Pro Quadrature Viewer ZX790

User's Guide

Jorge Almeyda, John Kluesner, Max Mays

May 6, 2013

Welcome to the user's guide for *Pro Quadrature Viewer ZX790.* This application has the power to find the area under a curve of a function efficiently using many different numerical methods. The source code was written in Python using the NUMPY and SCIPY libraries as well as QTDESIGNER for the General User Interface. The numerical methods were cheifly written by Jorge Almeyda and John Kluesner, while the user interface was created by Max Mays. This was the 2-month long project created by the trio for Dr. Robert Olsen's 2013 Spring term Python Computing class at the Richard Stockton College of New Jersey.

The numerical methods included are (in order of accuracy):

1. Monte Carlo Integration
2. Left Endpoint Riemman Summation
3. Right Endpoint Riemman Summation
4. Composite Trapezoidal Rule
5. Midpoint Riemman Summation
6. Composite Simpsons Rule
7. Gaussian-Legendre Quadrature

Pro Quadrature Viewer ZX790 has many applications such as: a teaching tool, a way to verify answers for homework problems, or comparing different numerical integration methods in terms of speed and accuracy for a problem you're working on.

Table of Contents

1. Getting Started

This section of the user's guide is written for people who just want to get things going and see some results. It's the recommended section to start with and almost a prerequisite to using *Pro Quadrature Viewer ZX790*. If you don't care about the mathematical underpinnings of the numerical methods, then it's the only section you'll need to read. The math is fun though, so you really should at least glaze over it!

i. Using the GUI

The *Pro Quadrature Viewer ZX790* interface has been designed for ease of use and dynamic comparison of the various quadrature methods supported by the program. In general, the program is broken up into the following sections and components:

1. **Function Graph** The graph of the function, as well as the polygons used for quadrature

2. **Function Input** Used to input the function to be graphed, as well as its (optional) antiderivative for error analysis

3. **Graph Result and Error** Result and error for the function, using the active quadrature method (selected by the active method selector)

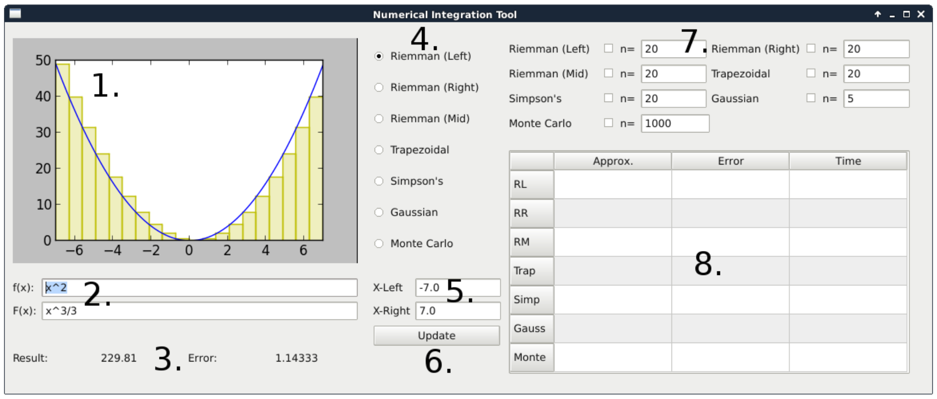
4. **Active Method Selector** Used to select the quadrature method to use when graphing the function

5. **X Bound Selector** Used to define the left and right x bounds for use in quadrature

6. **Update Button** When clicked, this updates the GUI to reflect new selections made and updated values

7. **Method Selection** Used to select which methods should have results displayed in the table, as well as defining n values for each quadrature method

8. **Results Table** Shows results, error (if applicable), and time taken for each quadrature method that was checked off in method selection



f(x): x^2

F(x): x^3/3

Result: 229.81 Error: 1.14333

[✓ Riemman (Left)] Riemman (Right) Riemman (Mid) Trapezoidal Simpson's Gaussian Monte Carlo

X-Left -7.0

X-Right 7.0

Riemman (Left)	n= 20	Riemman (Right)	n= 20
Riemman (Mid)	n= 20	Trapezoidal	n= 20
Simpson's	n= 20	Gaussian	n= 5
Monte Carlo	n= 1000		

	Approx.	Error	Time
RL			
RR			
RM			
Trap			
Simp			
Gauss			
Monte			

A labeled overview of the UI. Numbers correspond to the elements listed above.

To better explain how elements work, it will be good to go through an example workflow, in which we will define a function, its antiderivative, choose different methods to graph, try different x and n values, and finally compare different methods to see which is the fastest and most accurate. Along the way, there will be some helpful notes and caveats about the GUI elements, so this will serve as a good tutorial and reference as well as an example.

To begin, let's define a new function to graph, perhaps the cubic function $x^3 + 2x^2 + 3$ which has an antiderivative defined as $\frac{x^4}{4} + 2\frac{x^3}{3} + 3x$. The first step will be to define the function for the program. Under the graph, find the label `f(x)` and input `x^3+2*x^2+3` into the text box. Input the function's antiderivative, `x^4/4+2*(x^3/3)+3x` into the text box below, labeled `F(x)`. Now, look for the `Update` button to the right of the function inputs, and press it. You should see the function and the quadrature method's polygons that it used to integrate it displayed in the graph view, as well as the result and error for integration using that quadrature method under the function inputs. Before we continue, here's a few notes about function input:

1. Express powers using the ^character, even though this a python program. The program will turn the ^characters into valid ** python operators.

2. Besides the above caveat about powers, functions should use valid python expressions (i.e. `3*x` is correct, whilst `3x` will not be parsed correctly and you'll get an error

3. Whitespace is allowed, the program has no trouble parsing it

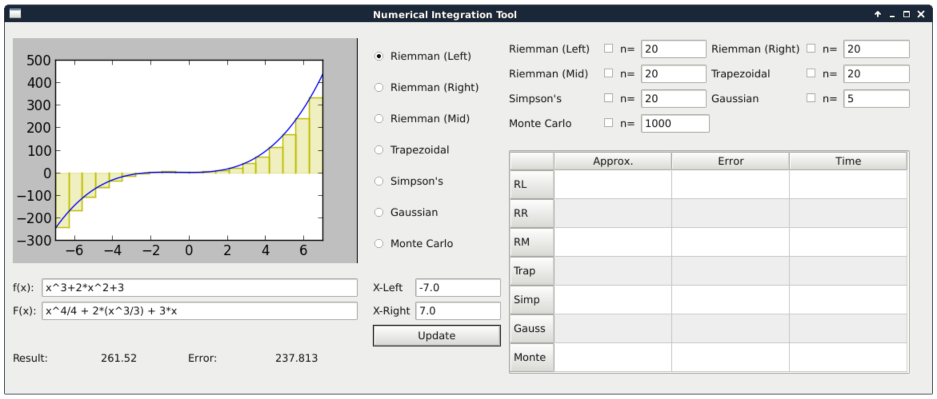
4. You can use parentheses as you would in a real mathematical expression to affect the order of operations, as well as just to make inputs easier on the eyes

5. You can use any valid numpy mathematical functions and constants in your function definition (i.e. `sin(x+pi)` is valid, as is `log(x)`. You may get some terminal errors when you graph functions that are not defined for all real x values (like `log`), but this will not interfere with the graph or the results.

