A Family of Correctors

Jayson Saumer
Chad Scott, Ph.D.
Department of Mathematics and Computer Science

Abstract

For the differential equation y' = f(x,y), we find a family of correctors derived from 3-point quadrature. Further, we isolate the effectiveness of these correctors and pinpoint in one specific case which member of the family is the best corrector.

1. Introduction

In the world of differential equations, there are many occasions where an exact solution for y cannot be obtained. In this case, mathematicians and scientists use what are called numerical methods to get an approximation to the differential equation y. The inspiration of this paper was the paper listed in [1], and this paper was to follow up that paper by extending their results. One thing that we found lacking in the paper was the detail about corrector methods for their quadrature methods. After a lot of investigation of three point quadrature, we obtained the following:

Main Theorem:

For A,B,C satisfying:

$$A = t, B = 2 - 2t, C = t, y_j - y_{j-2} = h\left(Af_{j-2} + Bf_{j-1} + Cf_j\right)$$

is a one parameter family of corrector formulas on $[0,1]$ that include the quadrature correctors arising from rectangular, trapezoidal, and Simpson's quadrature methods.

Further, several programs were written to investigate the family and the best corrector was found to be approximately $t = 0.3339155$ in the case of $y = y'$.

2. Definitions

Def. Differential equation – An equation that contains an unknown function and some of its derivatives. [6 p.614]

Def. Mesh point - The points that are generated by approximation methods for differential equations. [8 p.514]

Def. One-step methods - Methods using the approximation at one previous mesh point to determine the approximation at the next point. [3 p.237]

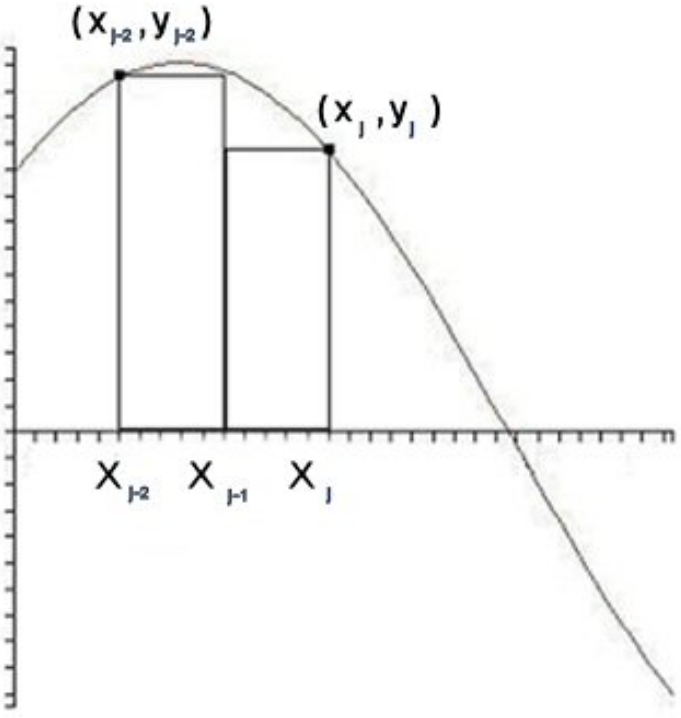
Def. Multi-step methods - Methods using the approximation at more than one previous mesh point to determine the approximation at the next point.
[3 p.237]

Def. Implicit methods - An implicit method is one for which there is not an explicit formula at a point for the value of the unknown terms appearing in the differential equation. Generally a nonlinear algebraic equation must be solved to determine the value at a given point. [8 p.514]

Def. Explicit methods - An explicit method is one for which there is an explicit formula at a point for the value of the unknown terms appearing in the differential equation [8 p.514]

Def. Predictor-corrector methods - The combination of an explicit and implicit technique to estimate the differential equation. [3 p.244]

3. A Corrector Arising from Rectangular 3-point Quadrature



Here is an example of using the rectangular method of estimating area under an integral curve for 3-point quadrature. By the Fundamental Theorem of Calculus [5 pp. 354-358] and $f(x,y) = y$,

