
Test 02: pass
```

## Test 03

Let's do $\int \sqrt{x}$.

```
def test_03():
    print('\n***** Test 03 ***********')
    fex = make_pwr('x', 0.5)
    print(fex)
    afex = antideriv(fex)
    assert not afex is None
    def gt(x): return (2.0/3.0)*(x**(3.0/2.0))
    afexf = tof(afex)
    assert not afexf is None
```

```
    err = 0.0001
    for i in range(1, 101):
        assert abs(afexf(i) - gt(i)) <= err
    print(afex)
    print('Test 03: pass')
```

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

```
***** Test 03 ***********
(x^0.5)
(((1.0/1.5)*(x^1.5))+0.0)
Test 03: pass
```

## Test 04

Let's do $\int \frac{1}{x^2} dx$.

```
def test_04():
    print('\n***** Test 04 ***********')
    fex = make_pwr('x', -2.0)
    print(fex)
    afex = antideriv(fex)
    assert not afex is None
    def gt(x): return -1.0/x
    afexf = tof(afex)
    assert not afexf is None
    err = 0.0001
    for i in range(1, 101):
        assert abs(afexf(i) - gt(i)) <= err
    print(afex)
    print('Test 04: pass')
```

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

```
***** Test 04 ***********
(x^-2.0)
(((1.0/-1.0)*(x^-1.0))+0.0)
Test 04: pass
```

## Test 05

Let's do $\int \frac{1}{x} dx$.

```
def test_05():
    print('\n***** Test 05 ***********')
    fex = make_pwr('x', -1.0)
    print(fex)
    afex = antideriv(fex)
    assert not afex is None
    afexf = tof(afex)
    assert not afexf is None
    def gt(x): return math.log(abs(x), math.e)
    err = 0.0001
    for i in range(1, 101):
        assert abs(afexf(i) - gt(i)) <= err
    for i in range(-100, 0):
        assert abs(afexf(i) - gt(i))
    print('Test 05: pass')
```

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

```
***** Test 05 ***********
(x^-1.0)
(ln|(x^1.0)|+0.0)
Test 05: pass
```

## Test 06

Let's do $\int \left(\frac{1}{x^3} + 7e^{5x} + \frac{4}{x}\right) dx$. I did not use numerical comparison assertions in this unit test, because the values become huge pretty fast. Instead, I just printed the first ten outputs of the ground truth function and the antiderivative function.

```
def test_06():
    print('\n***** Test 06 ***********')
    fex1 = make_pwr('x', -3.0)
    fex2 = make_prod(make_const(7.0),
                     make_e_expr(make_prod(make_const(5.0),
                                           make_pwr('x', 1.0))))
    fex3 = make_prod(make_const(4.0),
                     make_pwr('x', -1.0))
    fex4 = make_plus(fex1, fex2)
    fex = make_plus(fex4, fex3)
```

```
    print(fex)
    afex = antideriv(fex)
    assert not afex is None
    print(afex)
    def gt(x):
        v1 = -0.5*(x**(-2.0))
        v2 = (7.0/5.0)*(math.e**(5.0*x))
        v3 = 4.0*(math.log(abs(x), math.e))
        return v1 + v2 + v3
    afexf = tof(afex)
    assert not afexf is None
    err = 0.0001
    for i in range(1, 10):
        print(afexf(i), gt(i))
        #assert abs(afexf(i) - gt(i)) <= err
    print('Test 06: pass')
```

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

```
***** Test 06 ***********
(((x^-3.0)+(7.0*(2.71828182846^(5.0*(x^1.0)))))+(4.0*(x^-1.0)))
(((((((1.0/-2.0)*(x^-2.0))+0.0)+((7.0*(((1.0/5.0)*
(2.71828182846^(5.0*(x^1.0))))+0.0))+0.0))+0.0)+
((4.0*(ln|(x^1.0)|+0.0))+0.0))+0.0)
(207.2784227436072, 207.2784227436072)
(30839.699701451627, 30839.699701451624)
(4576628.66035455, 4576628.66035455)
(679231279.0876331, 679231279.087633)
(100806859078.75783, 100806859078.75781)
(14961064414141.379, 14961064414141.377)
(2220418833238806.8, 2220418833238806.5)
(3.295393735718273e+17, 3.2953937357182726e+17)
(4.890797948047902e+19, 4.8907979480479015e+19)
Test 06: pass
```

## Test 07

Let's do $\int 4x^3 dx$. Then let's do $\frac{d}{dx} \int 4x^3 dx$ to see if we can get $4x^3$ back.