6. If you know the exact value of the integral, $\mathcal{I}$, you're trying to approximate, but not the actual anti-derivative, then use

$$F(x) = \frac{\mathcal{I}}{b-a}x$$

in place of the actual anti-derivative.



Riemman (Left) [✓]
Riemman (Right)
Riemman (Mid)
Trapezoidal
Simpson's
Gaussian
Monte Carlo

Riemman (Left)	☐ n=	20	Riemman (Right)	☐ n=	20
Riemman (Mid)	☐ n=	20	Trapezoidal	☐ n=	20
Simpson's	☐ n=	20	Gaussian	☐ n=	5
Monte Carlo	☐ n=	1000			

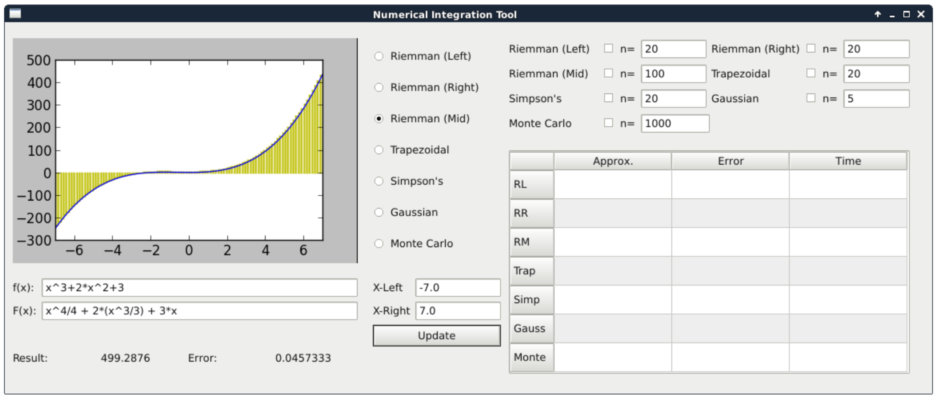
	Approx.	Error	Time
RL			
RR			
RM			
Trap			
Simp			
Gauss			
Monte			

f(x): x^3+2*x^2+3
F(x): x^4/4 + 2*(x^3/3) + 3*x
X-Left -7.0
X-Right 7.0
Result: 261.52 Error: 237.813

Inputting the function and updating the graph.

Now that we know about function input, it's time to address the ugly elephant in the room: the error for the quadrature method we used on this function is huge! Riemman methods aren't exactly famous for their accuracy, especially ones which use the left or right endpoint. We can see from the graph just how inaccurate that is. We'd probably get a better result using the midpoint. The method that is currently being graphed is called the *active* quadrature method, and we can change it by selecting one of the radio buttons to the right of the graph. Select `Riemman (Mid)` from the list, and hit the update button. The graph will update to reflect the current method, and so will the results below the function inputs. That error is much better, but let's improve it even further! In quadrature, the more polygons you use, the more accurate the results. We call the number of polygons to use n. Let's increase our n value to get even more accurate results. To the right of the radio buttons is the list of methods again, but with an input for the n value and a checkbox. We'll ignore the checkboxes for now, but we'll use the text input to change the n value. Note that each method has its own n input: that's because some methods have special restrictions on the n values they can use: the details of that will be explained in the section after this one. With all that being said, look for the label `Riemman (Mid)` and go to the text box to its right. Change n from 20 to 100 and hit update. The graph will update along with the results, which include a much lower error. Before we move on, let's discuss some of the special n limits in *Pro Quadrature Viewer ZX790*:

1. ALL methods have a maximum limit of 5,000,000
2. n must be greater than or equal to 1, for obvious reasons
3. Simpsons' Rule only accepts even n values.
4. Gaussian quadrature only accepts n up to 16
5. Any attempt to set to n to an illegal value will cause a reset to a legal value



Riemman (Left)	n= 20	Riemman (Right)	n= 20		
Riemman (Mid)	n= 100	Trapezoidal	n= 20		
Simpson's	n= 20	Gaussian	n= 5		
Monte Carlo	n= 1000				

f(x): x^3+2*x^2+3

F(x): x^4/4 + 2*(x^3/3) + 3*x

X-Left -7.0

X-Right 7.0

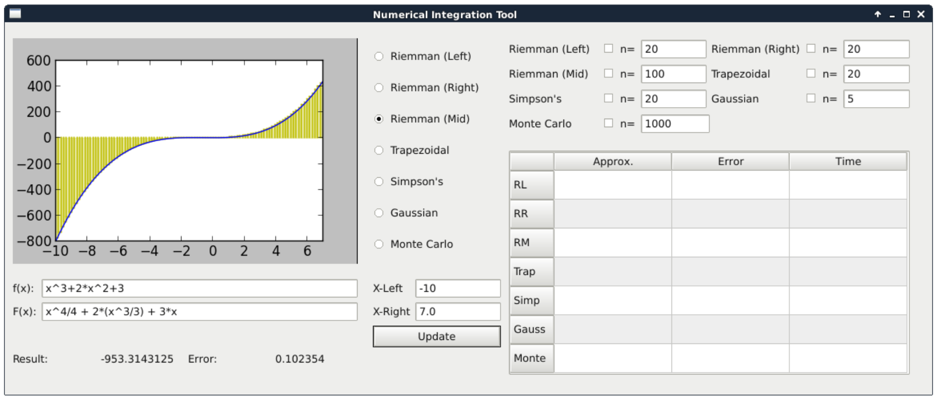
Result: 499.2876 Error: 0.0457333

	Approx.	Error	Time
RL			
RR			
RM			
Trap			
Simp			
Gauss			
Monte			

Updating the n value for the quadrature method

You can also change the x bounds to integrate over different intervals. You can set both the left and right x bounds in the two text boxes located above the update button. There's not much to it. The only thing to keep in mind is that if you set the left x bound to be greater than the right, then the x-axis on the graph will be reversed, and the numerical results will reflect that switch.

Try making the interval non-symmetrical by setting the left x bound to -10, and see how that changes the result and the graph.