$$y_j - y_{j-2} = \int_{x_{j-2}}^{x_j} f(x,y)dx$$

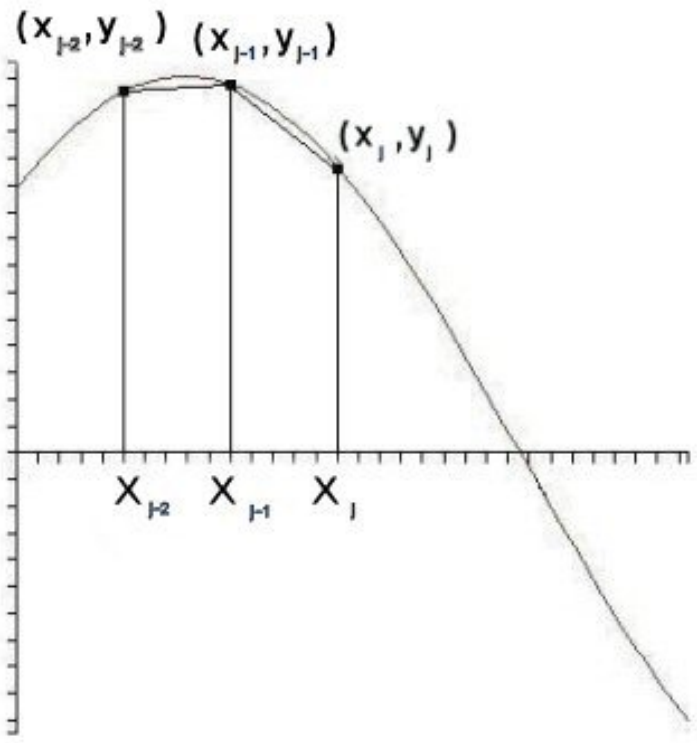
$$\textit{Where } y_j = y(x_j)$$

So then,

$$\begin{aligned} y_j - y_{j-2} &\approx hy_j + hy_{j-2} \\ &= hf_j + hf_{j-2} \\ &= h\left(f_j + f_{j-2}\right) \\ &= h\left(1f_{j-2} + 0f_{j-1} + 1f_j\right) \end{aligned}$$

So then, $A + B + C = 2$

4. A Corrector Arising from Trapezoidal 3-point Quadrature



Now we are going to look at an example of the use of the trapezoidal method of estimating the area under the integral curve for 3-point quadrature.

We know that

$$\begin{aligned} y_j - y_{j-2} &\approx h\left(\frac{y_{j-2}+y_{j-1}}{2}\right)+h\left(\frac{y_{j-1}+y_j}{2}\right) \\ &= h\left(\frac{f_{j-2}+f_{j-1}}{2}\right)+h\left(\frac{f_{j-1}+f_j}{2}\right) \\ &= h\left(\frac{f_{j-2}+f_{j-1}}{2}+\frac{f_{j-1}+f_j}{2}\right) \\ &= h\left(\frac{1}{2}f_{j-2}+1f_{j-1}+\frac{1}{2}f_j\right) \end{aligned}$$

So then, $A+B+C=2$ just as in the rectangular case.

This draws to a general two-parameter corrector formula based on 3-point quadrature.

$$y_j - y_{j-2} = h\left(Af_{j-2}+Bf_{j-1}+Cf_j\right)$$
$$With\ \ A+B+C=2$$

To narrow choices to one parameter, we will make the assumption that when the step size h is small; the solution behaves much like a quadratic on $\left[x_{j-2}, x_j\right]$.

5. A One Parameter Family of Corrector Formulas

Starting with a first order differential equation $y' = f(x,y)$ with a solution y, we will approximate the solution y with $y = ax^2 + bx + c$. Then we let A, B, and C be coefficients such that

$$\begin{aligned} y_j - y_{j-2} &= h\left(Af_{j-2}+Bf_{j-1}+Cf_j\right) \\ &= h\left(Ay'_{j-2}+By'_{j-1}+Cy'_j\right) \end{aligned}$$

Then on $[0,1]$, $x_j = j * h$ and $y' = 2ax + b$. When this formula is used, the left hand side of the equation becomes

$$\begin{aligned} y_j - y_{j-2} &= a(jh)^2 + b(jh) + c - \left(a((j-2)h)^2 + b((j-1)h) + c\right) \\ &= a(jh)^2 + b(jh) + c - \left(a(jh)^2 - 4ajh^2 + 4ah^2 + bjh - 2bh + c\right) \\ &= a(jh)^2 + b(jh) + c - a(jh)^2 + 4ajh^2 - 4ah^2 - bjh + 2bh - c \\ &= (4aj - 4a)h^2 + 2bh \end{aligned}$$

And the right side becomes

$$\begin{aligned} h\left(Ay'_{j-2} + By'_{j-1} + Cy'_j\right) &= h(A(2a(j-2)h + b) + B(2a(j-1)h + b) + C(2ajh + b)) \\ &= h(A(2ajh - 4ah) + Ab + B(2ajh - 2ah) + Bb + C2ajh + Cb) \\ &= A(2aj - 4a)h^2 + Abh + B(2aj - 2a)h^2 + Bbh + C2ajh^2 + Cbh \\ &= (A(2aj - 4a) + B(2aj - 2a) + C2aj)h^2 + (A + B + C)bh \end{aligned}$$

Equating the left and the right sides we obtain

$$A + B + C = 2$$

And

$$\begin{aligned} 4aj - 4a &= A(2aj - 4a) + B(2aj - 2a) + C2aj \\ 4aj - 4a &= A2aj - A4a + B2aj - B2a + C2aj \\ 4j - 4 &= A2j - A4 + B2j - B2 + C2j \end{aligned}$$

