CS 3430: S22: Scientific Computing
Midterm 02

Vladimir Kulyukin
Department of Computer Science
Utah State University

March 15, 2022

Introduction

This exam covers the material in Lectures 11 – 16.

There are 8 problems on this exam worth a total of 10 points.

You may use your class notes or the lecture PDFs posted under Canvas/Announcements. **You may not use any other materials (digital or paper)**.

You'll type and save your solutions in `cs3430_s22_midterm02.py` and submit this file (and, if necessary, other Python files you import from) **in Canvas by 11:59pm March 16, 2022**.

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

You may use your solutions to the previous assignments. For example, I will have no problem with you doing imports from your previous homework solutions as follows

```
from cdd import cdd
from rxp import rxp
from rmb import rmb
```

and then using your implementations of the central divided difference (CDD) in `cdd.py`, Richardson's Exptrapolation in `rxp.py`, and Romberg Integration in `rmb.py` to approximate values of various derivatives and integrals.

Remember to include into your submission zip all the Python files you're importing from. In other words, if you are importing anything from your `cs_3430_s22_hw08.py` or from `poly_parser.py`, include these files in the zip. When we run unit tests on your submission, we'll put all your files into the same working directory with your `cs3430_s22_midterm02.py`.

You may not use any third party libraries in this exam. You may use **only your own solutions** to previous/current assignments; you can use unit tests in `cs3430_s22_midterm02_uts.py` to test your solutions.

Remember to write your name and A-number in `cs3430_s22_midterm02.py`.

If you can, do me a favor and write below your name and A-number in `cs3430_s22_midterm02.py` how much time you spent on this exam. I give you my word that I won't make it public anywhere. This is only for me to assess the easiness/difficulty of the exam.

Good luck, Happy Hacking, and Happy Thinking!

Problem 1 (1 point)

Write a function `lambdify(s)` that takes a string `s` representing a polynomial and returns a Python function of one variable that computes the values of that polynomial.

You can use `test_mid02_01()` in `cs3430_s22_midterm02_uts.py` to test your solution.

We'll make the following simplifying assumptions that we made in Assignment 6 about string representations of polynomials. Each polynomial element is represented with the caret sign. For example, $5x^2$ is represented as `'5x^2'`, $10.5y^{-3}$ as `'10.5y^-3'`, $5.14z^3$ as `'5.14z^3'`.

Constants are represented as products whose first multiplicand is the constant itself and whose second multiplicand is the polynomial's variable raized to the 0^{th} power. For example, 100 is represented as `'100x^0'`. Each polynomial has the same variable in each of its elements. In other words, strings like `'5z^2 + 7z + 10'` and `'5x^2 + 7x + 10'` are great, but `'5x^2 + 7z + 10'` is not, because it represents a polynomial in 2 variables.

A variable raised to the power of 1 is represented as a product of 1 (a constant object) and a power object of the variable (a variable object) raised to the power of 1 (a constant object). In writing, this can be reflected by writing x `'1x^1'`, y as `'1y^1'`, etc.

There are no double minuses. In other words, such polynomial strings as `'5x^3 - -3x^3'` are not allowed. It's OK, however, to have a coefficient's minus follow a plus: `'5x^2 + -3x^3'`. Finally, there are always spaces on both sides of `'+'` and `'-'` when they are between two elements of a polynomial. Recall that if `s` is a string with a polynomial description, we can use `s.split()` to split `s` on white space to obtain string representations of individual elements.

Problem 2 (1 point)

Write the function `diff(s)` that takes a string `s` representing a polynomial with the same conventions about string representations as in Problem 1 above, parses that string into a function representation, computes another representation for the derivative of the polynomial, lambdifies the derivative's representation into a Python function, and returns that function.

You can use `test_mid02_02()` in `cs3430_s22_midterm02_uts.py` to test your solution.

Problem 3 (1 point)

Write the function `nra_approx(s, x0, num_iters=5)` that takes a string `s` with a polynomial, the first approximation to a zero root `x0`, and the number of iterations, and runs the Newton-Raphson algorithm (NRA) for the specified number of iterations and returns the float zero root approximation found after the specified number of iterations.

You can use `test_mid02_03()` in `cs3430_s22_midterm02_uts.py` to test your solution.

Problem 4 (2 points)

Write the functions `cdd_drv1_ord2(f, x, h)` and `cdd_drv1_ord4(f, x, h)` that use the central divided difference (CDD) to compute the second order approximation of the value of the first derivative of the function `f` at `x`, given a value of step size `h`.

Write the functions `cdd_drv2_ord2(f, x, h)` and `cdd_drv2_ord4(f, x, h)` that that use the central divided difference (CDD) to compute the fourth order approximation of the value of the second derivative of the function `f` at `x`, given a value of step size `h`.

You can use `test_mid02_04()` in `cs3430_s22_midterm02_uts.py` to test your solution.