Riemman (Left) / Riemman (Right) / [✓ Riemman (Mid)] / Trapezoidal / Simpson's / Gaussian / Monte Carlo

Riemman (Left)	☐	n= 20	Riemman (Right)	☐	n= 20
Riemman (Mid)	☐	n= 100	Trapezoidal	☐	n= 20
Simpson's	☐	n= 20	Gaussian	☐	n= 5
Monte Carlo	☐	n= 1000			

f(x): x^3+2*x^2+3

F(x): x^4/4 + 2*(x^3/3) + 3*x

X-Left: -10

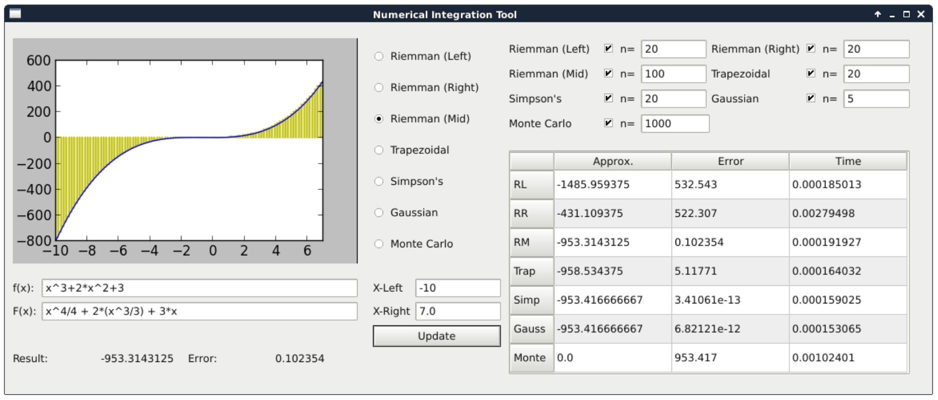
X-Right: 7.0

Result: -953.3143125 Error: 0.102354

	Approx.	Error	Time
RL			
RR			
RM			
Trap			
Simp			
Gauss			
Monte			

Changing x bounds for the interval

There's one final part of the program to talk about: the results table off to the right. This table can be used to run multiple quadrature methods on the function at the same time and compare results. The labels on the left side of the table correspond to the abbreviations of the quadrature methods. Let's compare all of the methods at the same time. To mark a quadrature method for display, go to the list above the table (where we previously inputted n values) and mark each checkbox. Each method that is checked will be run on the next update and the results displayed in the table. The integration result and the time taken to run the method will always be shown, along with the error value if an antiderivative is set. After checking all the methods, hit update to populate the results table. Now, you can compare methods to see which is the most accurate and which is the fastest. Remember that you don't have to check all of them: you can check only the ones which you want to know about if you so desire.



Riemman (Left) / Riemman (Right) / [✓ Riemman (Mid)] / Trapezoidal / Simpson's / Gaussian / Monte Carlo

Riemman (Left)	✓	n= 20	Riemman (Right)	✓	n= 20
Riemman (Mid)	✓	n= 100	Trapezoidal	✓	n= 20
Simpson's	✓	n= 20	Gaussian	✓	n= 5
Monte Carlo	✓	n= 1000			

f(x): x^3+2*x^2+3

F(x): x^4/4 + 2*(x^3/3) + 3*x

X-Left: -10

X-Right: 7.0

Result: -953.3143125 Error: 0.102354

	Approx.	Error	Time
RL	-1485.959375	532.543	0.000185013
RR	-431.109375	522.307	0.00279498
RM	-953.3143125	0.102354	0.000191927
Trap	-958.534375	5.11771	0.000164032
Simp	-953.416666667	3.41061e-13	0.000159025
Gauss	-953.416666667	6.82121e-12	0.000153065
Monte	0.0	953.417	0.00102401

Populating the table with results from multiple methods

ii. Limits and Parameters of the Numerical Methods

Riemman Summations

Left-Endpoint, Right-Endpoint, and Midpoint Riemman Summations work by creating rectangles using the height at the left-most, right-most, or mid-point, respectfully, over an interval $[a, b]$. The interval is divided into n subintervals of the same length to create n rectangles. The input from the GUI is the number of rectangles, or subintervals. The formulas are:

Left-endpoint:

$$\int_a^b f(x)\,dx = h\sum_{i=0}^{n-1} f(x_i) + \frac{h}{2}(b-a)f'(\mu),$$

Right-endpoint:

$$\int_a^b f(x)\,dx = h\sum_{i=1}^{n} f(x_i) - \frac{h}{2}(b-a)f'(\mu),$$

Midpoint:

$$\int_a^b f(x)\,dx = h\sum_{i=0}^{n-1} f(x_i) + \frac{h^2}{24}(b-a)f''(\mu),$$

where $\mu \in [a, b]$.

Left-endpoint and Right-endpoint can only integrate a polynomial of degree 0 (constant) perfectly. On the other hand, Midpoint Riemman can calculate polynomials of degree 1 (linear) perfectly.

Monte Carlo Integration

Monte Carlo is the only method of approximation that we used that is not a quadrature method. It works in this way:

Imagine that you throw darts at the graph of a function. Some darts would land in the area we want to calculate and the rest would land elsewhere (inside the bounding box). We use that to calculate

$$\frac{\textit{Number of darts in the area}}{\textit{Total number of darts}}.$$

That ratio is approximately $\frac{\textit{Area between the curve and the x-axis}}{\textit{Area taken up by the bounding box}}$, so we multiply by the area of the bounding box to get the area between the curve and the x-axis.

The n value obtained from the user will represent the number of darts. This number will be much higher than the n value for the quadrature methods. The logic is the same though, the higher n is, the more accurate the approximation will be. The user must keep in mind that the landing spots of the darts are randomly generated, which means that the precision of the method is random too. Sometimes, even with a high number of darts, the approximation is not great. We suggest to run this method several times and then calculate an average of the calculations.

Composite Trapezoidal Rule

The Composite Trapezoidal Rule is an intuitive method of finding the area under the curve. It approximates the area with a series of trapezoids that lie evenly spaced over the interval $[a, b]$. In our case, the interval $[a, b]$ is subdivided into n subintervals $[x_n, x_{n+1}]$ of width $h = \frac{b-a}{n}$ by using the equally spaced nodes $x_n = a + ih$ for $i = 0, 1, ..., n$ using the following formula:

$$\int_a^b f(x)\,dx = \frac{h}{2}\Big(f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\Big) - \frac{h^2}{12}(b-a)f''(\mu).$$

where $\mu \in [a, b]$.

Keep in mind that the number n provided as input is the number of trapezoids used for the approximation. This means the interval $[a, b]$ would be divided into n subintervals using $n+1$ evaluations. This method can integrate any polynomial of degree 1 (linear) perfectly.

Composite Simpsons Rule

Composite Simpsons Rule is a quadrature method that creates quadratic fits using sets of 3 points at a time. Since it uses a set of 3 points, the number of evaluations will always be odd and the number of fits will be even. If the n value provided by the user is not even, the code will calculate a new n value, $n-1$. The input from the application is 2 times the number of quadratic fits. The formula for Composite Simpsons Rule is:

$$\int_a^b f(x)\,dx = \frac{h}{3}\Big(f(x_0) + 4\sum_{i=0}^{\lfloor\frac{n}{2}\rfloor-1} f(x_{2i+1}) + 2\sum_{i=1}^{\lfloor\frac{n}{2}\rfloor-1} f(x_{2i}) + f(x_n)\Big) - \frac{h^4}{180}(b-a)f^{(4)}(\mu),$$

where $\mu \in [a, b]$.