Solving this equation, we obtain

$$\begin{aligned} 4j &= A2j + B2j + C2j \\ 2 &= A + B + C \end{aligned}$$

And

$$\begin{aligned} -4 &= -A4 - B2 \\ 4 &= 4A + 2B \\ 2 &= 2A + B \end{aligned}$$

So we obtain the solution

$$A + B + C = 2$$
$$2A + B = 2$$
$$A = C$$

Also solving this equation parametrically, we obtain

$$A = t$$
$$B = 2 - 2t$$
$$C = t$$

Applying the same analysis to at Simpson's 3-point quadrature [5 p.378] we see that it fits the mold of the above one-parameter family of corrector formulas. Putting together this and the results from §2 and §3, we now have with $A = t, B = 2 - 2t,$ and $C = t,$ the result mentioned in the introduction. Specifically,

Main Theorem:

For A,B,C satisfying: $A = t, B = 2 - 2t, C = t, y_j - y_{j-2} = h\left(Af_{j-2} + Bf_{j-1} + Cf_j\right)$ is a one parameter family of corrector formulas on $[0,1]$ that include the quadrature correctors arising from rectangular, trapezoidal, and Simpson's quadrature methods.

6. A Case Study $(y = y')$, Runge-Kutta with h = 1/4

Let's apply our corrector to a predictor; namely the Runge-Kutta [2 pp.435-439] with parameters: $y = y', y(0) = 1, h = 1/4$ we obtain the following mesh points:

$$y_0 = 1$$
$$y_1 = 1.28401692708333$$
$$y_2 = 1.64869946903653$$
$$y_3 = 2.1169580259162$$
$$y_4 = 2.71820993920132$$

Now we take our 3-point quadrature correctors, namely the rectangular, trapezoidal, and Simpson's methods to the mesh points from the Runge-Kutta. We will also analyze the relative error, $\varepsilon_r = \dfrac{|approximated - actual|}{actual}$ from the results of the correctors and the results from the Runge-Kutta.

Table 1, Runge-Kutta

Nodes	**Runge -Kutta**	ε_r
y_0	1.0	0
y_1	1.28401692708333	$6.612018647 * 10^{-6}$
y_2	1.64869946903653	$1.322358144 * 10^{-5}$
y_3	2.1169580259162	$1.983514391 * 10^{-5}$
y_4	2.71820993920132	$2.644648515 * 10^{-5}$

Table 2, Rectangular - One Iteration

Nodes	**Rectangular**	ε_r
y_2	1.662174867	$8.160018456 * 10^{-3}$
y_3	2.134260666	$8.153353265 * 10^{-3}$
y_4	2.740426821	$8.146687651 * 10^{-3}$

Table 3, Trapezoidal - One Iteration

Nodes	**Trapezoidal**	ε_r
y_2	1.652091666	$2.044247902 * 10^{-3}$
y_3	2.121313663	$2.037622090 * 10^{-3}$
y_4	2.723802652	$2.030997648 * 10^{-3}$

Table 4, Simpson's - One Iteration

Nodes	**Simpson's**	ε_r
y_2	1.648730597	$5.656504931 * 10^{-6}$
y_3	2.116997997	$9.541804364 * 10^{-7}$
y_4	2.718261261	$7.566176468 * 10^{-6}$

As we can see, only Simpson's resulted in a true correction after one "corrective" iteration. In the next sections, we seek more satisfying results by looking at 2 and 3 iterations.

7. A Case Study $(y = y')$, Two and Three Iterations, Runge-Kutta with h = 1/4

In this section, we further our investigation of rectangular, trapezoidal, and Simpson's correctors by performing the correctors on a second and third iteration.

Table 5, Rectangular – 1, 2, and 3 Iterations

Nodes	ε_r - First Iteration	ε_r - Second Iteration	ε_r - Third Iteration
y_2	$8.160018456 * 10^{-3}$	$1.020332927 * 10^{-2}$	$1.071415667 * 10^{-2}$
y_3	$8.153353265 * 10^{-3}$	$1.019665037 * 10^{-2}$	$1.070747417 * 10^{-2}$
y_4	$8.146687651 * 10^{-3}$	$1.018997100 * 10^{-2}$	$1.070079184 * 10^{-2}$

Table 6, Trapezoidal – 1, 2, and 3 Iterations

Nodes	ε_r - First Iteration	ε_r - Second Iteration	ε_r - Third Iteration
y_2	$2.044247902 * 10^{-3}$	$2.301431459 * 10^{-3}$	$2.333579403 * 10^{-3}$
y_3	$2.037622090 * 10^{-3}$	$2.294804422 * 10^{-3}$	$2.326952272 * 10^{-3}$
y_4	$2.030997648 * 10^{-3}$	$2.288178487 * 10^{-3}$	$2.320326000 * 10^{-3}$