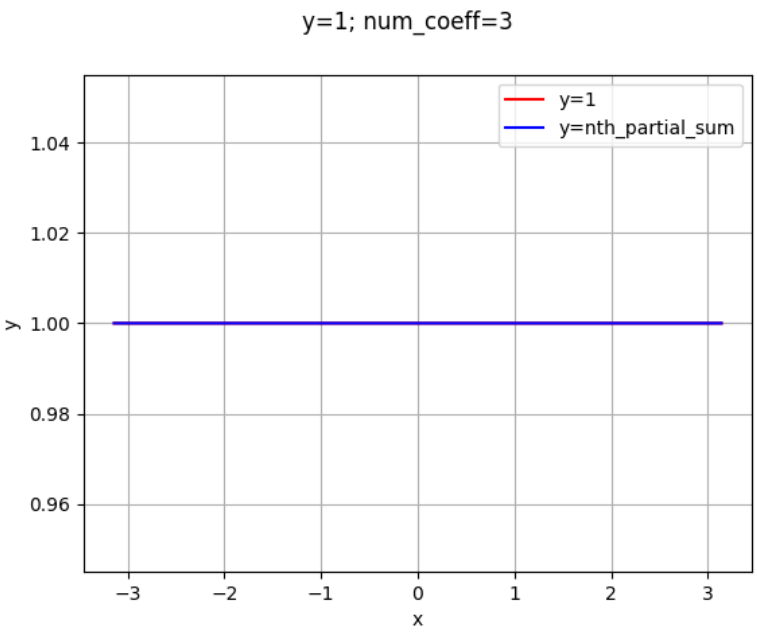

Figure 1: Plots of $y = 1$ and $s_3(x)$ on $[-\pi, \pi]$.

Problem 5 (1 point)

Type-draw the lattice for Romberg$_{4,4}$ (i.e., $R_{4,4}$) and state the error at each level. Use the notation $T_{i,j}$ to denote the calls to the trapezoidal rule.

Problem 6 (2 points)

Write the function `plot_fourier_nth_partial_sum(f, fstr, num_points=10000, num_coeffs=3, rn=15)` that takes a Python function `f`, the function string name (e.g., `'y=1'`) for the plot titles, the number of points linearly sampled from $-\pi$ upto π, the number of coefficients in s_n (i.e., the n-th partial sum of the Fourier series that approximates `f`), and the integer `rn` that specifies the top of the Romberg lattice for Romberg Integration (i.e., `rmb.rjl(f, -math.pi, math.pi, rn, rn)`). This function graphs two curves on the same plot and displays it: 1) the curve of `f` and 2) the curve of s_n that uses Romberg Integration in computing its Fourier coefficients.

You can use `test_mid02_06()` in `cs3430_s22_midterm02_uts.py` to test your solution.

Consider the following code segment.

```
f = lambda x: 1
fstr = 'y=1'
plot_fourier_nth_partial_sum(f, fstr, num_points=10000, num_coeffs=3, rn=15)
```

When I run the above code segment, my implementation produces the plot in Figure 1.

Write the function `plot_fourier_nth_partial_sum_error()` that takes the same parameters as `plot_fourier_nth_partial_sum()` and plots the true error curve between the values of $f(x)$ and $s_n(x)$ for

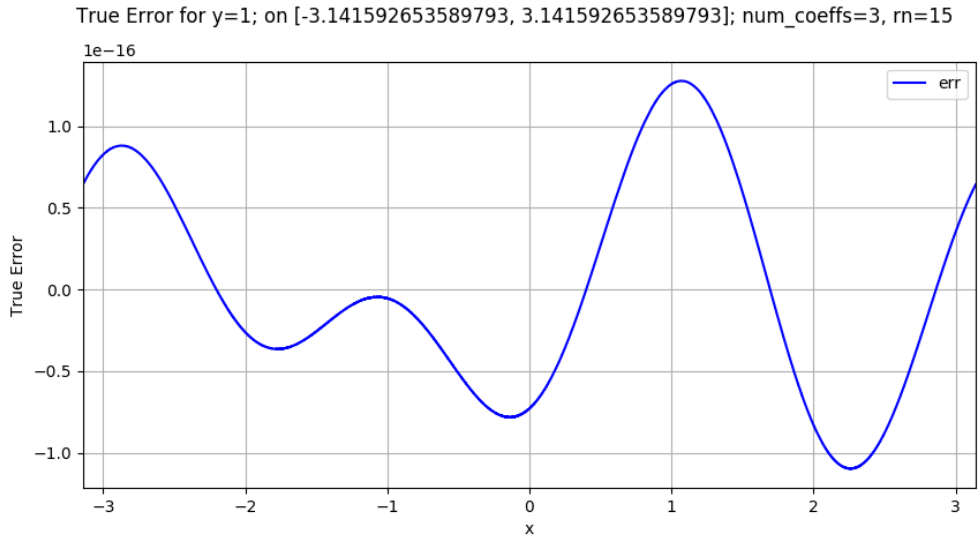

Figure 2: True Error between $y = 1$ and $s_3(x)$ on $[-\pi, \pi]$.

a given value of n.

Consider the following code segment.

```
f = lambda x: 1
fstr = 'y=1'
plot_fourier_nth_partial_sum_error(f, fstr, num_points=10000, num_coeffs=3, rn=15)
```

When I run the above code segment, my implementation produces the plot in Figure 2.