It would appear that this method can only integrate polynomials of degree 2 perfectly, but it actually perfectly approximates the integral of any polynomial of degree 3.

Gaussian-Legendre Quadrature

The main difference between this method and the others is that it does not use equally spaced subintervals. It instead chooses the most optimal values in the interval. It uses the general formula

$$\int_{-1}^{1} f(x)\,dx = \sum_{k=0}^{n} c_k f(x_k),$$

where x_k is a root of the n^{th} Legendre Polynomial and the coefficients are the Lagrange Polynomial weights integrated from -1 to 1, $c_k = \displaystyle\int_{-1}^{1} \prod_{\substack{j=0 \\ j\neq k}}^{n} \frac{x - x_j}{x_k - x_j}\,dx.$

As we can see in the formula, the limits of the interval are -1 and 1 which means that we need to apply the linear transformation:

$$\begin{aligned} z &= \frac{b-a}{2}x + \frac{b+a}{2}, \\ dz &= \frac{b-a}{2}dx, \end{aligned}$$

which yields the final equation

$$\int_{b}^{a} f(z)dz \quad = \quad \frac{b-a}{2}\sum_{i=k}^{n} c_k f(\frac{b-a}{2}x_k + \frac{b+a}{2}).$$

Gaussian-Legendre Quadrature uses a list of pre-calculated optimal nodes (roots) and coefficients to approximate the integral. The code to calculate the optimal values have been avoided because the values are well known and are available to anyone.

This method can perfectly integrate any polynomials of degree $2n - 1$, where n is the number of roots and coefficients used for the approximation. *Pro Quadrature Viewer ZX790* stores n values up to 16, which would allow it to perfectly integrate any polynomial up to degree 31.

2. Mathematical Background

This section is for the user who wants to see where these methods come from. A prerequisite to understanding the methods would be Calculus 1 and 2. In particular: definite integrals, derivatives, and Taylor Expansions.

There are 4 types of numerical integration included in *Pro Quadrature Viewer ZX790*: Stochastic Methods, Composite Newton-Cotes Quadrature, Riemman Summations, and Gaussian Quadrature.

Stochastic Methods work by using random numbers to approximate a result. They are useful because often times they are iterative in nature. This means that if your current result isn't good enough, you can just do more iterations to get a better result. Their weakness is that they often take more time to get the same results as a more sophisticated method. The Stochastic Method we decided to include is Monte Carlo Integration.

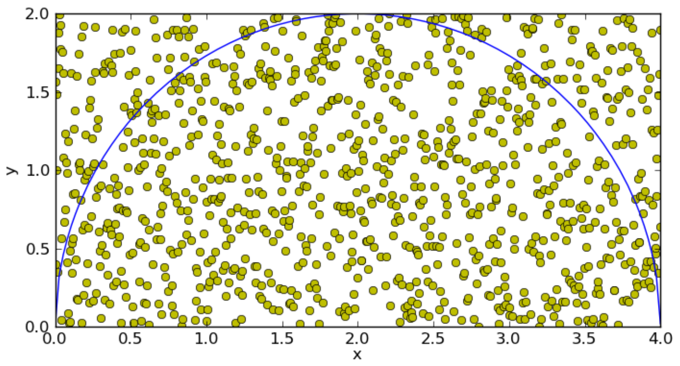
Composite Newton-Cotes Quadrature Methods work by using evenly spaced intervals to approximate the area under the curve in the form of a summation. That is, they use $\int_a^b f(x)\,dx \approx \sum_{i=k}^{n} c_i f(x_i)$, where each x_i is an equal distance from eachother. The Composite Newton Cotes methods we decided to include are the Trapezoidal Rule and Simpsons Rule. Riemman summations are also special cases of Composite Newton-Cotes Quadrature methods that are presented in most introductory Calculus courses.

The Gaussian Quadrature method we decided to include is Gaussian-Legendre Quadrature. It's the most common form of Gaussian Quadrature used and it's often the best. It is similar to a Newton Cotes method in that it approximate an integral with a summation, $\int_a^b f(x)\,dx \approx \sum_{i=k}^{n} c_i f(x_i)$, but the difference is the x_i's are often times not equally spaced. It cleverly picks the best x_i values to get best approximation. This allows one to arrive at better approximations with less work.

It should be noted that even though the mathematics presented in this user's guide arrives at the correct results, it abstracts away a lot of mathematical details.

i. Monte Carlo Integration

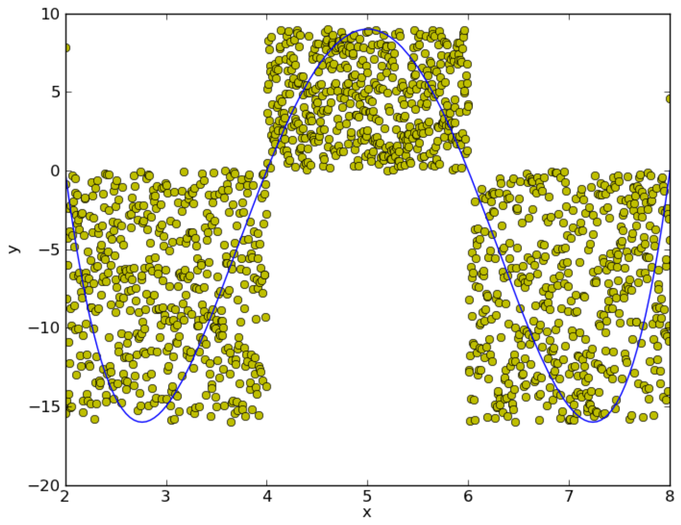
One can think of Monte Carlo Integration as randomly throwing a bunch of darts at the graph of a function. The points the darts land at are then compared to the actual value of the function and sees if it's between the curve and the x-axis. Then, the ratio of the area taken up by the curve and the curve's bounding box is approximated by dividing the number of darts between the curve and the x-axis by the total number of darts. From here, we can multiply that ratio by the area of the bounding box to get an approximate area under the curve. Here's an example using 1000 darts to approximate $\int_0^4 \sqrt{4-(x-2)^2}\,dx$.



An example of Monte Carlo Integration approximating $\int_0^4 \sqrt{4-(x-2)^2}\,dx$ with 1000 darts.

One can see that using a large number of darts leads to a pretty good approximation of the area under the curve. For further proof, the above example gave the approximate integral of 6.2238414358. The actual value should be half the area of circle with radius of 2, or $2\pi \approx 6.283185307179586$.