Table 7, Simpson's – 1, 2, and 3 Iterations

Nodes	ε_r - First Iteration	ε_r - Second Iteration	ε_r - Third Iteration
y_2	$5.656504931 * 10^{-6}$	$7.229845462 * 10^{-6}$	$7.361462616 * 10^{-6}$
y_3	$9.541804364 * 10^{-7}$	$6.192725505 * 10^{-7}$	$7.501180856 * 10^{-7}$
y_4	$7.566176468 * 10^{-6}$	$5.992756098 * 10^{-6}$	$5.861791017 * 10^{-6}$

We can see from the following tables that the only method that truly "corrects" the previous result is the Simpson's method.

8. A Case Study $(y = y')$, Runge-Kutta with h = 1/10

Let's apply our corrector to a predictor again; namely the Runge-Kutta with parameters: $y = y', y(0) = 1, h = 1/10$. This time we obtain the following mesh points:

$$y_0 = 1$$
$$y_1 = 1.10517083333333$$
$$y_2 = 1.22140257085069$$
$$y_3 = 1.34985849706254$$
$$y_4 = 1.49182424008069$$
$$y_5 = 1.64872063859684$$
$$y_6 = 1.82211796209193$$
$$y_7 = 2.01375162659678$$
$$y_8 = 2.22553956329232$$
$$y_9 = 2.45960141378007$$
$$y_{10} = 2.71827974413517$$

Again, we take our 3-point quadrature correctors, namely the rectangular, trapezoidal, and Simpson's methods to the mesh points from the Runge-Kutta. We will also analyze the relative error, $\varepsilon_r = \dfrac{|approximated - actual|}{actual}$ from the results of the correctors and the results from the Runge-Kutta.

Table 8, Runge-Kutta

Nodes	Runge -Kutta	ε_r
y_0	1.0	0
y_1	1.10517083333333	$7.691118054 * 10^{-8}$
y_2	1.22140257085069	$1.531026508 * 10^{-7}$
y_3	1.34985849706254	$2.303944666 * 10^{-7}$
y_4	1.49182424008069	$3.070065810 * 10^{-7}$
y_5	1.64872063859684	$3.833273769 * 10^{-7}$
y_6	1.82211796209193	$4.599041511 * 10^{-7}$
y_7	2.01375162659678	$5.363121282 * 10^{-7}$
y_8	2.22553956329232	$6.133340362 * 10^{-7}$
y_9	2.45960141378007	$6.899487126 * 10^{-7}$
y_{10}	2.71827974413517	$7.666607555 * 10^{-7}$

Table 9, Rectangular - One Iteration

Nodes	Rectangular	ε_r
y_2	1.222140257	$6.038131117 * 10^{-4}$
y_3	1.350673766	$6.037357353 * 10^{-4}$
y_4	1.492725252	$6.036593986 * 10^{-4}$
y_5	1.649716411	$6.035829206 * 10^{-4}$
y_6	1.823218460	$6.035062039 * 10^{-4}$
y_7	2.014967866	$6.034301013 * 10^{-4}$
y_8	2.226883714	$6.033526425 * 10^{-4}$
y_9	2.461086931	$6.032761926 * 10^{-4}$
y_{10}	2.719921494	$6.031994119 * 10^{-4}$

Table 10, Trapezoidal - One Iteration

Nodes	Trapezoidal	ε_r
y_2	1.221587212	$1.510181623 * 10^{-4}$
y_3	1.350062557	$1.509409716 * 10^{-4}$
y_4	1.492049761	$1.508642405 * 10^{-4}$
y_5	1.648969878	$1.507877677 * 10^{-4}$
y_6	1.822393414	$1.507113587 * 10^{-4}$
y_7	2.014056049	$1.506351793 * 10^{-4}$
y_8	2.225876001	$1.505580040 * 10^{-4}$
y_9	2.459973235	$1.504811888 * 10^{-4}$
y_{10}	2.718690670	$1.504045665 * 10^{-4}$

Table 11, Simpson's - One Iteration

Nodes	Simpson's	ε_r
y_2	1.221402863	$8.596672908 * 10^{-8}$
y_3	1.349858820	$8.889818645 * 10^{-9}$
y_4	1.491824598	$6.703200459 * 10^{-8}$
y_5	1.648721034	$1.437477663 * 10^{-7}$
y_6	1.822118399	$2.200734661 * 10^{-7}$
y_7	2.013752109	$2.969580117 * 10^{-7}$
y_8	2.225540096	$3.738416982 * 10^{-7}$
y_9	2.459602003	$4.504791830 * 10^{-7}$
y_{10}	2.718280395	$5.271712393 * 10^{-7}$

It is clear that only Simpson's resulted in a true correction after one "corrective" iteration. Let's look at these methods in two and three iterations.

