Analysis on Methods: Numerical Integration and ODE

Seth Shelnutt

December 11, 2012

Abstract

Numerical Analysis, it is defined as “The branch of mathematics that deals with the development and use of numerical methods for solving problems”. As we have advanced into the modern age engineering problems are become increasing complex. With this a turn to computer simulation to aid in development and problem solving. In these cases two topics of the numerical analysis field are most often sought and used. Numerical integration and numerical solutions to ordinary differential equations are two topics which come up often in simulating engineering systems. Various methods of these topics will be examined and tested for accuracy and speed.

Contents

1 Introduction

When implementing the field of numerical analysis there are several factors to first consider. Most of the factors are inherited from what type of problem is being analyzed and solved. In this case study numerical integration and ordinary differential equations is the target. As in the last paper on numerical analysis, the chosen language for this work is Python. Python lies in a middle ground between straight C and using matlab, which are the two common techniques. Python offers the robust mathematical support of matlab through numeric python (numpy), scientific python (scipy) and symbolic python (sympy). With the extension of Matplotlib, python also offers a diverse graphical package. Python is a higher level language than C, and offers many features such as garbage collecting and memory allocation. Code is run inside an interpreter which means it can be run on any platform python is support. By the open source nature of python, nearly any existing platform, be it mainstream or embedded is supported. Python also offers various methods of optimization, be it in the code, or the interpreter. Python also offers support for in-line C code if needed. Overall python's rugged and robust offerings make it the ideal language for numerical analysis and thus will be used in this paper.

2 Numerical Integration

For the purpose of this section the object is to find the integral of $\int\limits_0^4 f(x)dx$**, where** $f(x) = e^{3x}$**and** $f(x) = 1 + \sin(10\pi x)$**. For each of the function the step sizes of .1, 0.05, 0.01 and 0.005 will be examined.**

2.1 Riemann Sums

Riemann sums was developed by the German mathematician Bernhard Riemann. This is one of the simplest methods for numerical integration. The method works by dividing up the function into steps of size h. Then rectangle boxes are formed and the area of $h * y_n$**is summed from** $x_0 < x < x_n$**. This yield an approximation for the area under the curve. The general formula is** $\int\limits_a^b f(x)dx = \sum\limits_{i=a}^{b} h * f(x_i)$ **.**

For each of the function the step sizes of .1, 0.05, 0.01 and 0.005 will be examined. Starting with $\int\limits_0^4 f(x)dx = \int\limits_0^4 e^{3x}dx$**, Riemann sums yields an area under the curve of 46519.85, 50284.1, 53441.5 and lastly 53845.4. This is approaching the actual value of 54251.3. The error at h=0.005 is 405.9. The time it took to calculate at h=0.005 was 0.0051 seconds.**

With the second function of $\int\limits_0^4 f(x)dx = \int\limits_0^4 (1+\sin(10\pi x))dx$ at h=0.1 the area yielded is 4.0, and it took 0.0004 seconds to calculate. Curiously at h=0.05, h=0.01, and h=0.005 the area yielded is also 4.0. The time for each increases from 0.00077 seconds to 0.0065 seconds. The actual value of the integral is 4.0. It appears for this case the Riemann sums is exact.

2.2 Trapezoid Rule

The trapezoid rule is a member of Newton–Cotes functions and was co-developed by Newton and Cotes. The method works by dividing up the function into steps of size h. Then by taking two points at $y_n$and y_{n+1} trapezoids are formed and the area of $\frac{h}{2} * (y_n + y_{n+1})$is summed from $x_0 < x < x_n$. This yield an approximation for the area under the curve. The general formula is $\int\limits_a^b f(x)dx = h * \sum\limits_{i=a}^{b} (f(x_{i+1}) + f(x_i))$.

For each of the function the step sizes of .1, 0.05, 0.01 and 0.005 will be examined. Starting with $\int\limits_0^4 f(x)dx = \int\limits_0^4 e^{3x}dx$, the trapezoid rule results an area under the curve of 54657.64 for h=0.1. This takes 0.00027 seconds to calculate. At h=0.05 the value is 54352.9. This is approaching the actual value of 54251.3. At h=0.01 and h=0.005 the areas are 54255.34 and 54252.2 . The error at h=0.005 is 9 and the time to calculate is a mere 0.004 seconds. This is far better than the Riemann sums.

With the second function of $\int\limits_0^4 f(x)dx = \int\limits_0^4 (1+\sin(10\pi x))dx$ at h=0.1 the area yielded is 4.1, and it took 0.00037 seconds to calculate. At h=0.05 the area is 4.05, at h=0.01 area is 4.01, and finally h=0.005 the area yielded is 4.005. At h=0.005 the time it took the calculate was 0.0059 seconds, which is less than that of the Riemann sums. However the error is 0.005, instead of 0, which the Riemann sums yield.

2.3 Simpson's Rule

Simpson's rule is credited to the mathematician Thomas Simpson. The rule works by interpolating a quadratic equation for the points and then integrating that quadratic. This yield an approximation for the area under the curve. The general formula is $\int\limits_a^b f(x)dx = \approx \frac{h}{3}[f(x_0) + 2\sum_{j=1}^{n/2-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n)]$.