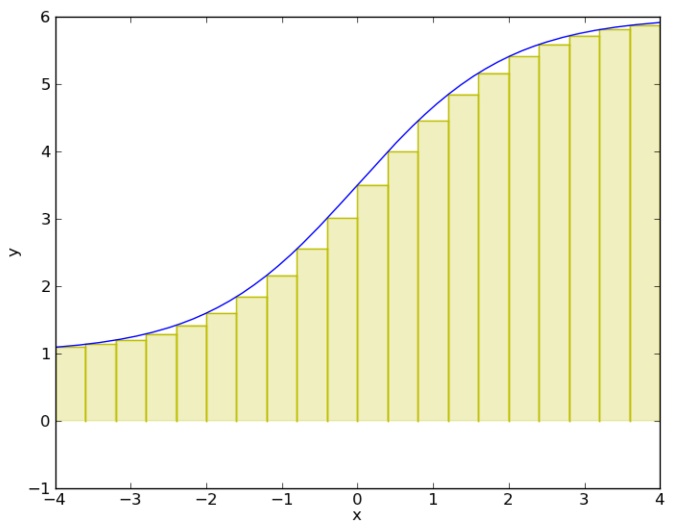
Often times, this theory is presented with strictly positive functions. Indeed, that restriction still shows the core theory and understanding of the method. How to apply the same theory to a function that jumps from negative to positive values is a little trickier. Our approach is to do different Monte Carlos Integrations over intervals that are strictly positive and strictly negative. Here's an example that uses 1500 darts to approximate $\int_0^8 (x-2)(x-4)(x-6)(x-8)\,dx$.



An example of Monte Carlo Integration approximating $\int_0^8 (x-2)(x-4)(x-6)(x-8)\,dx$ with 1500 darts.

ii. Left Endpoint Riemman Summation

This method is the one that is most often taught in a Calculus 1 course. It approximates the area under the curve by a series of rectangles from the left endpoint of the integral to the right endpoint of the integral. Here's an example of what it looks like when approximating the integral of a logistic equation, $\displaystyle\int_{-4}^{4} \frac{5}{1+e^{-x}} + 1\,dx$.



An example of a Left Endpoint Riemman Summation approximating $\int_{-4}^{4} \frac{5}{1+e^{-x}} + 1\,dx$ with 20 rectangles.

Next, we will make a proof outline of The Left Endpoint Riemman Summation to get an error term. First, we approximate our function, $f(x)$, with a Taylor Expansion around the point $x_0 = a$. This gives us

$$f(x) = f(a) + f'(\epsilon)(x-a), \epsilon \in [a,b].$$

We then integrate the left and right sides from a to b and obtain

$$\begin{aligned}\int_a^b f(x)\,dx &= \int_a^b f(a) + f'(\epsilon)(x-a)\,dx \\ &= f(a)x + \frac{f'(\epsilon)(x-a)^2}{2}\Bigg|_a^b \\ &= f(a)(b-a) + \frac{f'(\epsilon)(b-a)^2}{2}\end{aligned}$$

If we let the step size, $h = (b-a)$, we get

$$\int_a^b f(x)\,dx = hf(a) + \frac{f'(\epsilon)h^2}{2}.$$

The term $\frac{f'(\epsilon)h^2}{2}$ is called the remainder term. It can be used to find a bound for the error of the method.

The way we have it set up now is for a single rectangle to approximate the area under the curve, which is obviously a horrible approximation. What we do from there to get to the familar Riemman Sum is split the interval $[a, b]$ into n equally spaced subintervals with the step size $h = \frac{b-a}{n}$. We can denote this in summation form using $x_i = a + ih$ and discarding the error term as

$$\int_a^b f(x)\,dx = h\sum_{i=0}^{n-1} f(x_i).$$

Next, we handle the error term, denoted by R. The error term gets summed the same way as above, that is

$$R = \sum_{i=0}^{n-1} \frac{f'(\epsilon_i)h^2}{2}.$$

We define μ such that $f'(\mu) = \max_{i=0..n-1}(f'(\epsilon_i))$. Then,

$$\begin{aligned} R &= \sum_{i=0}^{n-1} \frac{f'(\epsilon_i)h^2}{2} \\ &\leq \sum_{i=0}^{n-1} \frac{f'(\mu)h^2}{2} \\ &= n\frac{f'(\mu)h^2}{2} \\ &= \frac{b-a}{h} \cdot \frac{f'(\mu)h^2}{2} \\ &= \frac{h}{2}(b-a)f'(\mu). \end{aligned}$$

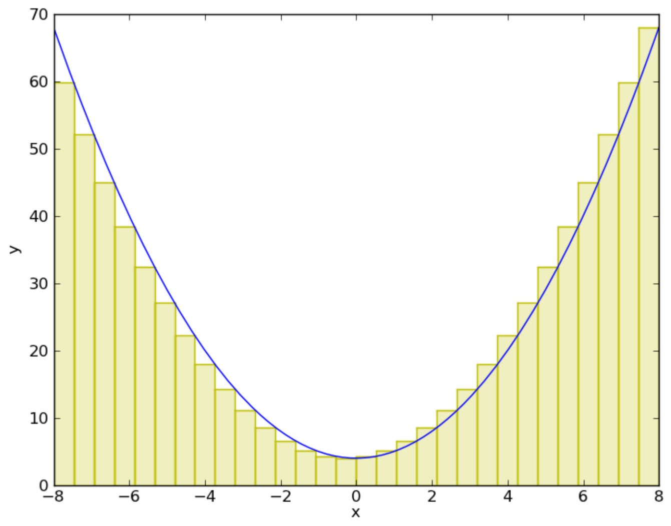
Thus, our final form for the Left Endpoint Riemman Summation is

$$\int_a^b f(x)\,dx = h\sum_{i=0}^{n-1} f(x_i) + \frac{h}{2}(b-a)f'(\mu).$$

Before you continue, take a closer look at that error term, specifically at $f'(\mu)$. What we want is for the error term to be as close to zero as possible. What we can do is pick a function $f(x)$ such that $f'(x) = 0$. It turns out that one such function is a constant Polynomial, like $f(x) = 5$ or $f(x) = \pi$. This sounds silly now, but what if we could generate a Quadrature method which has an error term of something like $f^{(4)}(\mu)$. That would mean that our Quadrature method could approximate any cubic Polynomial perfectly. It turns out there are many quadrature methods that can do this and we will examine some later.

iii. Right Endpoint Riemman Summation

Right Endpoint Riemman Summation is another Quadrature method that is often introduced in a Calculus 1 course. Here's an example for approximating $\int_{-8}^{8} x^2 + 4\,dx$ with 30 rectangles.



An example of a Right Endpoint Riemman Summation approximating $\int_{-8}^{8} x^2 + 4\,dx$ with 30 rectangles.