9. A Case Study $(y = y')$, Two and Three Iterations, Runge-Kutta with h = 1/10

In this section, we further our investigation of Rectangular, Trapezoidal, and Simpson's correctors is furthered by the correctors being performed on a second and third iteration.

Table 12, Rectangular – 1, 2, and 3 Iterations

Nodes	ε_r - First Iteration	ε_r - Second Iteration	ε_r - Third Iteration
y_2	$6.038131117 * 10^{-4}$	$6.642100607 * 10^{-4}$	$6.702498374 * 10^{-4}$
y_3	$6.037357353 * 10^{-4}$	$6.641324224 * 10^{-4}$	$6.701715725 * 10^{-4}$
y_4	$6.036593986 * 10^{-4}$	$6.640559050 * 10^{-4}$	$6.700954887 * 10^{-4}$
y_5	$6.035829206 * 10^{-4}$	$6.639794241 * 10^{-4}$	$6.700192564 * 10^{-4}$
y_6	$6.035062039 * 10^{-4}$	$6.639029244 * 10^{-4}$	$6.699425965 * 10^{-4}$
y_7	$6.034301013 * 10^{-4}$	$6.638267923 * 10^{-4}$	$6.698662628 * 10^{-4}$
y_8	$6.033526425 * 10^{-4}$	$6.637491953 * 10^{-4}$	$6.697890752 * 10^{-4}$
y_9	$6.032761926 * 10^{-4}$	$6.636729287 * 10^{-4}$	$6.697125209 * 10^{-4}$
y_{10}	$6.031994119 * 10^{-4}$	$6.635956513 * 10^{-4}$	$6.696354959 * 10^{-4}$

Table 13, Trapezoidal – 1, 2, and 3 Iterations

Nodes	ε_r - First Iteration	ε_r - Second Iteration	ε_r - Third Iteration
y_2	$1.510181623 * 10^{-4}$	$1.585766847 * 10^{-4}$	$1.589541195 * 10^{-4}$
y_3	$1.509409716 * 10^{-4}$	$1.584995399 * 10^{-4}$	$1.588773572 * 10^{-4}$
y_4	$1.508642405 * 10^{-4}$	$1.584234396 * 10^{-4}$	$1.588015001 * 10^{-4}$
y_5	$1.507877677 * 10^{-4}$	$1.583463528 * 10^{-4}$	$1.587242214 * 10^{-4}$
y_6	$1.507113587 * 10^{-4}$	$1.582701413 * 10^{-4}$	$1.586477237 * 10^{-4}$
y_7	$1.506351793 * 10^{-4}$	$1.581932076 * 10^{-4}$	$1.585711090 * 10^{-4}$
y_8	$1.505580040 * 10^{-4}$	$1.581166159 * 10^{-4}$	$1.584945105 * 10^{-4}$
y_9	$1.504811888 * 10^{-4}$	$1.580397253 * 10^{-4}$	$1.584174285 * 10^{-4}$
y_{10}	$1.504045665 * 10^{-4}$	$1.579626496 * 10^{-4}$	$1.583408297 * 10^{-4}$

Table 14, Simpson's – 1, 2, and 3 Iterations

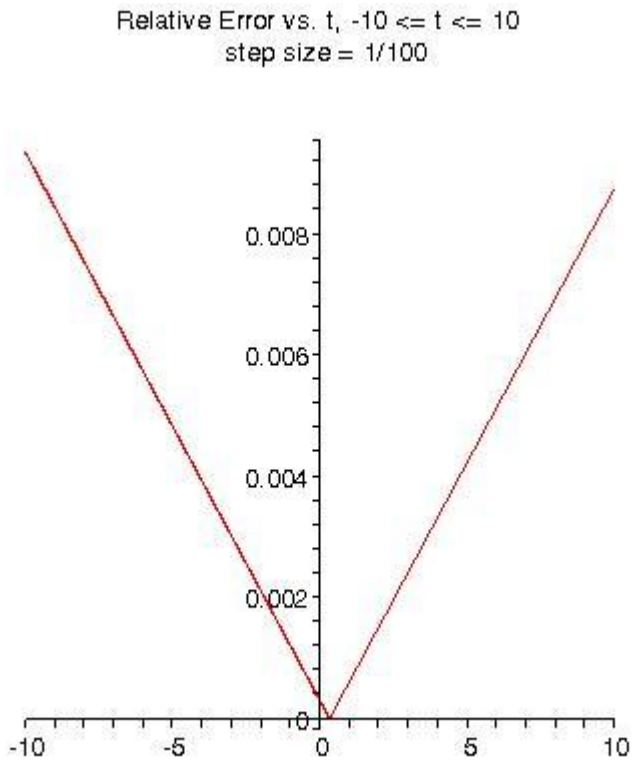
Nodes	ε_r - First Iteration	ε_r - Second Iteration	ε_r - Third Iteration
y_2	$8.596672908 * 10^{-8}$	$9.415403662 * 10^{-8}$	$9.415403662 * 10^{-8}$
y_3	$8.889818645 * 10^{-9}$	$1.703881907 * 10^{-8}$	$1.703881907 * 10^{-8}$
y_4	$6.703200459 * 10^{-8}$	$5.898816404 * 10^{-8}$	$5.898816404 * 10^{-8}$
y_5	$1.437477663 * 10^{-7}$	$1.358628678 * 10^{-7}$	$1.358628678 * 10^{-7}$
y_6	$2.200734661 * 10^{-7}$	$2.118412916 * 10^{-7}$	$2.118412916 * 10^{-7}$
y_7	$2.969580117 * 10^{-7}$	$2.885160616 * 10^{-7}$	$2.880194763 * 10^{-7}$
y_8	$3.738416982 * 10^{-7}$	$3.657537769 * 10^{-7}$	$3.657537769 * 10^{-7}$
y_9	$4.504791830 * 10^{-7}$	$4.427543595 * 10^{-7}$	$4.423477898 * 10^{-7}$
y_{10}	$5.271712393 * 10^{-7}$	$5.194457710 * 10^{-7}$	$5.190778916 * 10^{-7}$

