CS 3430: S22: Scientific Computing
Assignment 07
Central Divided Difference, Richardson Extrapolation, Romberg Integration

Vladimir Kulyukin
Department of Computer Science
Utah State University

February 26, 2022

Learning Objectives

1. Central Divided Difference (CDD)
2. Richardson Extraploation (RE)
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 1 in `cdd.py`, to Problem 2 in `rxp.py`, and to Problem 3 in `rmb.py`. Included in the zip is `cs3430_s22_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 0: (0 points)

Review the slides for Lectures 13 and 14 (or you class notes if you attend my F2F lectures) and become comfortable with the Central Divided Difference, Richardson Extrapolation, the true error, and the absolute relative true error, the Trapezoidal Rule of definite integral approximation, and Romberg Integration.

Problem 1: CDD (1 point)

The file `cdd.py` contains the stubs of four static methods you'll implement to approximate $f'(x)$ and $f''(x)$. Let's take a look at the reading handout "Central-Difference Formulas" in `CDD.pdf` included in the zip. It contains the formulas that we discussed in class. This reading handout will help you if you didn't attend the last two lectures.

Go to Tables 6.3 and 6.4 on p. 339 in the handout. Note that in these formulas f_{-1} refers to $f(x-h)$ and f_1 refers to $f(x+h)$. Analogously, f_{-2} refers to $f(x-2h)$ and f_2 refers to $f(x+2h)$. We'll use these formulas to approximate first and second derivatives at a given point x in unit tests. Here's

how you can translate the notation in CDD.pdf to the notation that we used in Lecture 13. Slide 5 in Lecture 13 contains the following formula (with the truncation error $O(h^2)$ omitted, because we don't need it here).

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h}.$$

In the notation used in CDD.pdf, the same formula is stated in Table 6.3 on p. 339 as

$$f'(x_0) = \frac{f_1 - f_{-1}}{2h}.$$

Of course, to be precise, we need to replace $=$ with $\approx$ in both formulas, but let's stick with $=$ for the sake of simplicity, because *all* values we're computing in this assignment are approximations.

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)`. These two methods take a Python function `f`, a real number `x`, and a real number `h` (i.e., the step size), and approximate $f'(x)$ at the given value of `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)`. These 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 should return `np.longdouble` values for greater precision.

Let's run some unit tests from `cs3430_s22_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). Your values may be slightly different but they should pass this unit test at this error level.

```
***** CS3430: S22: HW07: Problem 01: Unit Test 01 ************
av=-0.7173441350244558
tv=-0.7173560908995228
CS 3430: S22: HW07: Problem 01: Unit Test 01: pass
```

Unit test 15 for Problem 1 uses `tof.tof()` and `poly_parser.parse_sum()` from Assignments 5 and 6 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
        df = tof.tof(drv.drv(poly_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 1 produces the following output on my laptop.

```
***** CS3430: S22: HW07: Problem 01: Unit Test 15 ************
av=150.2500000000051
tv=150.25
CS 3430: S22: HW07: Problem 01: Unit Test 15: pass
```

Unit test 17 for Problem 1 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.

```
***** CS3431: S22: HW07: Problem 01: Unit Test 17 ************
av=-0.6967066512597597
tv=-0.6967067093471654
CS 3430: S22: HW07: Problem 01: Unit Test 17: pass
```

Problem 2: RE (1 point)

The file `rxp.py` contains the stubs of the static methods you'll implement to do Richardson Extrapolation (RE) 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 required RE to approximate $f'(x)$ at `x` and `h`. As the names suggest, `drv1(f, cdd, x, h)` does the 1st RE and `drv2(f, cdd, x, h)`

does the 2nd RE. Recall (see Slide 14 in Lecture 13) that the first RE is just the value of the passed CDD method applied to `f`, `x`, and `h`.

The unit test module contains 6 unit tests for Problem 02. Let's run the 6-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 2nd RE `rxp.drv2` and `cdd.drv1_ord2`.

```
def test_hw07_prob02_ut06(self):
    f = lambda x: 4*(x**5.0) - 3.0*(x**-3.0) + 10.0*(x**2) - 5.0*x
    df = tof.tof(drv.drv(poly_parser.parse_sum('4x^5 - 3x^-3 + 10x^2 - 5x^1')))
    x = 0.5
    h = 0.0001
    av = rxp.drv2(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))
```

When I run it on my laptop, I get the following output.

```
***** CS3430: S22: HW07: Problem 02: Unit Test 06 ************
av=150.24999999988964
tv=150.25
CS 3430: S22: HW07: Problem 02: Unit Test 06: pass
```

Problem 3: Romberg (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). To make this method run faster, I recommend that you use the dynamic programming trick we discussed in Lecture 14 (I wrote it up on Slide 15 in Lecture 14).

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
    ## this is the ground truth (true value)
    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))
```

I get the following output on my laptop. Again, the approximate value (i.e., `av`) may be different on your computer, but it should pass the unit test at the given error level.

```
***** CS3430: S22: HW07: Problem 03: Unit Test 11 ************
av = 885356.50000002
tv = 885356.5
CS 3430: S22: 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`, `poly_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!