Michael VanDenburgh
SBU ID:

February 14, 2020
AMS 326: Numerical Analysis

Homework 1 Report

Problem 1.1

1 Problem Description

The heart equation is $x^2 + (y - \sqrt{|x|})^2 = 2$. A disc of maximum area is dug out of the heart. Use a numerical integration algorithm to compute the area of the remaining heart and estimate the number of floating-point operations needed for each method used.

2 Algorithm Description and Pseudo-Code

2.1 Calculating the area of the disc

First, algebraically rearrange the heart equation:

$$y^2 + (-2\sqrt{|x|})y + (|x| + x^2 - 2) = 0.$$

The center and radius of the disc can be computed numerically as follows.
Using the rearranged heart equation, I can calculate the two y values of the heart at $x = 0$ as $-\sqrt{2}$ and $\sqrt{2}$. The lower value, which is the y value at which the two halves of the heart meet at the bottom, is not useful since the disc clearly doesn't touch it, as observed from viewing the picture. However, the higher value, which is the y value at which two halves of the heart meet at the top, is the highest point (maximum y value) of the disc; this can be reasoned by observing that (1) $x = 0$ has the lowest y value of all the points on the "top" of the heart, and (2) the disc is the maximum possible size while staying inside the heart. Since the heart is symmetric, the disc must also be symmetric, therefore for the disc to be max area within the heart, its center must be at $x = 0$. To get the y value of the center, the following algorithm can be used.
Start just below the top of the disc $y = \sqrt{2}$ (something like $y = \sqrt{2} - 0.0001$, depending on how many significant digits are needed). Consider this value as the center of the current disc, with the radius being the distance from the center to the top of the disc, $\sqrt{2} - center$. Check if the disc intersects the heart at any point (only one side is necessary here since the graph is symmetric, I chose the left (negative) half). If it does, the current center and radius

are approximately equal to the center and radius of the disc. Otherwise, repeat the process again with a slightly smaller y value for center. Visually, I am moving the center of the disc with a maximum radius down the y axis until it it intersects with the heart.
Using this algorithm, I approximated the center of the disc to be at coordinates (0, 0.3814510000000019) with a radius of 1.0327625623730932. The area of the disc can now be calculated:

$$\boxed{A = \pi r^2 \approx \pi(1.0327625623730932)^2 \approx 3.350818044098033}$$

Pseudo-code:

```
y = $\sqrt{2}$ // the top of the circle $y$ value
center = y
while (center is inside heart) {
    current_radius = y - center
    while (current_radius > 0) {
        if heart intersects the disc at any point then break loops
        else increment current_radius
    }
    decrement center
}
area = $\pi$ * current_radius ^ 2
```

2.2 Calculating the area of the heart

Analytically, the equation of the heart can be split down the y axis into two halves yielding two equations for each half of the heart:

$$y^2 + (-2\sqrt{-x})y + (|x| + x^2 - 2) = 0 \text{ (left side)}$$
$$y^2 + (-2\sqrt{x})y + (|x| + x^2 - 2) = 0 \text{ (right side)}$$

Since the heart is symmetric it's only necessary to find the area of one of the halves and double it; the left side is used here. The equation can be decomposed into two functions as follows:

$$y^2 + (-2\sqrt{-x})y + (|x| + x^2 - 2) = 0$$

$$\Downarrow$$

$$y = \sqrt{-x} \pm \sqrt{-x^2 - x - |x| + 2}$$

From these equations, it's clear from the presence of $-x$ under the square root that x must always be zero or negative, which makes sense since the equation represents the left half of the heart. It can also be algebraically determined that x is undefined for any value less than $-\sqrt{2}$, which means $-\sqrt{2}$ represents the point furthest to the left on the heart. Using these two functions, we can represent the area inside the heart:

$$A = 2\Bigg[\int_{-\sqrt{2}}^{0}(\sqrt{-x} + \sqrt{-x^2 - x - |x| + 2})dx - \int_{-\sqrt{2}}^{-1}(\sqrt{-x} - \sqrt{-x^2 - x - |x| + 2})dx$$
$$+ \int_{0}^{-1}(\sqrt{-x} - \sqrt{-x^2 - x - |x| + 2})dx\Bigg]$$

To solve for the area I use Simpson's Rule for numerically approximating integrals: $\int_a^b f(x)\,dx \approx \frac{\Delta x}{3}\left(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots + 4f(x_{n-1}) + f(x_n)\right)$

In Simpson's rule, we divide the area under the curve into several parabolas and sum the areas of each one. The usage of parabolas instead of rectangles or trapezoids results in a more accurate approximation than the rectangle or trapezoid rules.

Pseudo-code for Simpson's Rule for integration:

```
integrate(f, a, b, n) {
    $\Delta$x = $\frac{b-a}{n}$
    sum = 0
    for i = 0 through n, do {
        x = a + i * $\Delta$x
        if i == 0 then sum = sum + f(x)
        else if i == n then sum += f(a * n * $\Delta$x)
        else if i is odd then sum = sum + 4 * f(x)
        else sum = sum + 2 * f(x)
    }
    return ($\Delta$x / 3) * sum
}
```

Using this algorithm with $n = 10000000$ results in an approximate area of 6.283185227879018.