We can see again from the following tables that the only method that truly "corrects" the previous result is the Simpson's method. More specifically, the Simpson's method only "corrected" itself on nodes y_4 through y_{10}.

Looking closely at rectangular, trapezoidal, and Simpson's "correctors," we see that they are actually special cases of the main theorem when $t = 1, t = 1/2, t = 1/3,$ respectively. With that in mind, we seek the best possible parameter *t* in the next section.

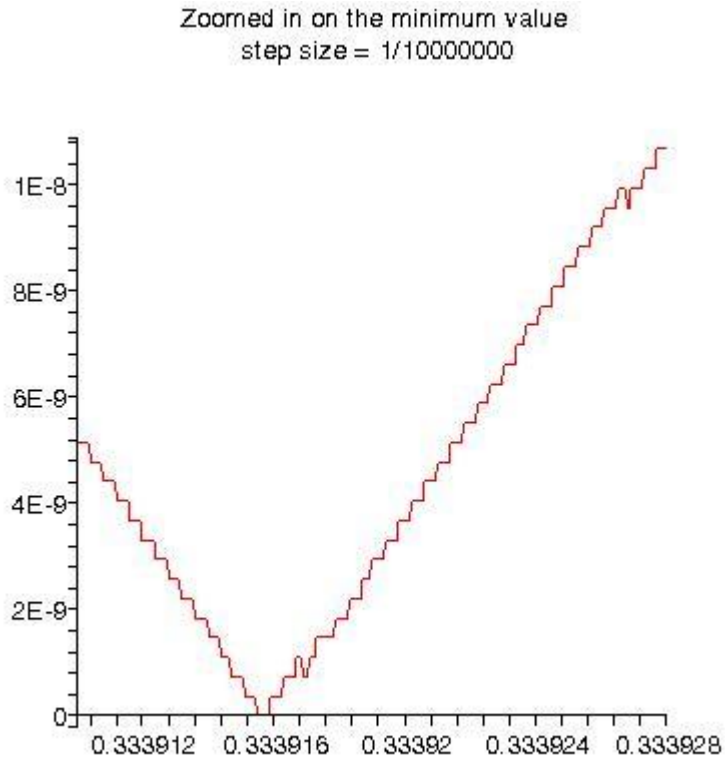
10. The Best Member of the Family

To try to find the minimal relative error with a corrector, we will be looking at a graph of relative error vs. *t*, the parameter from the main theorem with $-10 \le t \le 10$. For the graph, we will be using the mesh points from the Runge-Kutta with step size $h = 1/10$ and we will be evaluating the relative error at y(1).



As we can see on the graph, the relative error appears to be minimal near 1/3, which would be Simpson's method. Let's take a closer look to get a more accurate estimate of that minimizer.

In the zoomed view, we look at the relative error from .333912 to .333928.



On this graph we can see that there are a few points on the bottom of the graph where we could select a value of t for a new corrector to test. In the next section, I will select a value of *t* to test in our case study and compare it with Rectangular, Trapezoidal, and Simpson's.

11. A Case Study $(y = y')$, Testing a New value of *t*, Runge-Kutta with h = 1/4 and h = 1/10

For the value of *t* that we are going to use, we are going to choose the value of $t = 0.3339155$ and use it to correct $y(1)$ with the nodes from both Runge-Kutta with $h = 1/4$ and $h = 1/10$. We will then compare the correction to the results from Simpson's since Simpson's was the best corrector of the methods that we tested.

