Math475A – Homework2

Tyler J Gabb

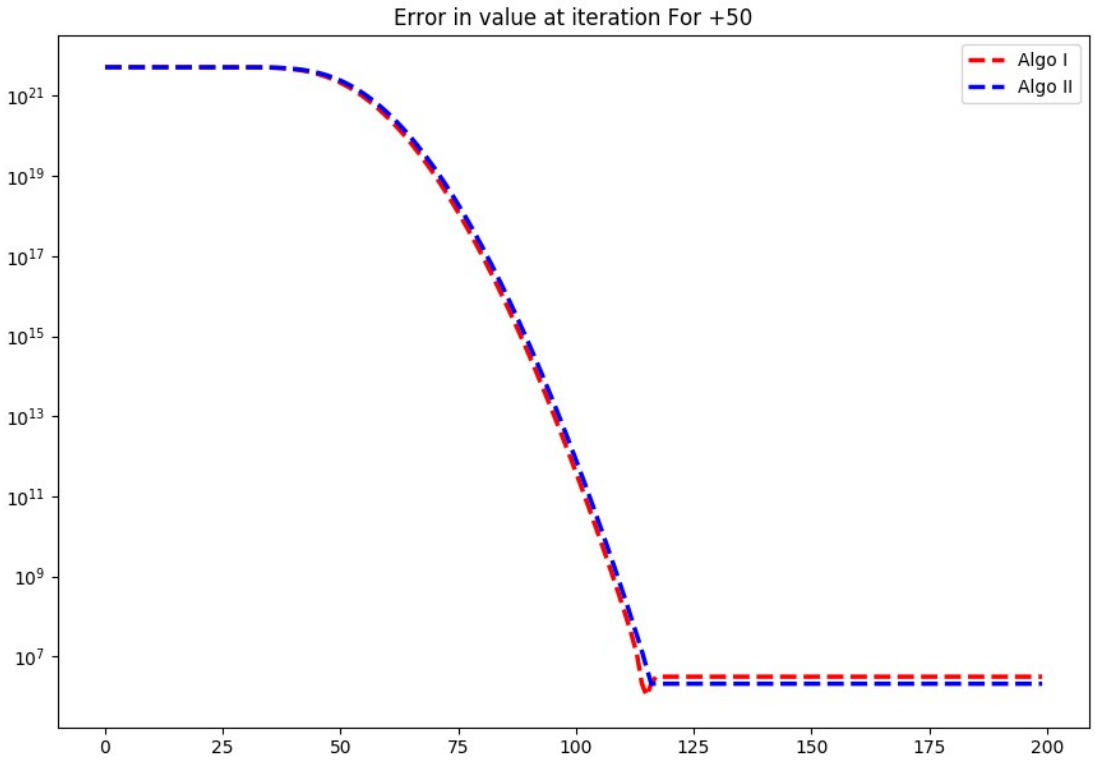
TASK1: We were tasked with applying two given algorithms for approximating exp(x) at x=50. These algorithms are given below in python3.

