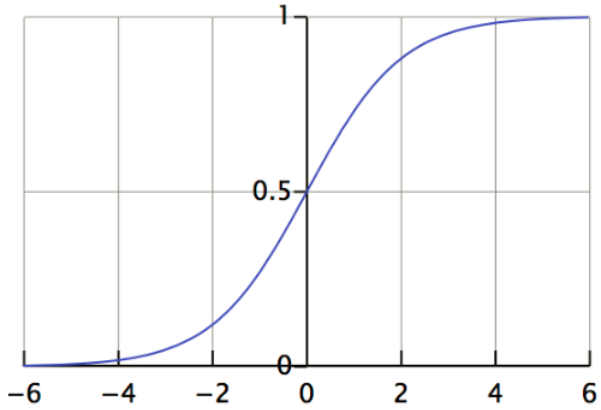
Jeffrey Rodriguez 110733867
AMS 326
Homework 1
2/16/2018

1.1

The sigmoid function is defined as $S(x) = 1/(1 + e^{-x})$ and can be seen as depicted in the following graph:



Taken from homework assignment PDF

Using numerical integration algorithms, we can approximate the area underneath this curve bounded between $x \in [-4, 5]$. The first algorithm used is the rectangle rule, in which we partition the region into a collection of rectangles, with width based on the amount of rectangles created, and height based on the value of the function at these two endpoints. The smaller the width of the rectangle, the more accurate our approximation is. The algorithm is applied as:

1. Set a, b equal to the lower and upper bounds respectively.
2. Let n denote the number of rectangles in our partition
3. Define the width of each rectangle as $h = \frac{b-a}{n}$
4. Let $x_i = a + i \cdot h$, for all integers i, such that $0 \leq i < n$.
5. Compute $I(x) = h * \sum_{i=0}^{n-1} S(x_i)$.

The next algorithm used is a modified version of the trapezoid rule, shown by the following algorithm.

1. Set a, b equal to the lower and upper bounds respectively.

2. Partition the function's region into n rectangles.

3. Define the width of each rectangle as $h = \frac{b-a}{n}$

4. Compute the midpoint $m = \frac{S(a)+S(b)}{2}$

5. Let $x = a + i \cdot h$, for all integers i, such that $1 \leq i < n$.

6. Compute $I(x) = h \cdot (m + \sum_{i=0}^{n-1} S(x_i))$.

These methods were used in the attached Python program 'HW1.py' and made use of the numpy package to shorten the amount of lines needed to perform operations on an array of x_i values. For the rectangle rule, with $n = 1,000,000$, we have $I(x) = \underline{4.988}5610316270075$. For the trapezoid rule, also with $n = 1,000,000$. we see that $I(x) = \underline{4.988}565420571235$.
To estimate the amount of flops performed by these functions, we first look at the function $S(x)$. The makes use of addition (1 operation), and exponentiation followed by division (roughly 8 operations). So we can guess that calling this function n times will require $9n$ flops.
Computing the width h requires about 5 flops (1 for addition, 4 for multiplying by n^{-1}). The array of x values requires $2n$ operations (addition and multiplication both performed n times, with one less operation for addition).
For the rectangle rule, summing over each $S(x_i)$ will cost a total of $(n-1) \cdot 9n = 9n^2 - 9n$ flops (0 to $n-1$ terms to sum), and multiplying this result by h will cost 1 additional flop. So in total, we can expect this to cost $5 + 2n + (n-1) \cdot 9n + 1 =$

$$9n^2 - 7n + 6 \text{ flops}$$

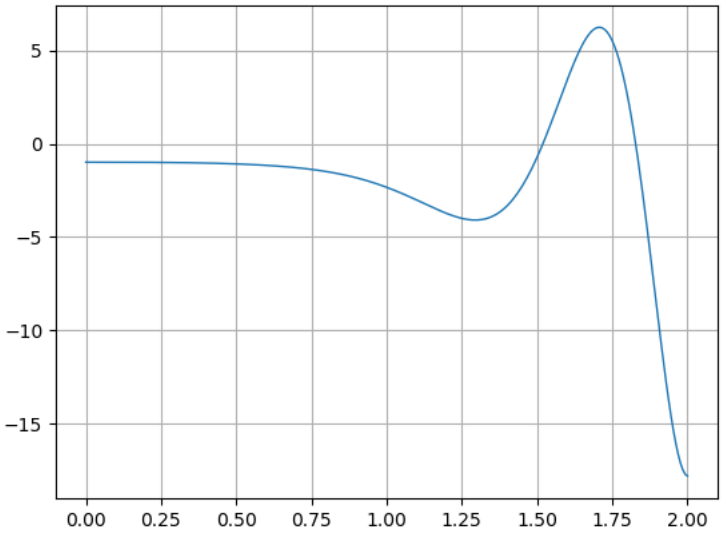
.
For the trapezoid rule, the average of $S(a)$, $S(b)$ costs $1 + 2{\cdot}9{=}19$ operations. The return with the summation will cost $1{+}1{+}(n{-}1){\cdot}8n$. The total cost is $5{+}2n{+}19{+}1{+}(1{+}9(n{-}1)(n{-}2)) =$

$$9n^2 - 25n + 43 \text{ flops}$$

.
So, we see that the trapezoid rule will take a bit less time to run, although both run with $O(n^2)$ time complexity. Both run fast, but increasing n will of course increase the time required to run.

1.2

Next, we wish to estimate one of the roots of the following functions. This is graphed with Python's matplotlib.pyplot package.



$$f(x) = 2.018^{-x^3} - x^4 \sin(x^3) - 1.984$$

Based on the graph, we estimate one of these roots is in a δ-neighborhood around the point $x_1 = 1.51$. For $\delta = 0.01$, we see that the root will lie in the interval $[x_1 - \delta, x_1 + \delta] = [1.50, 1.52]$. To find an approximation to this root, we use the bisection method defined by the following algorithm.

1. Set $a = 1.50$, $b = 1.52$, such that $f(a)f(b) \leq 0$.
2. Let $m_i = \frac{a_i + b_i}{2}$
3. Let $\epsilon > 0$
4. Set a counter, count = 0 to count the amount of iterations.
5. Define a limit of iterations allowed L.
6. While $|f(m_i)| > \epsilon$ and count $< L$:
   - If $f(a)$ and $f(m_i)$ share the same sign, set $a_{i+1} = m_i$. If instead, $f(b)$ and $f(m_i)$ share the same sign, set $b_{i+1} = m_i$.
     Every time $f(m_i)$ is evaluated, increment the counter by 1.

This algorithm was evaluated on Python, with a limit L set to 50 and $\epsilon = 10^{-7}$, but this is easily modifiable. We find that $x_n = \underline{1.519}1737079620358$ after 20 iterations. This program runs fast, but further decreasing the value of ϵ will increase run-time. The attached file '326HW1.py' contains commented lines at the very end giving the functions to call the graph and bisection methods.