For this calculation, we have to increase the precision of the value of e^1 to 20 digits to be able to get an error other than 0 on the test value of t for the RK_{10}

Table 15, Test value of *t*, $y(1)$

Nodes	ε_r - Test value of t, K_4	ε_r - Test value of t, RK_{10}
y_2	$4.451341239 * 10^{-7}$	$* \ 1.692245429 3 * 10^{-10}$
Nodes	ε_r - Simpson's, RK_4	ε_r - Simpson's, RK_{10}
y_2	$7.566176468 * 10^{-6}$	$5.271712393 * 10^{-7}$

We can see that our new value of t improves the result to a far greater precision than Simpson's method after one correction.

12. Future Research

It would be beneficial to look at other differential equations to see which corrector in the family is better. We speculate that the best corrector lies consistently around 1/3 (Simpson's) thus making Simpson's the best general choice.

13. Appendix

a. The following code was used in Maple 9 to produce the values in §6 and §7, similar code was used to create the results in §8 and §9.

```
y0 := 1.0;
y1 := 1.28401692708333;
y2 := 1.64869946903653;
y3 := 2.1169580259162;
y4 := 2.71820993920132;
```

```
rect1_2 := evalf((1/4)*(y2 + y0) + y0);
rect2_2 := evalf((1/4)*(rect1_2 + y0) + y0);
rect3_2 := evalf((1/4)*(rect2_2 + y0) + y0);
re_rect1_2 := evalf(abs(rect1_2 - exp(.5))/exp(.5));
re_rect2_2 := evalf(abs(rect2_2 - exp(.5))/exp(.5));
re_rect3_2 := evalf(abs(rect3_2 - exp(.5))/exp(.5));

rect1_3 := evalf((1/4)*(y3 + y1) + y1);
rect2_3 := evalf((1/4)*(rect1_3 + y1) + y1);
rect3_3 := evalf((1/4)*(rect2_3 + y1) + y1);
re_rect1_3 := evalf(abs(rect1_3 - exp(.75))/exp(.75));
re_rect2_3 := evalf(abs(rect2_3 - exp(.75))/exp(.75));
re_rect3_3 := evalf(abs(rect3_3 - exp(.75))/exp(.75));

rect1_4 := evalf((1/4)*(y4 + y2) + y2);
rect2_4 := evalf((1/4)*(rect1_4 + y2) + y2);
rect3_4 := evalf((1/4)*(rect2_4 + y2) + y2);
re_rect1_4 := evalf(abs(rect1_4 - exp(1))/exp(1));
re_rect2_4 := evalf(abs(rect2_4 - exp(1))/exp(1));
re_rect3_4 := evalf(abs(rect3_4 - exp(1))/exp(1));

trap1_2 := evalf((1/4)*((1/2)*y0 + y1 + (1/2)*y2) + y0);
trap2_2 := evalf((1/4)*((1/2)*y0 + y1 + (1/2)*trap1_2) +
y0);
trap3_2 := evalf((1/4)*((1/2)*y0 + y1 + (1/2)*trap2_2) +
y0);
re_trap1_2 := evalf(abs(trap1_2 - exp(.5))/exp(.5));
re_trap2_2 := evalf(abs(trap2_2 - exp(.5))/exp(.5));
re_trap3_2 := evalf(abs(trap3_2 - exp(.5))/exp(.5));

trap1_3 := evalf((1/4)*((1/2)*y1 + y2 + (1/2)*y3) + y1);
trap2_3 := evalf((1/4)*((1/2)*y1 + y2 + (1/2)*trap1_3) +
y1);
trap3_3 := evalf((1/4)*((1/2)*y1 + y2 + (1/2)*trap2_3) +
y1);
re_trap1_3 := evalf(abs(trap1_3 - exp(.75))/exp(.75));
re_trap2_3 := evalf(abs(trap2_3 - exp(.75))/exp(.75));
re_trap3_3 := evalf(abs(trap3_3 - exp(.75))/exp(.75));

trap1_4 := evalf((1/4)*((1/2)*y2 + y3 + (1/2)*y4) + y2);
trap2_4 := evalf((1/4)*((1/2)*y2 + y3 + (1/2)*trap1_4) +
y2);
```

```
trap3_4 := evalf((1/4)*((1/2)*y2 + y3 + (1/2)*trap2_4) +
y2);
re_trap1_4 := evalf(abs(trap1_4 - exp(1))/exp(1));
re_trap2_4 := evalf(abs(trap2_4 - exp(1))/exp(1));
re_trap3_4 := evalf(abs(trap3_4 - exp(1))/exp(1));

simp1_2 := evalf((1/4)*((1/3)*y0 + (4/3)*y1 + (1/3)*y2) +
y0);
simp2_2 := evalf((1/4)*((1/3)*y0 + (4/3)*y1 +
(1/3)*simp1_2) + y0);
simp3_2 := evalf((1/4)*((1/3)*y0 + (4/3)*y1 +
(1/3)*simp2_2) + y0);
re_simp1_2 := evalf(abs(simp1_2 - exp(.5))/exp(.5));
re_simp2_2 := evalf(abs(simp2_2 - exp(.5))/exp(.5));
re_simp3_2 := evalf(abs(simp3_2 - exp(.5))/exp(.5));

simp1_3 := evalf((1/4)*((1/3)*y1 + (4/3)*y2 + (1/3)*y3) +
y1);
simp2_3 := evalf((1/4)*((1/3)*y1 + (4/3)*y2 +
(1/3)*simp1_3) + y1);
simp3_3 := evalf((1/4)*((1/3)*y1 + (4/3)*y2 +
(1/3)*simp2_3) + y1);
re_simp1_3 := evalf(abs(simp1_3 - exp(.75))/exp(.75));
re_simp2_3 := evalf(abs(simp2_3 - exp(.75))/exp(.75));
re_simp3_3 := evalf(abs(simp3_3 - exp(.75))/exp(.75));

simp1_4 := evalf((1/4)*((1/3)*y2 + (4/3)*y3 + (1/3)*y4) +
y2);
simp2_4 := evalf((1/4)*((1/3)*y2 + (4/3)*y3 +
(1/3)*simp1_4) + y2);
simp3_4 := evalf((1/4)*((1/3)*y2 + (4/3)*y3 +
(1/3)*simp2_4) + y2);
re_simp1_4 := evalf(abs(simp1_4 - exp(1))/exp(1));
re_simp2_4 := evalf(abs(simp2_4 - exp(1))/exp(1));
re_simp3_4 := evalf(abs(simp3_4 - exp(1))/exp(1));
```