3 Results

Now that the area of the entire heart (6.283185227879018) and disc (3.350818044098033) are known, the area of the remaining heart can be calculated by subtracting them:

$$\boxed{\text{Area of remaining heart} \approx 6.283185227879018 - 3.350818044098033 = 2.932367184}$$

4 Brief comments on performance & other questions

4.1 Area of the disc

The program runs in quadratic $O(n^2)$ time. This is because the total number of times both the outer while loop and inner while loop goes around depends on the value that the disc's center and radius is decremented/incremented by each iteration, respectively. In order to increase the accuracy, it is necessary to make this "incrementBy" value smaller. Due to the $O(n^2)$ time complexity, the program slows down significantly the smaller "incrementBy" gets if optimizations are not made. For example, setting incrementBy to be 0.001 results in a runtime of less than a second while setting it to 0.0001 results in a runtime of about 64 seconds. I was able to fix this by having the algorithm run several iterations of maximizing the disc, each time with a smaller "incrementBy" value (represented in the outer for loop in the code). To prevent redundantly checking values that have already been checked in previous iterations, the program uses the previous approximations of the radius and center to start at. For an incrementBy value of 0.0001, the program first approximates the radius and center using an incrementBy value of 0.001, and then uses those estimates as starting points in the iteration with incrementBy = 0.0001. This causes the time required for incrementBy = 0.0001 to go from about 64 seconds to about 2 seconds. This allows for more accurate approximations - I was able to get the required accuracy of four significant digits by running 4 iterations of this optimization.

Estimated number of floating-point operations for this method:

(A_{disc}/ incrementBy) * (3 + r_{disc} / incrementBy)

(Outer while loop goes around about A_{disc}/ incrementBy times, inside it has three floating point operations and the inner while loop, which also has three floating point operations and goes around about r_{disc}/ incrementBy times. Therefore, using a value for incrementBy of 0.00001, the estimated number of floating-point operations is

$(3.350818044098033/0.00001) * (3 + 1.0327625623730932/0.00001) \approx 34606999538$

4.2 Area of the heart

I analytically derived the heart's area as a sum of three integrals, so the only numerical calculation performed is the integrations. For the Simpson's Rule for integration method, the running time is proportional to the number of subdivisions the area is broken up into. Therefore, the time complexity is linear, $O(n)$. With $n = 10000000$ subdivisions, it took 13 seconds to calculate the approximate area.

The number of floating point operations can be estimated, with n being the number of subdivisions for Simpson's Rule:

1 (Δx calculation) + $2n$ (two operations in each loop iteration) + 1 (final division before returning) $\approx 2n + 2$ floating point operations. Therefore, with $n = 10000000$, the total number of floating-point operations needed is estimated to be

$2 * 10000000 + 2 = 2000002$

Problem 1.2

1 Problem Description

Use the bisection method to find the roots of the following equation:

$$f(x) = 2.020^{-x^3} - x^3 cos(x^4) - 1.984$$

2 Algorithm Description and Pseudo-Code

The function is the bisection method, implemented in Python. It takes 3 required arguments and 2 optional:

- f, the function one wishes to find a root of (required)
- a, the lower bound of the interval to search for a root (required)
- b, the upper bound of the interval to search for a root (required)
- ϵ, the desired precision (optional)
- n, the number of iterations to run the method (optional)

Pseudo-code:

```
bisection(f, a, b, $\epsilon$, n) {
    for i = 0 to n, do {
        x = $\frac{a+b}{2}$

        if f(x) == 0 or $\frac{b-a}{2} < \epsilon$ then return x

        if sign(f(x)) == sign(f(a)) then a = x

        else if sign(f(x)) == sign(f(b)) then b = x

        else no root is found, return nothing
    }
}
```

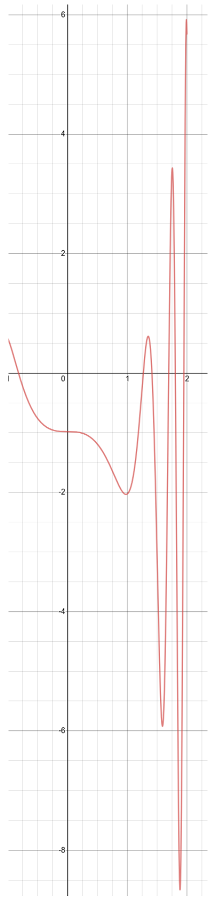
The algorithm takes in an interval $[a, b]$ to begin searching for a root and then begins iterating. It finds the midpoint, x, of $[a, b]$, and continuously narrows down the interval until it (1) finds the exact root or approximate root within the accepted error, ϵ, or (2) the signs of both $f(a)$ and $f(b)$ aren't equal to $f(x)$, at which point no root can exist and the program terminates.

3 Results

x_1 (Approximation)	δ	$[x_1 - \delta, x_1 + \delta]$	x_0 (Root)
-0.85	0.5	[-0.9, -0.8]	-0.8242859529505948
1.26	0.06	[1.2, 1.32]	1.269196365417447
1.4	0.1	[1.3, 1.5]	1.4142931303766089
1.7	0.05	[1.65, 1.75]	1.6955940658750484
1.8	0.05	[1.75, 1.85]	1.8067232403147502
1.95	0.05	[1.9, 2.0]	1.948277988686459

4 Brief comments on performance & other questions

The running time of this bisection function is proportional to the number of iterations given as an argument, N. Therefore, we can say that the function has an asymptotically worse-case running time of $O(N)$, i.e. linear time. Modern computers can execute billions of instructions a second, so this program will run fast as long as N isn't ridiculously large.
Graph of the function in interval $x \in [-1, 2]$:


6
4
2
0
1
2
-2
-4
-6
-8