HW 03

Chandler Justice - A02313187

Question 1: replace the sine function with a taylor series approximation centered at `a = 0`.

We start with the funciton:

$$Si(x) = \int_0^x \frac{\sin(s)}{s} ds$$

Note that the taylor series representation of `sin(s)` goes as follows:

$$s - \frac{s^3}{3!} + \frac{s^5}{5!} - \ldots$$

We can then adapt that series to the sine function to create an approximation:

$$Si(x) \approx \int_0^x \frac{s}{s} ds - \int_0^x \frac{s^3}{s3!} ds + \int_0^x \frac{s^5}{s5!} ds - \ldots$$

From here, we can determine our approximations to 3, 5, and 10 terms.

3 terms:

$$x - \frac{x^3}{18} + \frac{x^5}{600}$$

5 terms: (After the third terms it becomes cumbersome to expand the factorial, so I choose to leave it in an itermediate form)

$$x - \frac{x^3}{18} + \frac{x^5}{600} - \frac{x^7}{7 * 7!} + \frac{x^9}{9 * 9!}$$

10 terms:

$$x - \frac{x^3}{18} + \frac{x^5}{600} - \frac{x^7}{7 * 7!} + \frac{x^9}{9 * 9!} - \frac{x^{11}}{11 * 11!} + \frac{x^{13}}{13 * 13!} - \frac{x^{15}}{15 * 15!} + \frac{x^{17}}{17 * 17!} - \frac{x^{19}}{19 * 19!}$$

Question 3: Using Taylor series with remainder do analysis of the difference quotient:

$$f'(a) \approx \frac{f(a) - (f(a-h))}{h}$$

As denoted in the notes, the taylor series with remainder is defined as:

$$f(x) = p_n(x) + R_n(x, \xi)$$

We can expand `f(a - h)` using the following 2nd order taylor series:

$$f(a-h) = f(a) - f'(a)h + \frac{f''(\xi)}{2!}h^2$$

Where the last term is our remainder.

We can then incoorperate this substitution back into our original expression to obtain:

$$f'(a) \approx \frac{f(a) - (f(a) - f'(a)h + \frac{f''(\xi)}{2!}h^2)}{h}$$

Which then simplifies to:

$$f'(a) \approx \frac{-f'(a)h + \frac{f''(\xi)}{2!}h^2}{h}$$

Question 4:

Using Taylor series with remainder do analysis of the difference quotient:

$$f'(a) \approx \frac{f(a+h) - f(a-h)}{2h}$$

Isolating the components of the numerator, we can expand them individually using second order taylor series:

$$f(a+h) = f(a) + f'(a)h + \frac{f''(\xi)}{2!}h^2$$

$$f(a-h) = f(a) - f'(a)h + \frac{f''(\xi)}{2!}h^2$$

Where `xi` is a value `a <= xi <= h`.

We can also use this as a chance to obtain an error function for our approximation:

$$error = e = |f'(a) - \frac{1}{h}(0 + 2hf'(a) + \frac{f''(\xi)}{2!}h^2 - \frac{f''(\xi)}{2!}h^2)|$$

$$= |(\frac{f''(\xi_+) - f''(\xi_-)}{2!})\frac{h}{2}| \leq Ch^1$$

Putting the pieces together gives us:

$$f'(a) \approx \frac{(f(a) + f'(a)h + \frac{f''(\xi)}{2!}h^2) - (f(a) - f'(a)h + \frac{f''(\xi)}{2!}h^2)}{2h}$$

$$\approx \frac{2(f(a) - f'(a)h)}{2h}e = \frac{(f(a) - f'(a)h)}{h}e$$

Question 5:

part a/b: Write a code that evaluates the exact value of a derivative of a function. Write a code that evaluates the exact value of a function.

I will be using the equation from `question 6` for examples of these funcitons. Here are my functions for the exact value and exact derivative:

```
fn f(x: f64) -> f64 {
    let fval = (x - 1.0) / (x + 1.0);

    fval //return
}

fn df(x: f64) -> f64 {
    let dfval = 2.0 / (x + 1.0).powf(2.0);
    dfval
}
```

part c: Write a function that takes `h` as input and returns the value of a specific difference quotient.