b. The following code was used in Maple 9 to produce the graph from $-10 \le t \le 10$ in §10

```
y8 := 2.22553956329232;
y9 := 2.45960141378007;
y10 := 2.71827974413517;

for i from -1000 to 1000 do
a [ i ] := evalf((1/10)*(y8*(i/100) + y9*(2 - 2*(i/100)) +
y10*(i/100)) + y8);
end do;

for i from -1000 to 1000 do
b [ i ] := evalf(abs(a [ i ] - exp(1))/exp(1));
end do;

l := [[n/100, b[ n ] ] $n=-1000..1000];
plot(l, x=-10..10, style=line, title="Relative Error vs. t, -10
<= t <= 10");
```

c. The following code was used in Maple 9 to produce the zoomed in graph in §10

```
eqns := {u+v+w=2, 2*v + 4*w = 4};
soln := solve(eqns);

f8:=2.22553956239232;
f9:=2.45960141378007;
f10:=2.71827974413517;
y10:= evalf((f8*u) + (f9*v) + (f10*w))*(1/10) + f8;

for i from 3339100 to 3339280 do
re[ i ] :=
evalf(abs((subs(u=i/10000000,subs(subs(u=i/10000000,sol
n), y10)) - exp(1))/exp(1)));
end do;

l := [[n/10000000, re[ n ] ] $n=3339100..3339280];
plot(l, x=3339100/10000000..3339280/10000000,
style=line, title="Zoomed in on the minimum value");
```

14. Acknowledgements

I would like to thank Dr. Scott for all of his help and support throughout this project, and I also would to thank the UW-Superior McNair Program for giving me this opportunity to write this paper and to perform my own research.

15. References

[1] Hathaway, Jack. *To (3, 4, 5,…, ∞) and Beyond*. (2003)

[2] Boyce, William E., DiPrima, Richard C. "Numerical Methods" *Elementary Differential Equations 7th ed*. New York: John Wiley & Sons Inc., 2000. Pp.419-458

[3] Burden, Richard L., Faires, J. Douglas. "Initial-Value Problems for Ordinary Differential Equations" *Numerical Methods third ed*. Boston: PWS Publishers, 1985. Pp.199-289

[4] Wolfram Research. Mathworld. [Online] 20 June 2004. <http://mathworld.wolfram.com>

[5] Finney, Ross L., Weir, Maurice D., Giorando, Frank R., *Thomas's Calculus 10th ed*. Boston: Addison Wesley Longman, 2001

[6] Stewart, James, *Calculus Fourth Edition* Boston: Brooks/Cole Publishing, 1999

[7] Gear, C. William, *Numerical Initial Value Problems in Ordinary Differential Equations* New Jersey, 1971

[8] Zwillinger, Daniel, *Handbook of Differential Equations* Boston: Acadmenic Press Inc., 1989

[9] Deitel H.M., Deitel P. J., *C++ How to Program 3rd ed*. New Jersey: Prentice Hall, 2001

[10] Lafore, Robert, *Object-Oriented Programming in C++ 4th ed*. Indianapolis: SAMS Publishing, 2002

[11] Ujević, Nenad, Roberts, A. J., *A corrected quadrature formula and applications* (2004)