# Homework 1: Numerical Integral and Differentiation

Ivan F. Valerio
Department of Physics, Astronomy, and Applied Physics,
Rensselaer Polytechnic Institute
valeri@rpi.edu

February 23, 2013

## 0.1 Assignment

1. Write a double-precision program to integrate an arbitrary function numerically using the trapezoid rule, the Simpson rule, and Gaussian quadrature (for GC, limit yourself to a simple 3 point approximation within each interval: use one interval only in this case). In the discussion of your results, a plot of the relative error $\epsilon = |\frac{\text{numerical}-\text{exact}}{\text{exact}}|$ for each approach should be included.

2. Compute the first derivative of $x^2$, $x^3$, $e^{-x}$, and another well-behaved, non-trivial, function of your choice, using forward difference, central difference, and 5-point approximation.

   (a) In each case plot the error as function of step size (consider using log-log scales)

   (b) In your discussion, make sure to talk about the best step size to use (Refer to the class notes).

   (c) Make sure you consider the particular cases where a given approximation is exact.

3. Write a C++ program that finds the zero of a function using the bisection method.

4. (optional, for 2 extra credits) Write a C++ program that finds the zero of a function using the newton-Raphson method.

# Chapter 1

# Integration

## 1.1 Algorithmic Considerations

There are various ways to compute the integral of a function. the first method involves deviding the area under the function into very thin trapezoid sections and adding them up. knowing the functional value of the two sides of such trapezoid and the width is enough information for computing such integrals. The second method uses something called Simpson's rule. It uses polynomials to fit the curve with more precision. This is condisedered to be an improvement over the first method, as this decreases the spaces left out when using the trapezoid method. The last method is called Gaussian Quadrature, which uses optimal abscissas to estimate the integral of a function. Each method has an advantage over the other when it comes to obtaining an integral, and some might be able to determine the integral exactly, that is if we not consider number representation and machine error in a computer.

## 1.2 Implementation and results

Let us first begin the discussion with the well known Trapezoid rule of integration. This method is very similar to the well known Riemann sum, except that instead of infinitely thin rectangles, we use infintely thin trapezoids, which would fit the function in question slightly better. in using such method, the algorithm used to do such calculations only requires us to care for the values at the ends of the function. We can represent this mathematically as follows:

$$\int f(x)dx \approx \sum_{i=1}^{N} w_i f_i \tag{1.1}$$

where

$$w_i = \{\frac{h}{2}, h, \cdots, h, \frac{h}{2}\} \tag{1.2}$$

Here h is simly the step size used on the integration. This is relatively simple to apply in a programing language, and we only really need a while loop that cares for the first and last values of the vector $w_i$. The algorithm used to implement this was a simple loop that would simply add up the values of the function, while still caring for the endpoints. This can be summerized as follows:

```
while true do
    if first value on the array then
        value of the function times half of step;
    end
    if last value of the array then
        add to the summation the value of the function times half the
        step;
    end
    if not within the range then
        break;
    else
        add value of the function times the step;
    end
end
```

**Algorithm 1:** algorithm for Trapezoidal Rule

Simpson's rule is a little harder to implement, as it requires more points. The integrand can be aproximated as follows:

$$\int_a^b f(x)dx \approx \frac{b-a}{2}\left(f(a) + 4f\Big(\frac{a+b}{2}\Big) + f(b)\right) \tag{1.3}$$

Simpson's rule follows a very similar loop when implementing this in C++, except for the fact that we do not need conditional statements for the ends of the steps. Because of this, Simpson's rule is more compact than the previous Trapezoidal integration, and easy to implement more generally.
Gaussian Quadrature, although more analytically complicated than the previously discussed methods, it is by far the simpliest to write in code, and as it turns out, the one with the most accuracy per operations. The Formula for computing an integral with Gaussian Quadrature is as follows:

$$\int_{-1}^{1} g(x)dx \approx \sum_{i=1}^{n} c_i g(x_i) \tag{1.4}$$

And the neccessary variable changes can be done using the following:

$$\int_a^b f(x)dx = \int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{b+a}{2}\right)\frac{b-a}{2}dx \tag{1.5}$$

the values of $c_i$ and $x_i$ have already been predetermined for the integral from -1 to 1.