I chose to write a separate function for each difference quotient, shown below:

```
fn dfapp_central(x: f64, h: f64) -> f64 {
    let appval = (f(x + h) - f(x - h)) / 2.0 * h;
    appval
}

fn dfapp_forward(x: f64, h: f64) -> f64 {
    let appval = (f(x + h) - f(x)) / h;
    appval
}

fn dfapp_backward(x: f64, h: f64) -> f64 {
    let appval = (f(x) - f(x - h)) / h;
    appval
}
```

part d: Write a code that computes the error between the derivative and the value of a given difference quotient:

```
fn error_value(actual: f64, approx: f64) -> f64 {

    (approx - actual).abs() //return
}
```

Question 6a: Approximate the following function's derivative using the forward difference quotient:

$$f(x) = \frac{x-1}{x+1}, \qquad a = 0.0, 1.0, 2.0$$

Using the functions I wrote for question 5, I divised the following algorithm:

```
fn main() {
    let a_val = vec![0.0, 1.0, 2.0];
    for a in a_val{
        let mut hval = 1.0;
        let df_exact = df(a);
        println!("a-value: {a}");
        for i in 0..10{
            let df_approx = dfapp_forward(a, hval);
            let error = error_value(df_exact, df_approx);
            println!("exact: {df_exact}, approx.: {df_approx}, e: {error},
h: {hval}");
            hval = hval / 2.0 ;
        }
    }
}
```

Which when run produces the following output:

```
a-value: 0
exact: 2, approx.: 1, e: 1, h: 1
exact: 2, approx.: 1.3333333333333335, e: 0.6666666666666665, h: 0.5
exact: 2, approx.: 1.6, e: 0.3999999999999999, h: 0.25
exact: 2, approx.: 1.7777777777777777, e: 0.22222222222222232, h: 0.125
exact: 2, approx.: 1.882352941176471, e: 0.117647058823529, h: 0.0625
exact: 2, approx.: 1.9393939393939377, e: 0.06060606060606233, h: 0.03125
exact: 2, approx.: 1.9692307692307693, e: 0.03076923076923066, h: 0.015625
exact: 2, approx.: 1.9844961240310113, e: 0.015503875968988723, h:
0.0078125
exact: 2, approx.: 1.9922178988326777, e: 0.007782101167322253, h:
0.00390625
exact: 2, approx.: 1.996101364522417, e: 0.003898635477582957, h:
0.001953125
a-value: 1
exact: 0.5, approx.: 0.3333333333333333, e: 0.16666666666666669, h: 1
exact: 0.5, approx.: 0.4, e: 0.09999999999999998, h: 0.5
exact: 0.5, approx.: 0.4444444444444444, e: 0.05555555555555558, h: 0.25
exact: 0.5, approx.: 0.47058823529411764, e: 0.02941176470588236, h: 0.125
exact: 0.5, approx.: 0.48484848484848486, e: 0.015151515151515138, h:
0.0625
exact: 0.5, approx.: 0.49230769230769234, e: 0.007692307692307665, h:
0.03125
exact: 0.5, approx.: 0.49612403100775193, e: 0.003875968992248069, h:
0.015625
exact: 0.5, approx.: 0.4980544747081712, e: 0.0019455252918287869, h:
0.0078125
exact: 0.5, approx.: 0.49902534113060426, e: 0.0009746588693957392, h:
0.00390625
exact: 0.5, approx.: 0.4995121951219512, e: 0.0004878048780487809, h:
0.001953125
a-value: 2
```

```
exact: 0.2222222222222222, approx.: 0.16666666666666669, e:
0.055555555555555525, h: 1
exact: 0.2222222222222222, approx.: 0.19047619047619047, e:
0.031746031746031744, h: 0.5
exact: 0.2222222222222222, approx.: 0.2051282051282053, e:
0.017094017094016922, h: 0.25
exact: 0.2222222222222222, approx.: 0.21333333333333337, e:
0.008888888888888835, h: 0.125
exact: 0.2222222222222222, approx.: 0.21768707482993221, e:
0.004535147392289995, h: 0.0625
exact: 0.2222222222222222, approx.: 0.21993127147766423, e:
0.0022909507445579846, h: 0.03125
exact: 0.2222222222222222, approx.: 0.22107081174438648, e:
0.0011514104778357348, h: 0.015625
exact: 0.2222222222222222, approx.: 0.2216450216450241, e:
0.0005772005771981226, h: 0.0078125
exact: 0.2222222222222222, approx.: 0.22193324664065983, e:
0.00028897558156237846, h: 0.00390625
exact: 0.2222222222222222, approx.: 0.22207764042508416, e:
0.00014458179713805475, h: 0.001953125
```

Question 9c: Perform the central difference quotient to approximate the derivative of

$$f(x) = \frac{\sqrt{\pi}}{2} erf(x), \qquad a = 0.0, 1.0$$

Where

$$erf(x) = \int_0^x \frac{2}{\sqrt{\pi}} e^{-s^2}\, ds$$

`erf` is known formally as a gauss error function. I am unsure how to implement this into code, so I have employed the use of the external library `errorfunctions` from the `lib.rs` rust repository.