Through a similar process to above, we can arrive at the formula

$$\int_a^b f(x)\,dx = h\sum_{i=1}^{n} f(x_i) - \frac{h}{2}(b-a)f'(\mu),$$

where $x_i = a + ih$ and $\mu \in [a,b]$.

iv. Composite Trapezoidal Rule

Before we look at the Trapezoidal Rule, let's look back at the formulas for the Right and Left Endpoint Riemman Summations:

$$\begin{aligned} Q_L(x_i) &= h\sum_{i=0}^{n-1} f(x_i) + \frac{h}{2}(b-a)f'(\mu_L) \\ Q_R(x_i) &= h\sum_{i=1}^{n} f(x_i) - \frac{h}{2}(b-a)f'(\mu_R) \end{aligned}$$

If we pretend that $\mu_L \approx \mu_R = \mu$, then what would happen if we added them? The remainder terms would cancel out! That means we can go into the next term of the Taylor Expansion and use this new method to approximate any linear Polynomial. We effectively doubled the approximated area though, so a better idea would be to average $Q_L(x_i)$ and $Q_R(x_i)$. Or, if we let $T(x_i)$ be our new approximation:

$$\begin{aligned} T(x_i) &= \frac{Q_L(x_i) + Q_R(x_i)}{2} \\ &\approx \frac{h\sum_{i=0}^{n-1} f(x_i) + \frac{h}{2}(b-a)f'(\mu) + h\sum_{i=1}^{n} f(x_i) - \frac{h}{2}(b-a)f'(\mu)}{2} \\ &= \frac{h\sum_{i=0}^{n-1} f(x_i) + h\sum_{i=1}^{n} f(x_i)}{2} \\ &= \frac{h}{2}\left(\sum_{i=0}^{n-1} f(x_i) + \sum_{i=1}^{n} f(x_i)\right) \\ &= \frac{h}{2}\Big((f(x_0) + f(x_1) + f(x_2) + \cdots + f(x_{n-1})) + (f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + f(x_n))\Big) \\ &= \frac{h}{2}\Big(f(x_0) + (2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1})) + f(x_n)\Big) \\ &= \frac{h}{2}\Big(f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\Big). \end{aligned}$$

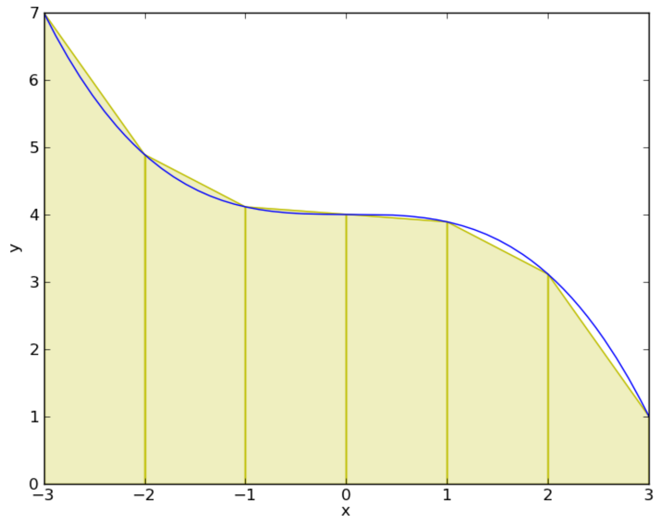
This is called the Composite Trapezoidal Rule. The method of getting a better approximation from combining 2 or more worse approximations is called extrapolation and is a very common and useful technique in numerical analysis and mathematics in general. Through another method[1] , we can obtain a remainder term and the general form of the Composite Trapezoidal Rule,

[1]The formula is taken from the 9th edition introductory Numerical Analysis textbook by Richard L. Burden and J. Douglas Faires, page 206.

$$\int_a^b f(x)\,dx = \frac{h}{2}\Big(f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\Big) - \frac{h^2}{12}(b-a)f''(\mu).$$

Looking at our remainder term, our intuition was correct and the Composite Trapezoidal Rule indeed approximates any linear polynomial perfectly.

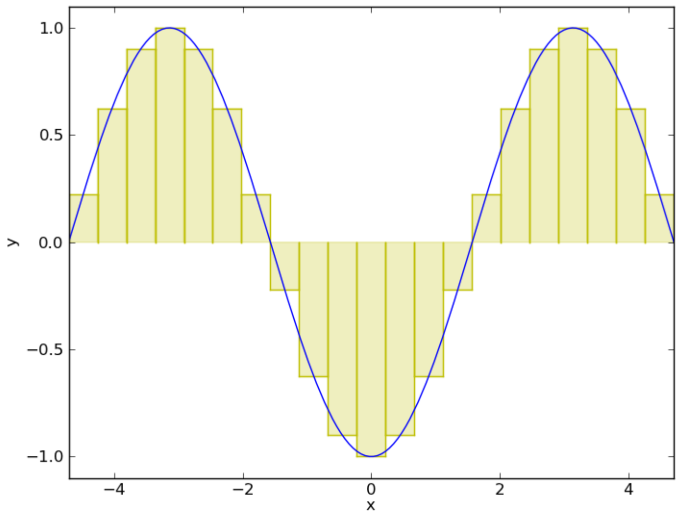
Additionally, the reason why it's called the Trapezoidal Rule is because it effectively creates trapezoids from adjacent points of the curve to approximate the area. Here's an example for approximating $\int_{-3}^{3} -\frac{x^3}{9} + 4\,dx$ with 6 trapezoids:



An example of the Composite Trapezoidal Rule approximating $\int_{-3}^{3} -\frac{x^3}{9} + 4\,dx$ with 6 trapezoids.

v. Midpoint Riemman Summation

The Midpoint Riemman Summation works similarly to the Left and Right Endpoint Summations. It chooses instead to make a rectangle with height from middle of the interval instead of the left or right. Here's an example for approximating $\int_{-\frac{3}{2}\pi}^{\frac{3}{2}\pi} -\cos(x)\,dx$:



An example of a Midpoint Riemman Summation approximating $\int_{-\frac{3}{2}\pi}^{\frac{3}{2}\pi} -\cos(x)\,dx$ with 21 rectangles.

Naively, one would think that a Midpoint Riemman Summation would be no better than it's Left or Right Endpoint counterparts. Actually though, it works better than the Composite Trapezoidal Rule. To see why, let's do a proof outline similar to the way we did it with the Left Endpoint Riemman Summation.

First, let's create a Taylor Polynomial expanded around the midpoint of a and b, which is $x_0 = \frac{a+b}{2}$,

$$f(x) = f(x_0) + f'(x_0)\Big(x - \frac{a+b}{2}\Big) + f''(\epsilon)\frac{(x - \frac{a+b}{2})^2}{2},$$

where $\epsilon \in [a, b]$.

