CS 3430: SciComp with Py
Assignment 07
Central Divided Difference, Richardson Extrapolation, Romberg Integration

Vladimir Kulyukin
Department of Computer Science
Utah State University

February 22, 2020

Learning Objectives

1. Central Divided Difference (CDD)
2. Richardson Extrapolation
3. Romberg Integration

Introduction

In this assignment, we'll implement several CDD formulas to approximate $f'(x)$, use these formulas to implement the first, second, third, and fourth Richardson extrapolations for differentiation to approximate $f'(x)$ and then implement the Romberg Integration method to approximate $\int_a^b f(x)dx$.

You'll code up your solutions to Problem 01 in `cdd.py`, to Problem 02 in `rxp.py`, and to Problem 03 in `rmb.py`. Included in the zip is `cs3430_s20_hw07_uts.py` with my unit tests for this assignment. Remember not to run all unit tests at once. Proceed one unit test at a time. Easy does it.

Problem 01: (1 point)

The file `cdd.py` contains the stubs of four static methods you'll implement to approximate $f'(x)$ and $f''(x)$.

```
class cdd(object):

    @staticmethod
    def drv1_ord2(f, x, h):
        ### Table 6.3 formula 1, p. 339 in CDD handout
        pass

    @staticmethod
    def drv1_ord4(f, x, h):
        ### Table 6.4 formula 1, p. 339 in the CDD handout
        pass

    @staticmethod
```

```
def drv2_ord2(f, x, h):
    ### Table 6.3, formula 2, p. 339 in CDD handout
    pass

@staticmethod
def drv2_ord4(f, x, h):
    ### Table 6.4, formula 2, p. 339 in CDD handout
    pass
```

Take a look at the reading handout "Central-Difference Formulas" that I distributed before and after the lecture on the CDD. If you didn't attend the lecture, you can stop by my office (Old Main 402D) to pick it up (please email me first so that I can make a copy) or ask a friend who attended the lecture to share the handout and the lecture notes with you.

Go to Tables 6.3 and 6.4 on p. 339 in the handout. Implement the first formula in Table 6.3 as `cdd.drv1_ord2(f, x, h)` and the first formula in Table 6.4 as `cdd.drv1_ord4(f, x, h)`. Implement the second formula in Table 6.3 as `cdd.drv2_ord2(f, x, h)` and the second formula in Table 6.4 as `cdd.drv2_ord4(f, x, h)`. The first two methods take a Python function `f`, a real number `x`, and a real number `h`, and approximate $f'(x)$ at the given value of `h`. The second two methods take a Python function `f`, a real number `x`, and a real number `h`, and approximate $f''(x)$ at the given value of `h`. All four methods return `np.longdouble` values for greater precision.

Let's run some unit tests from `cs3430_s20_hw07_uts.py`. In the first unit test, we use `cdd.drv1_ord2()` to approximate $cos'(x)$ at $x = 0.8$ and $h = 0.01$ and compare the returned value with the ground truth of $-sin(0.8)$ at the error level of 0.0001.

```
def test_hw07_prob01_ut01(self):
    f = lambda x: math.cos(x)
    ## 1. compute approximate value av
    av = cdd.drv1_ord2(f, 0.8, 0.01)
    ## 2. compute true value tv
    tv = -math.sin(0.8)
    ## 3. set error level
    err = 0.0001
    ## 4. make sure you return np.longdouble for max precision
    assert isinstance(av, np.longdouble)
    ## 5. compare av and tv at specified error level
    assert abs(av - tv) <= err
    print('av={}'.format(av))
    print('tv={}'.format(tv))
```

Running this unit test produces the following output on my laptop (Python 3.6.7, Ubuntu 18.04 LTS).

```
***** CS3430: S20: HW07: Problem 01: Unit Test 01 ************
av=-0.7173441350244558
tv=-0.7173560908995228
CS 3430: S20: HW07: Problem 01: Unit Test 01: pass
```

Unit test 15 for Problem 01 uses `tof.tof()` and `parser.parse_sum()` from Assignments 05 and 06 and `cdd.drv1_ord4()` to approximate the first derivative of $f(x) = 4x^5 - 3x^{-3} + 10x^2 - 5x$ at $x = 0.5$ and $h = 0.0001$.

```
    def test_hw07_prob01_ut15(self):
        f = lambda x: 4*(x**5.0) - 3.0*(x**-3.0) + 10.0*(x**2) - 5.0*x
        ## rename parser if necessary.
        df = tof.tof(drv.drv(parser.parse_sum('4x^5 - 3x^-3 + 10x^2 - 5x^1')))
        x = 0.5
        h = 0.0001
        av = cdd.drv1_ord4(f, x, h)
        tv = df(x)
        err = 0.0000001
        assert abs(tv - 150.25) <= err
        assert isinstance(av, np.longdouble)
        assert abs(av - tv) <= err
        print('av={}'.format(av))
        print('tv={}'.format(tv))
```

Running this unit test for Problem 01 produces the following output on my laptop.

```
***** CS3430: S20: HW07: Problem 01: Unit Test 15 ************
av=150.2500000000051
tv=150.25
CS 3430: S20: HW07: Problem 01: Unit Test 15: pass
```

Unit test 17 for Problem 01 uses `cdd.drv2_ord2()` to approximate the second derivative of $f(x) = cos(x)$ at $x = 0.8$ and $h = 0.0001$.

```
    def test_hw07_prob01_ut17(self):
        f = lambda x: math.cos(x)
        ## df is the ground truth second derivative function
        df = lambda x: -math.cos(x)
        x = 0.8
        h = 0.001
        av = cdd.drv2_ord2(f, x, h)
        tv = df(x)
        err = 0.00001
        assert isinstance(av, np.longdouble)
        assert abs(av - tv) <= err
        print('av={}'.format(av))
        print('tv={}'.format(tv))
```