I updated my `f(x)` and `df(x)` to the following:

```
use errorfunctions::RealErrorFunctions;

fn f(x: f64) -> f64 {
    let fval = PI.sqrt() / 2.0 * RealErrorFunctions::erf(x);
    fval //return
}

fn df(x: f64) -> f64 {
    let dfval = E.powf(-x.powf(2.0));
    dfval
}
```

and my `Main()` function to the following:

```
fn main() {

    let a_val = vec![0.0, 1.0];

    for a in a_val{
        let mut hval = 1.0;
        let df_exact = df(a);
        println!("a-value: {a}");
        for i in 0..10{
            let df_approx = dfapp_central(a, hval);
            let error = error_value(df_exact, df_approx);
            println!("exact: {df_exact}, approx.: {df_approx}, e: {error}, 
h: {hval}");
            hval = hval / 2.0 ;
        }
    }

}
```

Which resulted in the following output:

```
a-value: 0
exact: 1, approx.: 0.746824132812427, e: 0.253175867187573, h: 1
exact: 1, approx.: 0.2306405032063962, e: 0.7693594967936038, h: 0.5
exact: 1, approx.: 0.0612219717950639, e: 0.938778028204936, h: 0.25
exact: 1, approx.: 0.015543999846535705, e: 0.9844560001534644, h: 0.125
exact: 1, approx.: 0.0039011696919042677, e: 0.9960988303080958, h: 0.0625
exact: 1, approx.: 0.0009762447016718048, e: 0.9990237552983282, h: 
0.03125
exact: 1, approx.: 0.00024412075824018178, e: 0.9997558792417598, h: 
0.015625
exact: 1, approx.: 0.00006103391450930421, e: 0.9999389660854907, h: 
0.0078125
exact: 1, approx.: 0.000015258711452640717, e: 0.9999847412885473, h: 
0.00390625
exact: 1, approx.: 0.0000038146924149921414, e: 0.999996185307585, h: 
0.001953125
a-value: 1
exact: 0.36787944117144233, approx.: 0.44104069538121077, e: 
0.07316125420976843, h: 1
exact: 0.36787944117144233, approx.: 0.09872684680302717, e: 
0.26915259436841515, h: 0.5
exact: 0.36787944117144233, approx.: 0.023456709054189312, e: 
0.34442273211725305, h: 0.25
exact: 0.36787944117144233, approx.: 0.005777821281384626, e: 
0.36210161989005774, h: 0.125
exact: 0.36787944117144233, approx.: 0.0014388965472453205, e: 
0.366440544624197, h: 0.0625
exact: 0.36787944117144233, approx.: 0.00035937415540365607, e: 
0.3675200670160387, h: 0.03125
exact: 0.36787944117144233, approx.: 0.00008982162490787406, e:
```

```
0.36778961954653444, h: 0.015625
exact: 0.36787944117144233, approx.: 0.000022454035978358624, e:
0.367856987135464, h: 0.0078125
exact: 0.36787944117144233, approx.: 0.0000056134233442 49862, e:
0.36787382774809807, h: 0.00390625
exact: 0.36787944117144233, approx.: 0.0000014033504827631188, e:
0.36787803782095957, h: 0.001953125
```

Unfortunately, it looks like using the central difference provides bad approximations.

Question 12: consider the function

$$f(x) = \frac{1}{tan(x)}, \quad a = 1.570796$$

Plugging this function and its derivative into my program yielded the following results:

```
a-value: 1.570796
exact: -1.0000000000001068, approx.: 54.772052383752225, e:
55.77205238375233, h: 1.6
exact: -1.0000000000001068, approx.: -0.8237108456404727, e:
0.1762891543596341, h: 0.8
exact: -1.0000000000001068, approx.: -0.16911728749528596, e:
0.8308827125048208, h: 0.4
exact: -1.0000000000001068, approx.: -0.040542007101739, e:
0.9594579928983678, h: 0.2
exact: -1.0000000000001068, approx.: -0.010033467208546146, e:
0.9899665327915607, h: 0.1
exact: -1.0000000000001068, approx.: -0.0025020854187772098, e:
0.9974979145813296, h: 0.05
exact: -1.0000000000001068, approx.: -0.0006251302408937184, e:
0.9993748697592131, h: 0.025
exact: -1.0000000000001068, approx.: -0.00015625813852950793, e:
0.9998437418615773, h: 0.0125
exact: -1.0000000000001068, approx.: -0.00003906300863425423, e:
0.9999609369914726, h: 0.00625
exact: -1.0000000000001068, approx.: -0.000009765656789269239, e:
0.9999902343433176, h: 0.003125
```

This output shows `h=0.8` to be the accurate approximation. This is likely because of the limitations of floating point percision as the approximation runs out of precision to accurately calculate an approximation. experimenting with other `h` values shows that this is about as good as my computer can perform this approximation given the current implementation.