Note that $b - \frac{a+b}{2} = \frac{b-a}{2}$ and $a - \frac{a+b}{2} = -\frac{b-a}{2}$. Then, we integrate both sides from a to b with respect to x,

$$\begin{aligned}
\int_a^b f(x)\,dx &= \int_a^b f(x_0) + f'(x_0)\Big(x - \frac{a+b}{2}\Big) + f''(\epsilon)\frac{(x-\frac{a+b}{2})^2}{2}\,dx \\
&= f(x_0)x + f'(x_0)\frac{(x-\frac{a+b}{2})^2}{2} + f''(\epsilon)\frac{(x-\frac{a+b}{2})^3}{6}\Bigg|_a^b \\
&= f(x_0)(b-a) + f'(x_0)\left(\frac{(b-\frac{a+b}{2})^2}{2} - \frac{(a-\frac{a+b}{2})^2}{2}\right) \\
&\quad + f''(\epsilon)\left(\frac{(b-\frac{a+b}{2})^3}{6} - \frac{(a-\frac{a+b}{2})^3}{6}\right) \\
&= f(x_0)(b-a) + f'(x_0)\left(\frac{(\frac{b-a}{2})^2}{2} - \frac{(-\frac{b-a}{2})^2}{2}\right) \\
&\quad + f''(\epsilon)\left(\frac{(\frac{b-a}{2})^3}{6} - \frac{(-\frac{b-a}{2})^3}{6}\right) \\
&= f(x_0)(b-a) + f'(x_0)\left(\frac{(\frac{b-a}{2})^2}{2} - \frac{(\frac{b-a}{2})^2}{2}\right) \\
&\quad + f''(\epsilon)\left(\frac{(\frac{b-a}{2})^3}{6} + \frac{(\frac{b-a}{2})^3}{6}\right) \\
&= f(x_0)(b-a) + f''(\epsilon)\frac{(\frac{b-a}{2})^3}{3} \\
&= f(x_0)(b-a) + f''(\epsilon)\frac{(b-a)^3}{24}.
\end{aligned}$$

From here, we break $b-a$ into n subintervals of size $h = \frac{b-a}{n}$ and define μ such that $f''(\mu) = \max_{i=0..n-1} f''(\epsilon_i)$ and $x_i = a + \frac{h}{2} + ih$. We wind up with

$$\begin{aligned}
\int_a^b f(x)\,dx &= h\sum_{i=0}^{n-1} f(x_i) + \sum_{i=0}^{n-1} f''(\epsilon_i)\frac{h^3}{24} \\
&\leq h\sum_{i=0}^{n-1} f(x_i) + \sum_{i=0}^{n-1} f''(\mu)\frac{h^3}{24} \\
&= h\sum_{i=0}^{n-1} f(x_i) + nf''(\mu)\frac{h^3}{24} \\
&= h\sum_{i=0}^{n-1} f(x_i) + \frac{b-a}{h}\cdot f''(\mu)\frac{h^3}{24} \\
&= h\sum_{i=0}^{n-1} f(x_i) + \frac{h^2}{24}(b-a)f''(\mu).
\end{aligned}$$

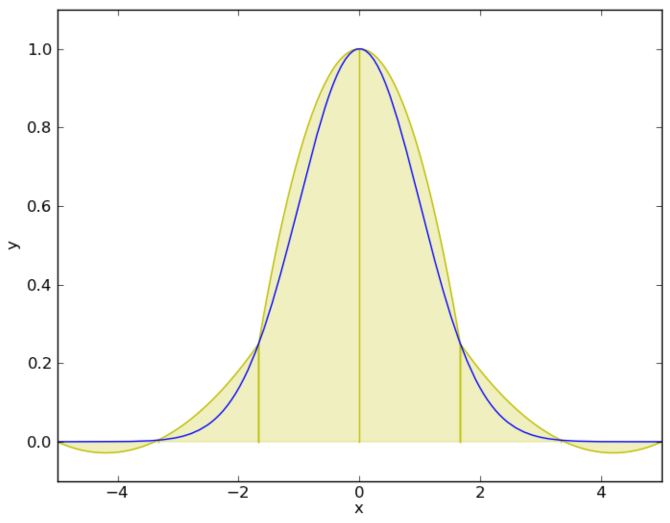
Finally, our approximation using a Midpoint Riemman Summation is

$$\int_a^b f(x)\,dx = h\sum_{i=0}^{n-1} f(x_i) + \frac{h^2}{24}(b-a)f''(\mu).$$

By the remainder term, we can see that it approximates any linear polynomial perfectly. Also, note that the remainder term of the Composite Trapezoidal Rule is twice that of a Midpoint Riemman Summation. This is why a Midpoint Riemman Summation is better than the Composite Trapezoidal Tule theoretically and in practice (try it if you still don't believe me!).

vi. Composite Simpsons Rule

What the Composite Simpsons Rule does is create a quadratic fit between adjacent sets of three adjacent points. So, if the points to be evaluated by the Composite Simpsons Rule were x_0, x_1, x_2, x_3, and x_4, it would create a quadratic fit for x_0, x_1, and x_2, and a second one for x_2, x_3, and x_4. Here's an example of the Composite Simpsons Rule approximating an integral of a Gaussian Function $\int_{-5}^{5} e^{-\frac{x^2}{2}}\,dx$ with 3 quadratic fits:



An example of the Composite Simpsons Rule approximating $\int_{-5}^{5} e^{-\frac{x^2}{2}}\,dx$ with 3 quadratic fits.

The formula for the Composite Simpsons Rule will be presented here without any sort of proof outline as it is a little advanced for the target audience of this User's Guide[2]:

$$\int_a^b f(x)\,dx = \frac{h}{3}\Big(f(x_0)+4\sum_{i=0}^{\lfloor\frac{n}{2}\rfloor-1} f(x_{2i+1})+2\sum_{i=1}^{\lfloor\frac{n}{2}\rfloor-1} f(x_{2i})+f(x_n)\Big)-\frac{h^4}{180}(b-a)f^{(4)}(\mu).$$

The formula above looks confusing, but all it really is is a complex way of writing

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\big(f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+\cdots+2f(x_{n-2})+4f(x_{n-1})+f(x_n)\big).$$

Observe that the remainder term has a 4^{th} derivative in it, which means Composite Simpson's Rule can approximate any cubic polynomial perfectly.

Note that because the Composite Simpson's Rule needs sets of 3 points to make a quadratic fit, the number of function evaluations has to be odd and the number of subintervals has to be even.

[2]The formula is taken from the 9th edition introductory Numerical Analysis textbook by Richard L. Burden and J. Douglas Faires, page 206.

vi. Gaussian-Legendre Quadrature

Gaussian Quadrature works by using sets of unequally spaced nodes in a quadrature formula. It often times happens that these nodes are more optimal than using equally spaced nodes. The simplest and most optimal points appear from using Gaussian-Legendre Quadrature. Here's a basic outline to generate a Gaussian Quadrature formula for $n = 2$.