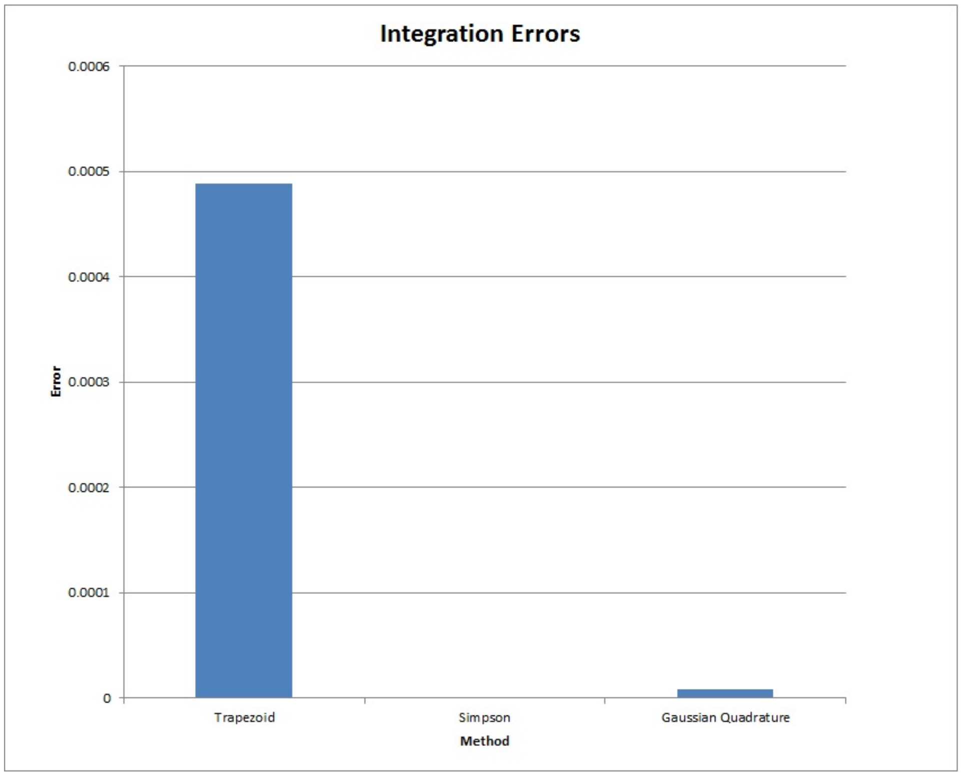

Figure 1.1: This is a bar graph of error for each method used

The methods discussed above were implemented on cosine from 0 to $\frac{\pi}{2}$. the analytical result is 1, and the computational methods were compared with this using the relative error given by $\epsilon = |\frac{\text{numerical}-\text{exact}}{\text{exact}}|$

| Method | Calculated value | Error |
|---|---|---|
| Trapezoid | 1.00048 | 0.000488085 |
| Simpson | 1 | $1.21394 \times 10^{-7}$ |
| Gaussian Quadrature | 1.00001 | $8.12209 \times 10^{-6}$ |

## 1.3 Discussion

We can clearly see a difference between the trapezoid method and Simpson's method. On the other hand, there isn't much of a difference between Simpson's method and the Gaussian Quadrature. This is due to the fact that Simpson's rule has analytical errors in the fourth order derivative, while the three point Gaussian Quadrature has errors in the third order derivative, but because the third term derivative of cosine is 0 in the taylor series, it truely begins in the fourth term like Simpson's rule.

# Chapter 2

# Differentiation

## 2.1 Algorithmic Considerations

The definition of a derivative is initially given as the slope of a line in a graph. this can be acheived by finding the difference in the value of the function at two points. the forward method is represented as follows:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{2.1}$$

Where h is the difference between the two points in question. It is apperent by the definition that the smaller h is, the closer we are to finding the actual value of the derivative at that point.
another method, called the Central method, places the point we wish to calculate the derivative for between the two points being used for the calculation. This is written:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h/2) - f(x-h/2)}{h} \tag{2.2}$$

The last method involves increasing the number of points used for the calculation. In turn the accuracy improves and we arrive at a more accurate result:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h} \tag{2.3}$$

## 2.2 implementation and results

To observe how step sizes affect the accuracy of the calculateion, a loop was established that would calculate the slope many times for an array of step sizes. This data is then stored in a dat file and graphed using Microsoft Excel. These methods of differentiation can be compared with each other by observing how they behave with various functions versus the step length. How accurate can these functions be? the results are shown in the following graphs:

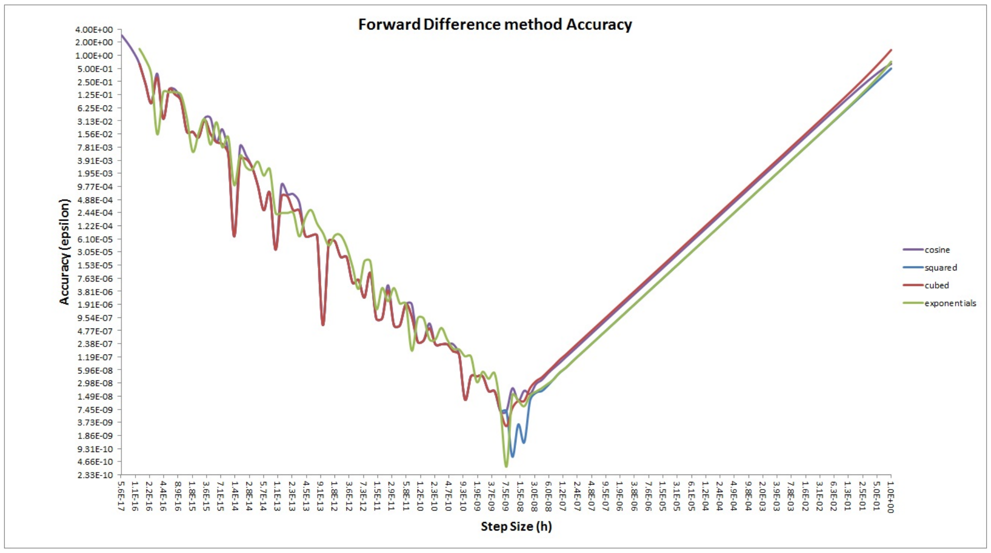

Figure 2.1: A graph of accuracy of method versus step size for the forward method of differentiation

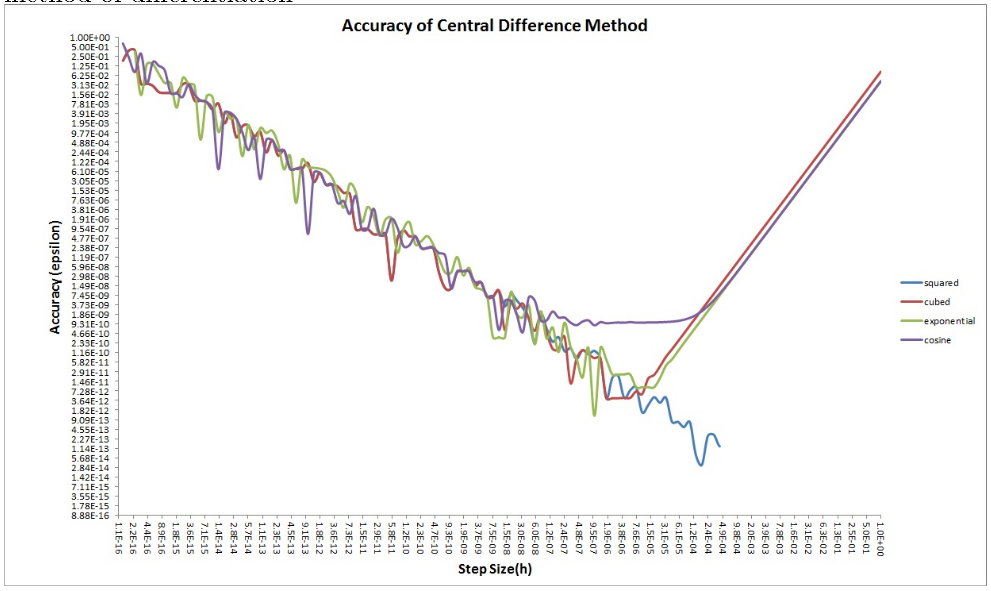

Figure 2.2: A graph of accuracy of method versus step size for the central method of differentiation

