CS 3430: S24: Scientific Computing
Takehome Midterm 02

Vladimir Kulyukin
Department of Computer Science
Utah State University

March 27, 2024

Instructions

1. This takehome exam has 10 problems worth a total of 10 points. You may use your class notes, my lecture PDFs and code samples in Canvas. You may use your own homework solutions. You may not use any other materials (digital or paper).

2. Save your solutions in `cs3430_s24_midterm02.py` and submit this file (and, if necessary, other files – see item 7 below) **in Canvas by 11:59pm on April 1, 2024**. And, no, this deadline is not a joke!

3. Write your name and A-number in `cs3430_s24_midterm02.py`.

4. You may not talk to anyone when you are working on this exam orally, digitally, in writing, or telepathically.

5. You may use your interactive Python IDE, including the Python documentation that comes with it.

6. As stated above, you may use your solutions to the previous assignments. For example, you can do imports from your previous solutions as

```
from tiny_de import tiny_de
from nra import nra
from cs3430_s24_hw06 import *
from rxp import rxp
from cdd import cdd
```

   and then use your implementation of functions and class methods from these modules to solve the midterm problems.

7. **Remember to include in your submission zip all the Python files you import from.** E.g., if you import from your `cs_3430_s24_hw06.py`, `cdd.py`, `nra.py`, or `rxp.py`, include these files in the zip. When I run unit tests on your submission, I will put all your files into the same working directory with your `cs3430_s24_midterm02.py`. If something is not included, the unit tests may fail, and I will have to deduct points.

8. You may not use any third party libraries in this exam, except the ones we have used in this class, e.g., `numpy`, `decimal`, `sympy`, `PIL`, etc. You may use **only** your own solutions to previous/current assignments. My unit tests are in `cs3430_s24_midterm02_uts.py` to test your solutions.

9. If you can, do me a favor and write below your name and A-number in `cs3430_s24_midterm02.py` how much time you spent on this exam. I will not make it public anywhere. This is only for me to assess the easiness/difficulty/reasonableness of the exam.

10. I wish you best of luck and, as always, Happy Hacking!

Problem 1 (1 point)

Implement the function `diff_file(file_path)` that takes a path to a file where each line contains one polynomial written in the convention we established when implementing our tiny differentiation engine (`tiny_de`) and returns a list of the lambdified first derivatives of all polynomials in the file. Remember our polynomial chant: real-variable-caret-real! Recall that, under this convention, we use only `'+'` to connect the summands of the polynomial. If the coefficient of a specific summand is negative, we put `'-'` right before the coefficient. Each summand is in the form of a real, a variable (we use only `'x'` for variable names), the caret `'^'`, and another real that specifies the degree of the variable. The text `'x^2'` parses into a `pwr` object whose base is the variable with the name of `'x'` and whose degree is the constant with the value of 2. The text `'5x^2'` parses into a product object whose first multiplicand is a `constant` object whose value is 5 and whose second multiplicand is a `pwr` object.

Your implementation of the function `diff_file(file_path)` should use the appropriate methods of `tiny_de`. The 3 unit tests for this problem in `cs3430_s24_midterm02_uts.py` uses three files of polynomials, i.e., `polys_01.txt`, `polys_02.txt`, and `polys_03.txt`, included in the zip. Each unit test differentiates and lambdifies each polynomial with `sympy` and then compares the outputs of `sympy` functions with those of the functions computes by `diff_file(file_path)`.

Support materials: Lectures 8, 10; HW05.

Problem 2 (1 point)

Implement the function `zero_root(poly_text, num_iters=5)` that takes a string `poly_text` and returns a zero root of the polynomial represented in the string according to our convention, e.g., `'1x^5 - 171x^0'`, by running the Newton-Raphson algorithm for the number of iterations specified by the second keyword parameter.

Support materials: Slides 20–22, Lecture 11; HW05.