Running unit test 17 for Problem 01 produces the following output.

```
***** CS3430: S20: HW07: Problem 01: Unit Test 17 ************
av=-0.6967066512597597
tv=-0.6967067093471654
CS 3430: S20: HW07: Problem 01: Unit Test 17: pass
```

Problem 02: (1 point)

The file `rxp.py` contains the stubs of three static methods you'll implement to do Richardson extrapolation for differentiation. Each method takes a Python function `f`, a first derivative CDD approximation method from the previous problem (e.g., `cdd.drv1_ord2`), a real number `x`, and a real number `h` and returns the `np.longdouble` value of the n-th Richardson extrapolation for

differentiation to approximate $f'(x)$ at `x` and `h`. Specifically, `drv1(f, cdd, x, h)` returns the 1st Richardson for differentiation, `drv2(f, cdd, x, h)` – the 2nd Richardson for differentiation, `drv3(f, cdd, x, h)` – the 3rd Richardson for differentiation, and `drv4(f, cdd, x, h)` – the 4th Richardson for differentiation. Recall that the first Richardson for differentiation is just the value of the passed CDD method applied to `f`, `x`, and `h`.

I know, I know – this technical jargon may get confusing pretty fast. To give you a helping hand and get you started, I've defined the 2nd Richardson for you. As you can see from the definition, the 2nd Richardson for differentiation is defined in terms of the 1st Richardson for differentiation. Similarly, the 3rd Richard is defined in terms of the 2nd and the 4th – in terms of the 3rd. Note that the denominator of the second element in the 2nd Richardson is equal to 3 and recall from our class discussion that the denominator in the second element of the 3rd Richardson is equal to 15 and the denominator in the second element of the 4th Richadson is equal to 63.

```
class rxp(object):

    @staticmethod
    def drv1(f, cdd, x, h):
        pass

    @staticmethod
    def drv2(f, cdd, x, h):
        x1 = rxp.drv1(f, cdd, x, h/2.0)
        x2 = rxp.drv1(f, cdd, x, h)
        return  np.longdouble(x1 + (x1 - x2)/3.0)

    @staticmethod
    def drv3(f, cdd, x, h):
        pass

    @staticmethod
    def drv4(f, cdd, x, h):
        pass
```

The unit test module contains 10 unit tests for Problem 02. Let's run the 10-th unit test. This unit test approximates the first derivative of $f(x) = 4x^5 - 3x^{-3} + 10x^2 - 5x$ at $x = 0.5$ and $h = 0.0001$ with the 3rd Richardson `rxp.drv3` and `cdd.drv1_ord2`.

```
    def test_hw07_prob02_ut10(self):
        f = lambda x: 4*(x**5.0) - 3.0*(x**-3.0) + 10.0*(x**2) - 5.0*x
        ## rename parser if necessary.
        df = tof.tof(drv.drv(parser.parse_sum('4x^5 - 3x^-3 + 10x^2 - 5x^1')))
        x = 0.5
        h = 0.0001
        av = rxp.drv3(f, cdd.drv1_ord2, x, h)
        tv = df(x)
        err = 0.0000001
        assert abs(tv - 150.25) <= err
        assert isinstance(av, np.longdouble)
        assert abs(av - tv) <= err
        print('av={}'.format(av))
        print('tv={}'.format(tv))
```

Running this unit test produces the following output.

```
***** CS3430: S20: HW07: Problem 02: Unit Test 10 ************
av=150.25000000021174
tv=150.25
CS 3430: S20: HW07: Problem 02: Unit Test 10: pass
```

Problem 03: (1 point)

The file `rmb.py` contains the stub of one static method `rmb.rjl(f, a, b, j, l)` that you'll implement compute elements of the Romberg lattice $R_{j,l}$ to approximate $\int_a^b f(x)dx$. This method takes a Python function `f`, two real numbers `a` and `b`, and two positive integers `j` and `l` and returns the `np.longdouble` value of $R_{j,l}$ (i.e., the j-th Romberg at level l). o make this method run faster, you may want to use the dynamic programming trick we discussed in Lecture 13.

I've written 11 unit tests for this problem in the unit test file. Let's run the last one where we approximate $\int_{51}^{100} 3x^2 + 5x - 10dx$ with $R_{15,14}$ and compare it with the ground truth computed by the anti-derivative of this function.

```
    def test_hw07_prob03_ut11(self):
        ## f(x)
        def f(x):
            return 3.0*x**2.0 + 5.0*x - 10.0
        ## anti-derivative of f(x)
        def antidf(x):
            return x**3.0 + 2.5*x**2.0 - 10.0*x
        err = 0.000001
        tv  = antidf(100.0) - antidf(51.0)
        ## Approximating integral of f(x) on [a=51, b=100] with R[15,14].
        av  = rmb.rjl(f, 51, 100, 15, 14)
        assert abs(tv - av) <= err
        print('av = {}'.format(av))
        print('tv = {}'.format(tv))
```

Running it on my laptop produces the following output.

```
***** CS3430: S20: HW07: Problem 03: Unit Test 11 ************
av = 885356.50000002
tv = 885356.5
CS 3430: S20: HW07: Problem 03: Unit Test 11: pass
```

What To Submit

Submit your code in `cdd.py`, `rxp.py`, and `rmb.py`. It'll be easiest for us to grade your code if you place all the files you used to run the unit tests (i.e., `var.py`, `const.py`, `pwr.py`, `plus.py`, `prod.py`, `pwr.py`, `maker.py`, `parser.py`, `tof.py`, `cdd.py`, `rxp.py`, and `rmb.py`) into one directory, zip it into `hw07.zip`, and upload your zip in Canvas.

Happy Hacking!