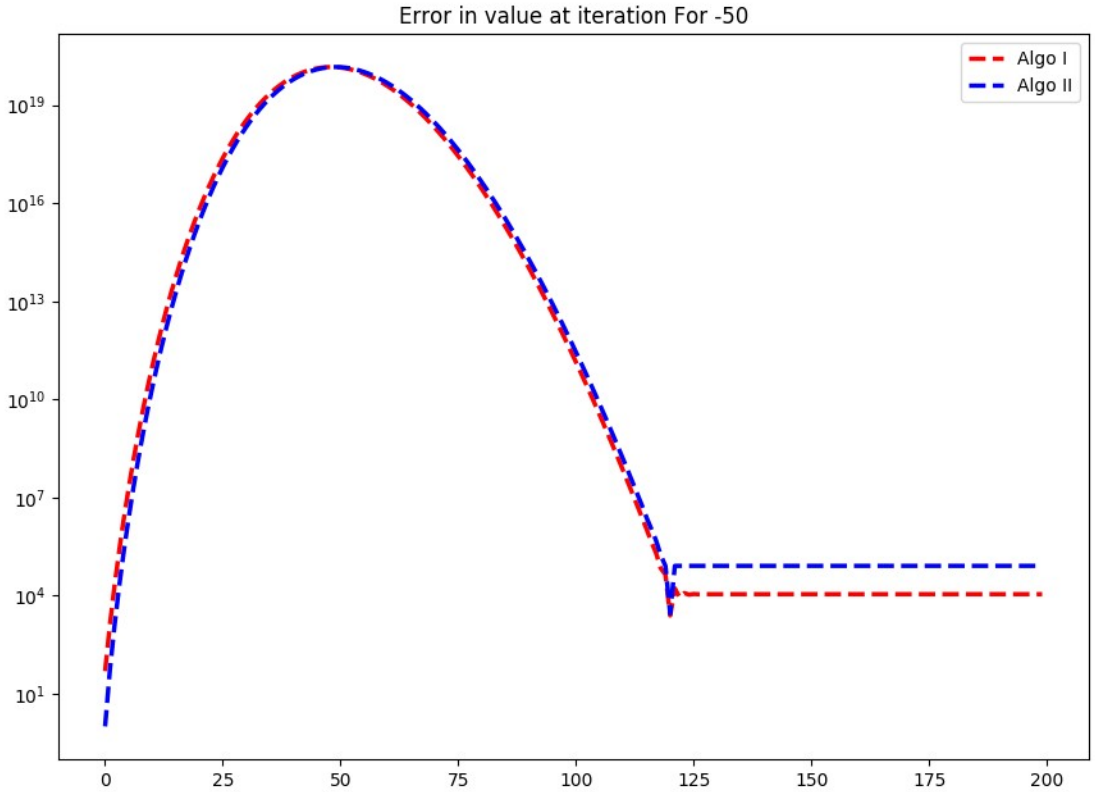
```
def algo1(x):
    """
    Returns an array of 200 error terms, each of a sucessive approximation
    for e raised to the power of x. uses "Algorithm I"
    """
    terms = []
    s = 1
    t = x
    N_values = list(range(1,ITERS + 1))
    for n in N_values:
        s = s + t
        t = t * x / (n + 1)
        terms.append(s)

    return [abs(M.exp(x) - y) for y in terms]
```

```
def algo2(x):
    """
    Returns an array of error terms, each of a sucessive approximation
    for e raised to the power of x. uses "Algorithm II"
    """
    terms = []
    for i in range(ITERS):
        s = 1
        for n in range(i,0,-1):
            s = s * x / n + 1
        terms.append(s)

    return [abs(M.exp(x) - y) for y in terms]
```

**Where ITERS is a global integer passed in as a command line argument

Below are the plots for x values of 50, and -50:


Error in value at iteration For +50
Algo I
Algo II
10^{21}
10^{19}
10^{17}
10^{15}
10^{13}
10^{11}
10^{9}
10^{7}
0
25
50
75
100
125
150
175
200


Error in value at iteration For -50
Algo I
Algo II
10^{19}
10^{16}
10^{13}
10^{10}
10^{7}
10^{4}
10^{1}
0
25
50
75
100
125
150
175
200

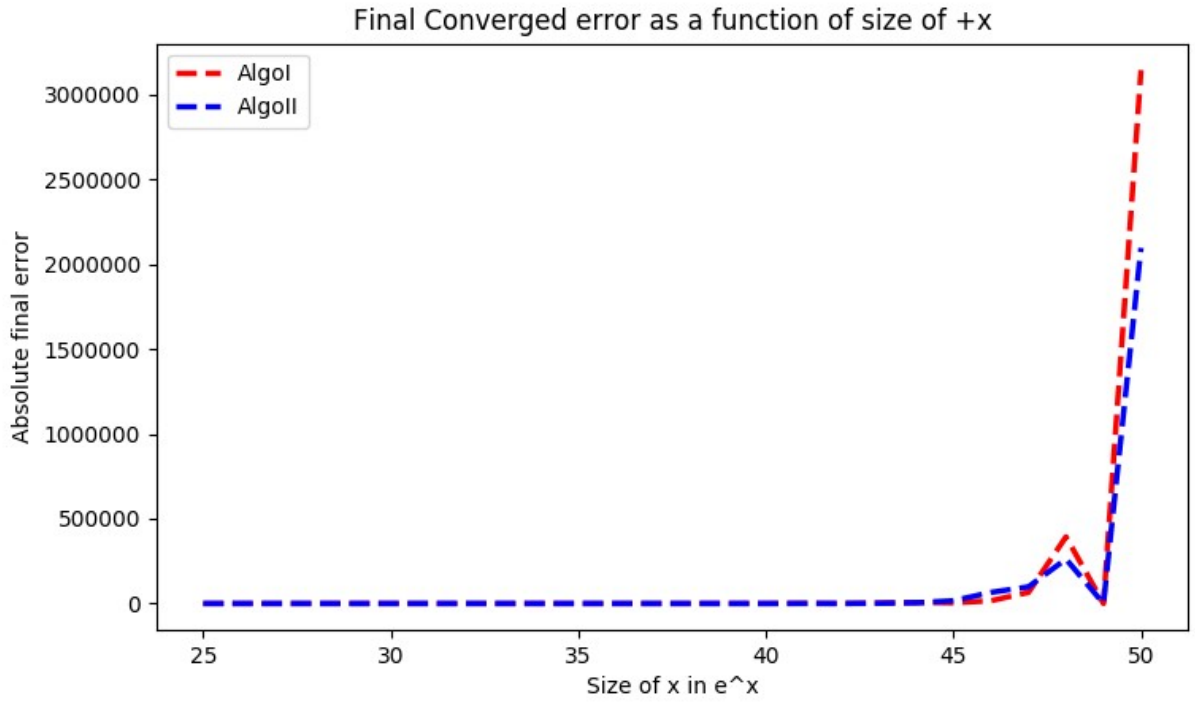
A few things I noticed:

1) for Positive 50, the error for AlgoI converged to a lower value than that of AlgoII, where for negative 50, the outcome was reversed, and there is more of a difference between the final values for each algorithm.

2) The rate of convergence seems to be equal for both algorithms in both scenarios, where one converges to a worse value than the other

3) The error still seems to be substantial, and I think this is due to the size of x, 50 in this case. As the numbers get larger we introduce more truncation error. Also, as the numbers get smaller, we introduce roundoff error.

Note* When this experiment was done again for values of x near 1, like ±1,2,3… The error which each algorithm converged to was much smaller than near ±50. For curiosity sake, below is a plot of the final (converged) error vs size of x after 200 iterations



I'd say thats peculiar

TASK2: We were tasked with finding the minimum and maximum integers and floating-point numbers in our computational environment, and asked to deduce our exponent and mantissa architecture

Below is code used to find the smallest value near a particular number.

```
def find_smallest_closest_to(val):
    """
    Finds the smallest value to the right of val on the real-number line
    """
    x = 1.0
    divisions = 0
    while (val + x > val):
        x_last = x
        x = x / 2
        divisions += 1
        if VERBOSE:
            print(str.format("Division={0}  Result={1}", divisions, x))
    return (str(val) + "+" + str( x_last),divisions-1)
```

Below is code used to find the largest representable integer in the environment.

```
def find_largest():
    """
    Finds largest representable number in environment
    """
    x = 1.0;
    stage = 0
    while (x != math.inf and x != math.nan):
        x_last = x
        x = x * 2
        stage += 1
        if VERBOSE:
            print(str.format("Stage={0} Result={1}",stage,x))
    return (str(x_last),stage-1)
```

These functions were used to produce m,n, and N. These three values were then used to calculate the largest representable floating point number.

Observe the code in main below, and its output below that

```
def main():
    (s0,n) = find_smallest_closest_to(0);
    (s1,m) = find_smallest_closest_to(1);
    (S2,N) = find_largest();
    F = 2**N*(2-0.5**m)
    print("The smallest value closest to 0 is {0} and n = {1}".format(s0,n))
    print("The smallest value closest to 1 is {0} and m = {1}".format(s1,m))
    print("The largest integer is {0} and N = {1}".format(S2,N))
    print("The largest float   is {0}".format(F))
    return F
```

```
>>> main()
The smallest value closest to 0 is 0+5e-324 and n = 1074
The smallest value closest to 1 is 1+2.220446049250313e-16 and m = 52
The largest integer is 8.98846567431158e+307 and N = 1023
The largest float   is 1.7976931348623157e+308
```

From this information, I was able to deduce that the exponent in the floating point equation was 11. I knew this because the value of N in binary is 11 bits long, and this is the power of the exponent for the largest representable number.
Since I know that my computers architecture is 64 bit, I can subtract 11 from 64, and end up with 53. Subtracting one more for the sign bit allows be to deduce that the significand has 52 available bits.

0/1 (sign)	11 Bit Exp	52Bit Significant