Consider the following code segment.

```
f = lambda x: math.sin(x) + math.cos(x) + 2*(math.sin(2*x) + math.cos(2*x))
fstr = 'y=sin(x)+cos(x)+2(sin(2x)+cos(2x))'
plot_fourier_nth_partial_sum(f, fstr, num_points=10000, num_coeffs=10, rn=15)
```

When I run the above code segment, my implementation produces the plot in Figure 3.

Consider the following code segment.

```
f = lambda x: math.sin(x) + math.cos(x) + 2*(math.sin(2*x) + math.cos(2*x))
fstr = 'y=sin(x)+cos(x)+2(sin(2x)+cos(2x))'
plot_fourier_nth_partial_sum_error(f, fstr, num_points=10000, num_coeffs=10, rn=15)
```

When I run the above code segment, my implementation produces the plot in Figure 4.

Problem 7 (1 point)

a) Give the definition of the basic trigonometric system and give the common period of the system's functions.

b) Let $f(x)$ and $g(x)$ be two functions integrable on $[a, b]$. When are $f(x)$ and $g(x)$ orthogonal?

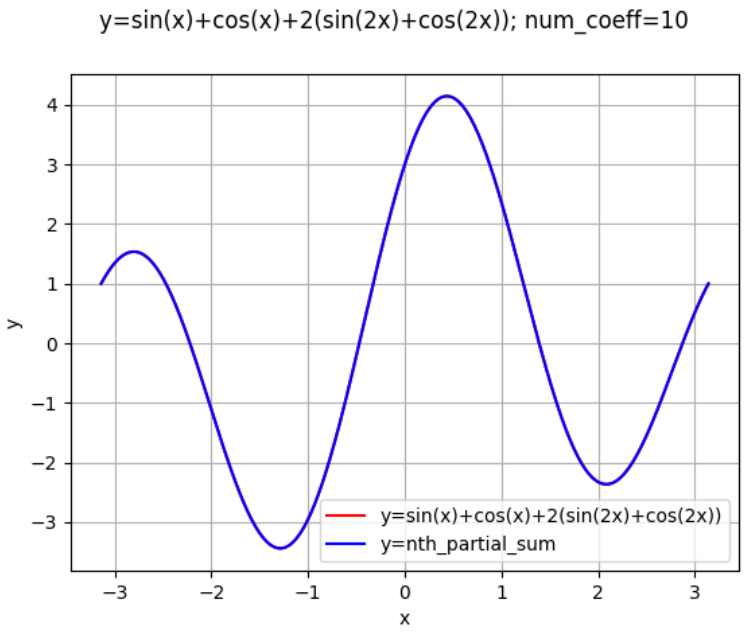

Figure 3: Plots of $y = 1$ and $s_3(x)$ on $[-\pi, \pi]$.

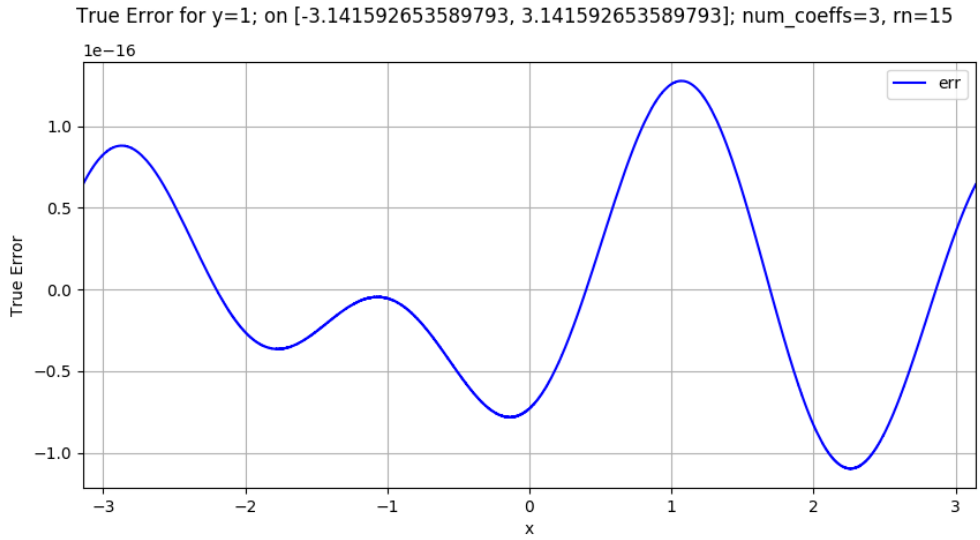

Figure 4: True Error between $y = 1$ and $s_3(x)$ on $[-\pi, \pi]$.

Problem 8 (1 point)

Give the definition of the basic trigonometric system and give the common period of the system's functions.

What to Submit?

Zip your `cs3430_s22_midterm02.py` (and all Python modules from which you import in `cs3430_s22_midterm02.py`) into `exam02.zip` and submit the zip archive in Canvas.