First, let's look at a the integral of a general cubic polynomial integrated from -1 to 1,

$$\begin{aligned}\int_{-1}^{1} p(x)\,dx &= \int_{-1}^{1} c_0 + c_1 x + c_2 x^2 + c_3 x^3\,dx\\ &= \int_{-1}^{1} c_0\,dx + \int_{-1}^{1} c_1 x\,dx + \int_{-1}^{1} c_2 x^2\,dx + \int_{-1}^{1} c_3 x^3\,dx\\ &= c_0 \int_{-1}^{1} 1\,dx + c_1 \int_{-1}^{1} x\,dx + c_2 \int_{-1}^{1} x^2\,dx + c_3 \int_{-1}^{1} x^3\,dx.\end{aligned}$$

What that tells us is if we can integrate $1, x, x^2, x^3$ perfectly, then we can integrate any cubic polynomial perfectly. If we try to do this integral with the quadrature formula,

$$\int_{-1}^{1} f(x)\,dx = c_0 f(x_0) + c_1 f(x_1).$$

We can guarentee this quadrature formula will integrate any cubic polynomial perfectly by generating four equations using $f(x) = 1, x, x^2, x^3$, like so:

$$\begin{aligned}\int_{-1}^{1} 1\,dx = 2 &= c_0 + c_1,\\ \int_{-1}^{1} x\,dx = 0 &= c_0 x_0 + c_1 x_1,\\ \int_{-1}^{1} x^2\,dx = \frac{2}{3} &= c_0 x_0^2 + c_1 x_1^2,\\ \int_{-1}^{1} x^3\,dx = 0 &= c_0 x_0^3 + c_1 x_1^3.\end{aligned}$$

Observe that we generated 4 equations and 4 unknowns. Therefore, this system can be solved. The algebra will be left to the motivated reader, but the solutions are

$$\begin{aligned} c_0 &= 1, \\ c_1 &= 1, \\ x_0 &= \frac{\sqrt{3}}{3}, \\ x_1 &= -\frac{\sqrt{3}}{3}. \end{aligned}$$

This gives us the final formula

$$\int_{-1}^{1} f(x)\,dx = f(\frac{\sqrt{3}}{3}) + f(-\frac{\sqrt{3}}{3}).$$

.

A good question to ask though is won't this work only for integrals from -1 to 1? The answer to that question is yes, but we can change the limits of integration by a simple linear transformation

$$\begin{aligned} z &= \frac{b-a}{2}x + \frac{b+a}{2}, \\ dz &= \frac{b-a}{2}dx \end{aligned}$$

Then, we get

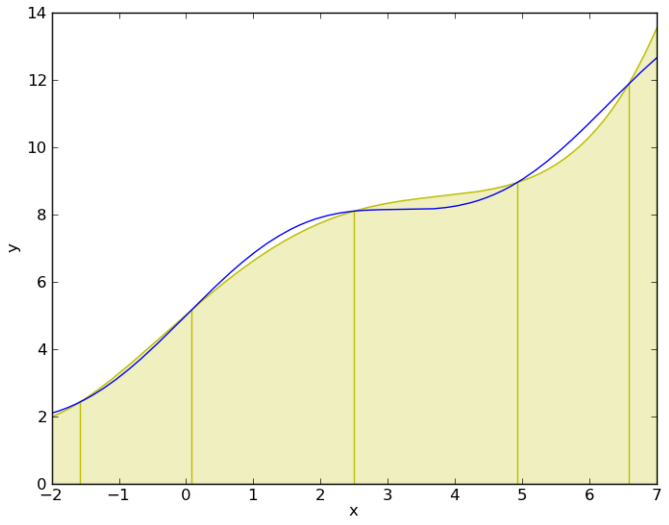
$$\begin{aligned} \int_{b}^{a} f(z)dz &= \int_{-1}^{1} f(\frac{b-a}{2}x + \frac{b+a}{2})\frac{b-a}{2}\,dx \\ &= \frac{b-a}{2}\int_{-1}^{1} f(\frac{b-a}{2}x + \frac{b+a}{2})\,dx \\ &= \frac{b-a}{2}\left(f(\frac{b-a}{2}\cdot\frac{\sqrt{3}}{3} + \frac{b+a}{2}) + f(-\frac{b-a}{2}\cdot\frac{\sqrt{3}}{3} + \frac{b+a}{2})\right) \end{aligned}$$

This definition makes it obvious that we can generate Quadrature formulas using n nodes to integrate a polynomial of degree $2n-1$ perfectly.

We can generate all the nodes and coefficients needed as above, but other's have already done that work, so it's more efficient to just take their lists[3].

[3]The nodes and weights in *Pro Quadrature Viewer ZX790* are taken from `http://www-troja.fjfi.cvut.cz/ klimo/nm/c8/legpoly.pdf`.

There's another way of viewing Gaussian-Legendre quadrature though, which lends itself much better to the pictorial sense. It turns out that Gaussian-Legendre Quadrature is equivalent to integrating a Polynomial fit that has nodes at the roots of the n^{th} Legendre Polynomial[4]. That's a much more abstract definition, but it works well for allowing the user to see what Guassian Legendre actually does. Here's an example:



An example of Gaussian Legendre Quadrature approximating $\int_{-2}^{7} \sin x + x + 5\, dx$ with 5 nodes.

[4]This is proven in the 9th edition introductory Numerical Analysis textbook by Richard L. Burden and J. Douglas Faires, page 231-232, Theorem 4.7.

3. Limitations and Extensions of ***Pro Quadrature Viewer ZX790***

Pro Quadrature Viewer ZX790 is most certainly a useful tool. We are very proud of our work and think that it would be useful to junior level computational science students to see how approximating the area under a curve works, or a way for students in a introductory numerical analysis course to visualize the quadrature methods they're learning about. There are of course some useful things that it cannot do, and important quadrature methods that are not included.

i. Limitations

One limitation of *Pro Quadrature Viewer ZX790* is that it only approximates integrals of 1 dimension. It turns out that approximating 2 dimensional integrals is pretty much just using the formula twice, once over each dimension. This is not something that can be done by *Pro Quadrature Viewer ZX790* though. It should be noted that the number of function evaluations grows exponentially with the dimension of the integral for the quadrature methods. This is why that after a certain point, Monte Carlo Integration becomes more efficient compared to the quadrature methods. Any introductory book on numerical analysis will go over it in more detail for the curious reader.

A bigger limitation though is that we don't have an option for the user to input a tolerance and have *Pro Quadrature Viewer ZX790* go until that tolerance is met and display the n that was needed. The reason we don't have this is because our code works by using NUMPY's array arithmetic. This is a much faster way of doing the quadrature methods than using an explicit for-loop. If we wanted to allow the user to enter a tolerance, that would require an explicit for-loop, which would be very slow. There is something called Adaptive Quadrature though that will approximate the integral to a certain tolerance and still uses array arithmetic. This wasn't included because it creates in own n value based on relative errors and it would make it hard to compare to the other methods.

ii. Extensions

This section will comprise of a list of popular quadrature methods, a very brief explanation, and a link to a better explanation:

Clenshaw-Curtis quadrature works by creating a discrete cosine transform and integrating term-by-term.

A python implementation:

`http://www.scientificpython.net/1/post/2012/04/clenshaw-curtis-quadrature.html`

Adaptive Simpsons Rule works by choosing the nodes selectively based on how much of the area under the curve it takes up.

The quadrature section at this link describes the theory and a MATLAB implementation:

`http://www.mathworks.com/moler/chapters.html`

Other types of Gaussian Quadrature are better suited to specific types of functions or integrals. For example, Gaussian-Laguerre Quadrature for improper integrals:

Wikipedia shows many types of Gaussian Quadrature methods with their nodes and weights:

`http://en.wikipedia.org/wiki/Gaussian_quadrature`