```
def test_07():
    print('\n***** Test 07 ***********')
    fex = make_prod(make_const(4.0), make_pwr('x', 3.0))
    print(fex)
    afex = antideriv(fex)
```

```
    assert not afex is None
    print(afex)
    fexf = tof(fex)
    assert not fexf is None
    fex2 = deriv(afex)
    assert not fex2 is None
    print(fex2)
    fex2f = tof(fex2)
    assert not fex2f is None
    err = 0.0001
    for i in range(11):
        assert abs(fexf(i) - fex2f(i)) <= err
    print('Test 07: pass')
```

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

```
***** Test 07 ***********
(4.0*(x^3.0))
((4.0*(((1.0/4.0)*(x^4.0))+0.0))+0.0)
((4.0*((((1.0/4.0)*(4.0*(x^3.0)))+((x^4.0)*0.0))+0.0))+0.0)
Test 07: pass
```

## Test 08

Let's do $\int (5x-7)^{-2} dx$. In addition to comparing the antiderivative with the ground truth, let's convert $\frac{d}{dx} \int (5x-7)^{-2} dx$ to a function and see if that function agrees with $(5x-7)^{-2}$ on a range of values.

```
def test_08():
    print('\n***** Test 08 ***********')
    fex1 = make_plus(make_prod(make_const(5.0),
                               make_pwr('x', 1.0)),
                     make_const(-7.0))
    fex = make_pwr_expr(fex1, -2.0)
    print(fex)
    afex = antideriv(fex)
    assert not afex is None
    print(afex)
    afexf = tof(afex)
    err = 0.0001
    def gt(x):
        return (-1.0/5.0)*((5*x - 7.0)**-1)
    for i in range(1, 100):
        assert abs(afexf(i) - gt(i)) <= err
```

```
    fexf = tof(fex)
    assert not fexf is None
    fex2 = deriv(afex)
    assert not fex2 is None
    print(fex2)
    fex2f = tof(fex2)
    assert not fex2f is None
    for i in range(1, 100):
        assert abs(fexf(i) - fex2f(i)) <= err
    print('Test 08: pass')
```

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

```
***** Test 08 ***********
(((5.0*(x^1.0))+-7.0)^-2.0)
(-0.2*(((5.0*(x^1.0))+-7.0)^-1.0))
(-0.2*((-1.0*(((5.0*(x^1.0))+-7.0)^-2.0))*
((5.0*(1.0*(x^0.0)))+0.0)))
Test 08: pass
```

## Test 09

Let's do $\int \frac{3}{2+x} dx$.

```
def test_09():
    print('\n***** Test 09 ***********')
    fex0 = make_plus(make_pwr('x', 1.0), make_const(2.0))
    fex1 = make_pwr_expr(fex0, -1.0)
    fex = make_prod(make_const(3.0), fex1)
    print(fex)
    afex = antideriv(fex)
    err = 0.0001
    afexf = tof(afex)
    def gt(x):
        return 3.0*math.log(abs(2.0 + x), math.e)
    for i in range(1, 101):
        assert abs(afexf(i) - gt(i)) <= err
    assert not afex is None
    print(afex)
    fexf = tof(fex)
    assert not fexf is None
    fex2 = deriv(afex)
    assert not fex2 is None
    print(fex2)
```

```
    fex2f = tof(fex2)
    assert not fex2f is None
    for i in range(1, 1000):
        assert abs(fexf(i) - fex2f(i)) <= err
    print('Test 09: pass')
```

Here is the output of `test_09` in the Py shell.

```
***** Test 09 ***********
3.0*(((x^1.0)+2.0)^-1.0))
((3.0*(1.0*ln|((x^1.0)+2.0)|))+0.0)
((3.0*(1.0*((((x^1.0)+2.0)^-1.0)*((1.0*(x^0.0))+0.0))))+0.0)
Test 09: pass
```

## Test 10

Let's do $\int (3x+2)^4\,dx$.

```
def test_10():
    print('\n***** Test 10 ***********')
    fex0 = make_prod(make_const(3.0), make_pwr('x', 1.0))
    fex1 = make_plus(fex0, make_const(2.0))
    fex = make_pwr_expr(fex1, 4.0)
    print(fex)
    afex = antideriv(fex)
    assert not afex is None
    print(afex)
    afexf = tof(afex)
    err = 0.0001
    def gt(x):
        return (1.0/15)*((3*x + 2.0)**5)
    for i in range(1, 10):
        assert abs(afexf(i) - gt(i)) <= err
    fexf = tof(fex)
    assert not fexf is None
    fex2 = deriv(afex)
    assert not fex2 is None
    print(fex2)
    fex2f = tof(fex2)
    assert not fex2f is None
    for i in range(1, 1000):
    print('Test 10: pass')
```

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