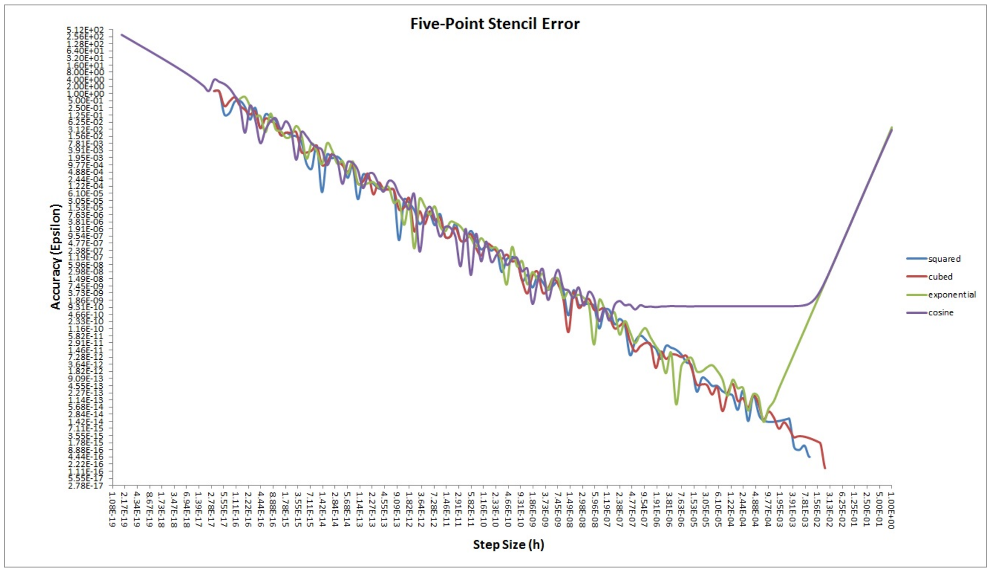

Figure 2.3: A graph of accuracy of method versus step size for the five-point stencil method of differentiation

We can see the change in accuracy as we decrease the step size between points.

## 2.3 Discussion

It is interesting to see that the accuracy does not continue to dicrease as the step size decreases. On the contrary, the error of the operation increases after reaching a certain minimum. This is due to the fact that the machine precision also plays a role in the calculations performed in the program. Because the computer's inability to represent decimals accurately and it's limit to store numbers smaller than $2.2 \times 10^{-16}$, the calculations made by the computer are a accumulated with uncertainties, resulting in the observed behaviour at the left side.
Another very important behavour is the result seen in both central and five-point stencil methods. Some functions do not seem to have an increase in error as $h$ increases. This is mostly due to the fact that analytical errors disappear with the use of the right method. What is the right method then? This all depends on the compatibility of the function. For example: The central method of differentiation can be derived to a result independent of h. This in turn means that no matter how far the points are from the point in question in a parabola, the result will always give you an accurate slope! other functions whose taylor expansions have zeros after the fourth order would certainly have virtually no error when using a method with analyticall error of that magnitude.

It also seems that despite the differences in accuracy, it would be best to use the square root of h as the step size. it seems that this would provide the most accurate representation for the most general case.

# Chapter 3

# Finding Zeros

## 3.1 Algorithmic Considerations

There are many uses to finding the point a function intersects the x-axis. The method described here is called Bisection method, and it uses guesses from the user to find the zero point between two points. The algorithm uses input from the user, an upper and lower bound, within which the intersection is expected to be found. The algorithm then begins to guess the location of the zero point by decreasing the size of such interval. This will eventually lead to an interval too small for the computer to represent numerically.

## 3.2 Implementation and Results

The algorithm used to implement this idea was composed of a loop that would decrease the size of said range and at the same time determine the size of it. this loop then breaks when the size is too small. This can can be represented as follows

```
while true do
    x = (leftbound - rightbound)/2;
    if abs(f(x)) < machineprecision then
        break;
    end
    if f(x) > 0 then
        leftbound = x;
    end
    if f(x) < 0 then
        rightbound = x;
    end
end
```

**Algorithm 2:** Finding zeros using bisection method.

Implementing this algorithm to cosine we find that the zero between 0.5 and 2.5 for cosine is $x = 1.5708$, which is equivalent to $\pi/2$.

## 3.3 Discussion

The efficiency is not very good, as it takes 850 steps to arrive to the answer mentioned previously using the above input range. This of course may be due to the fact that the range chosen was very large, but we can still see that this method may not be the most efficient way of finding the zero of a function. There are other problems that might arise from using this method, such as the fact that it depends on the machine precision. What if the function in question analytically never reaches zero, but it becomes small enough to fulfill the condition given above. A perfect example of this is the exponential function $\exp x$. If we were to go far enough in the negative values, eventually the machine will reach a value of the function smaller than $\epsilon$ and in turn we will see the function become zero. This indicates that to really understand the results given by the computer, one must also understand the limitations and be aware of the shortcomings of computing zero points on a computer.