Problem 3 (1 point)

Implement the function `detect_pil_edge_pixels(in_fp, out_fp, default_delta=1.0, magn_thresh=20)` that detects all edge pixels in the PIL image in the file `in_fp` and saves the image with the detected edge pixels in the file `out_fp`. This function applies the edge pixel detection algorithm from Lecture 12 (cf. Slide 07 in `cs3430_s24_lec12_edge_detection.pdf` in the zip of Lecture 12 in Canvas).

This PIL image saved in `out_fp` is a one channel PIL image where the edge pixels are labeled as white (i.e., their pixel value is 255) and non-edge pixels are labeled as black (i.e., their pixel value is 0). The keyword parameter `default_delta` is the threshold value for computing the vertical and horizontal luminosity changes at pixels. The keyword parameter `magn_thresh` specifies the value of the gradient's magnitude at a pixel for the pixel to qualify as an edge pixel. When you work on this function, remember that in PIL individual pixels are referenced in a column-first manner, i.e., a pixel at (x, y) is at column x and row y.

Each unit test for this problem `cs3430_s24_midterm02_uts.py` uses a separate image in the directory `imgs`. The directory `out_imgs` contains the images that my implementation generated for each unit test. The unit tests, when you run them, simply save your output images in the directory `output_imgs`. Your output images should look similar to mine.

Support materials: Lectures 12, 13; HW06.

Problem 4 (1 point)

Implement the first formula in Table 6.3 in the included file `CDD.pdf` 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 the step size `h`, e.g., $h = 0.01$.

Support materials: Lectures 12, 13; HW06.

Problem 5 (1 point)

Implement the second formula in Table 6.3 in `CDD.pdf` as `cdd_drv2_ord2(f, x, h)` and the second formula in Table 6.4 as `cdd_drv2_ord4(f, x, h)` to approximate $f''(x)$. 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 the step size `h`.

The unit tests for Problems 4 and 5 compare the outputs of these two Central Divided Difference (CDD) approximations with the values obtained by `sympy`.

Support materials: Lectures 12, 13; HW06.

Problem 6 (1 point)

Implement the function `romberg_integral(f, a, b, j, l)` to compute the value at the node $R_{j,l}$ in the Romberg Lattice to approximate $\int_a^b f(x)dx$.

Support materials: Lectures 14; HW07.

Problem 7 (1 point)

Implement the function `richardson_2(av_2n, av_n)` that computes the 2nd Richardson Slide 10 in Lecture 14.

Support materials: Lectures 13, 14; HW07.

Problem 8 (1 point)

Implement the function `e_cont_frac_rat(i)` that returns the ith ratio to approximate the number e computed according to the following continued fraction.

$$\mathrm{e} = 2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{4 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{6 + \cdots}}}}}}}}$$

Implement the function `e_cont_frac_real(i, prec=20)` that computes the ith ratio that approximates the number e computed with the above continued fraction and returns it as a real number represented as a Python decimal with a precsiion specified by the second parameter.

Support materials: Lectures 15, 16; HW08.

Problem 9 (1 point)

Implement the function `pi_cont_frac_rat(i)` that returns the ith ratio to approximate the number π computed according to the following continued fraction.

$$\pi = 3 + \cfrac{1^2}{6 + \cfrac{3^2}{6 + \cfrac{5^2}{6 + \cfrac{7^2}{6 + \cfrac{9^2}{6 + \cfrac{11^2}{6 + \cfrac{13^2}{6 + \cfrac{15^2}{6 + \cdots}}}}}}}}$$

Implement the function `pi_cont_frac_real(i, i, prec=20)` that computes the ith ratio that approximates the number π computed with the above continued fraction and returns it as a real number represented as a Python decimal with a precision specified by the second parameter.

Support materials: Lectures 15, 16; HW08.

Problem 10 (1 point)

Implement the function `chudnovsky_pi(n, prec=20)` to compute the π approximation on the interval $[1, n]$ with the Chudnovsky algorithm with a precision specified by the second argument. The pseudocode is given on Slides 24 and 25 in Lecture 17.

Support materials: Lectures 16, 17; HW08.

What to Submit

Submit your solutions in `cs3430_s24_midterm02.py`. **Remember to include into your submission zip all the Python files you import from.**