```
***** Test 10 ***********
(((3.0*(x^1.0))+2.0)^4.0)
(0.0666666666667*(((3.0*(x^1.0))+2.0)^5.0))
(0.0666666666667*((5.0*(((3.0*(x^1.0))+2.0)^4.0))*
((3.0*(1.0*(x^0.0)))+0.0)))
Test 10: pass
```

## Problem 2: Splitting, Merging, and Amplifying Images (2 points)

This problem will kick off our journey into OpenCV. The primary objective is to ensure that your installation of OpenCV is up and running.

The archive `hw07.zip` contains the folder `images` with 50 images of roads and highways in Cache Valley. Let's call the extension of an image file (e.g., `.jpg`) a file type or ftype. Write the function `read_img_dir(ftype, imgdir)` that takes two strings, `ftype` and `imgdir`, and returns a list of 2-tuples such that the first element of each 2-tuple of a path to an image of a given ftype in the directory `imgdir` and the second element is the 2D numpy matrix of that image. For example, the `ftype` can be `'.jpg'` or `'.png'`).

When you work on this function you will likely need to traverse a directory and return the absolute path names to each file that ends in a given extension (i.e., ftype). Here is one way of doing it with a Python generator. There are other ways of listing files in a directory in Python, but I prefer to use generators, because they are more memory efficient, which allows one to process terrabytes of data.

```
def generate_file_names(ftype, rootdir):
    '''
recursively walk dir tree beginning from rootdir
and generate full paths to all files that end with ftype.
sample call: generate_file_names('.jpg', /home/pi/images/')
    '''
    for path, dirlist, filelist in os.walk(rootdir):
        for file_name in filelist:
            if not file_name.startswith('.') and \
               file_name.endswith(ftype):
                yield os.path.join(path, file_name)
        for d in dirlist:
            generate_file_names(ftype, d)
```

Let's test this function.

```
>>> imglist = read_img_dir('.jpg', '/home/pi/images/')
>>> len(imglist)
50
```

Below we see that the first element of the first 2-tuple is an image path and the second is a 2D numpy array corresponding to the image saved in the specified path.

```
>>> imglist[0][0]
'/home/pi/images/output14889.jpg'
>>> imglist[0][1]
array([[[ 78, 140, 150],
        [ 62, 127, 136],
        [ 55, 124, 134],
        ...,
        [ 87, 137, 149],
        [ 89, 139, 151],
        [ 89, 139, 151]]], dtype=uint8)
```

Here is how we can test the dimensions of the first image.

```
>>> imglist[0][1].shape
(1080, 1920, 3)
```

So, it is a 1080 by 1920 image with 3 channels.

Write the function `grayscale(i, imglist)` that an integer $i$ and a list of 2-tuples returned by `read_img_dir` and shows two image windows. The first image window shows the i-th image (i.e., the 2D numpy matrix of the i-th 2-tuple in `imglist`). The title of the first window is the full path to the image (i.e., the first element of the i-th 2-tuple). The second image window shows the grayscaled version of the 2D numpy matrix. The title of that window should read "Grayscaled." These are pretty big images so, depending on the size of your screen, you may be able to see only parts of the windows.

Save your implementation in `hw07_s19.py`

Write the function `split_merge(i, imglist)` that takes the same arguments as `grayscale`. This function splits the image in the i-th 2-tuple in `imglist` into the blue, green, and red channels, and then displays each channel in a separate window along with the window with the original image. The title of the window with the original image is the image path. The red channel is displayed in the window whose title is "Red," the blue channel – in the window whose title is "Blue," and the green channel – in the window whose title is "Green." Each

window should display the image not in grayscale but in the corresponding color, i.e., red, green, and blue.

Save your implementation in `hw07_s19.py`

Write the function `amplify(i, imglist, c, amount)`. The first two arguments are the same as in `grayscale` and `split_merge`. The third argument, `c`, is a string whose value is `'b'`, `'g'`, or `'r'`. This argument specifies the channel. The fourth argument, `amount`, is an integer that gives the amount of amplification with which the channel `c` will be amplified. This function displays two windows. The first window displays the image in the i-th tuple of `imglist`. The title of the first window is the full path to the image. The second window displays the amplified image. The title of the second window is "Amplified."

Save your implementation in `hw07_s19.py`

## What to Submit

You definitely need to submit `antideriv.py` and `hw07_s19.py` You should also submit all the files needed to run your code. Zip your entire working directory in `hw07.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!