For each of the function the step sizes of .1, 0.05, 0.01 and 0.005 will be examined. Starting with the integral of $\int\limits_0^4 f(x)dx = \int\limits_0^4 e^{3x}dx$, and a step value of h=0.1, the

area of the curve is yielded to be 54253.7. The time it took to calculate this was 0.00036 seconds. This is already very close to the actual value of 54251.3. At h=0.05 the value is 54251.416. When the step is decreased again to h=0.01 the area becomes 54251.264 and at h=0.005 the are is 54251.263 The error at h=0.005 is 0.037. The time it took to calculate at h=0.005 was 0.0074 seconds. This is very close to the actual value and the convergence rate is high.

With the second function of $\int_0^4 f(x)dx = \int_0^4 (1+\sin(10\pi x))dx$ at h=0.1 the area yielded is 4.0, and it took 0.0004 seconds to calculate. Curiously at h=0.05, h=0.01, and h=0.001 the area yielded is also 4.0. The time for each increases from 0.00082 seconds to 0.00377 seconds and finally 0.00743. The actual value of the integral is 4.0. It appears for this case the Simpson's rule is exact as the Riemann sums were.

2.4 Comparison

Curiously interesting results were obtained from these three methods of numerical integration. As expect Riemann sums are the fastest as they are simple rectangles. Also as expect the speed in each case did not have a significant impact as with modern computers it took only fractions of a second to calculate each.

When integrating the first function of $f(x) = e^{3x}$it was seen that Simpson's rule yielded the fastest convergence and most accurate results. This was followed by trapezoid and lastly Riemann sums. The trapezoid yield a large improvement over Riemann sums, but Simpson's rule doesn't yield that large of an advantage of trapezoid. At h=0.005, the difference is 8.967, compared to the difference of 396.9 between Riemann sums and trapezoid.

Integration of the second function of $f(x) = 1 + \sin(10\pi x)$ delivered unexpected results. Due to the nature of the sine function and Riemann sums the area under the curve was exact for all step values. Simpson's rule also yielded an exact value. The trapezoid method converges to the exact value but at h=0.005, had an error of 0.005. Initial testing indicates that for sine and cosine functions Riemann sums might indeed be exact fits and trapezoids will always converge but never be exact. The exact nature of sine function and their interaction with Riemann sums would make an interesting paper, but is beyond the exact scope of this paper.

When examining both functions which were integrated and taking into account speed, error and convergence, the clear choice is Simpson's rule. From an implementations standpoint it take no more time or resources to implement than

trapezoid or Riemann sums. It has a much higher convergence rate than trapezoid or Riemann sums, and not a significant increase in calculation time. It is the most efficient method implemented.

3 Numerical ODE Solving Routines

Ordinary Differential Equations have vast applications in physics and engineering where they model various systems. Solving these numerically allows the said system to be modeled dynamically and on a computer. When solving odes it is important to remember that each method does not yield an interpolated function, instead it only gives the estimated y-value for each $x + n * h$ coordinate. The first ode to solve for is $y' = 3y$ with an initial value of $y(0) = 1$ over the range $[0, 3]$. The second ODE is $y' = \frac{1}{1+x^2} - 2y^2$ with an initial value of $y(0) = 0$ over the range $[0, 10]$. Each of these ordinary differential equations will be examined with four step sizes, 0.1, 0.05, 0.01, 0.005.

3.1 Euler's Method

Euler published his method for solving odes in 1768. The general method is given by $y_{n+1} = y_n + h * f(x_n, y_n)$, where $y' = f(x, y)$. As with each method of numerical ode solution methods this is an iterative process. In order to map out the solution each x and y coordinate are stored in a list and then draw on the graph.

As always when estimation is used, the error value is important. How good and how close is our estimations? For the purposes of this paper the maximum error given by $|f(x_n) - y_n|$ will be used.

For the function $y' = 3y$ over the range $[0, 3]$ and initial value $y(0) = 1$, four separate step values will be examined. First with a step size of h=0.1, then h=0.05, next h=0.01 and lastly h=0.005. See figure 1 for a graph showing all four step size.

When h = 0.1, the maximum error is calculated to be 5483.09. It took a mere 0.00006 seconds to calculate this. When the step size is decreased to h=0.05, it can be seen that there is an exponential growth in the time taken to calculate it. 0.00037 seconds. The error has drastically decreased to

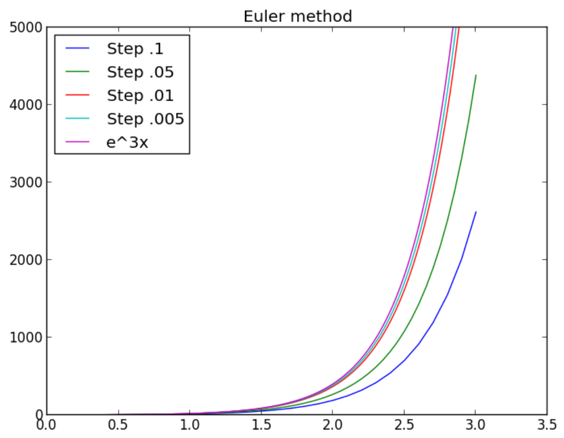

Figure 1:

3719.1 as a result of this step size change. The additional time taken to calculate is insignificant. When the step size is increased to h=0.01 and then to h=0.005 the respective maximum errors are 1004.6 and 523.8. The time taken for each is 0.00039 seconds and 0.00076 seconds. Again the error is reduced greatly with each change in step size. The additional time taken is still insignificant for the large gains in accuracy.

When examining the second initial value problem of $y' = \frac{1}{1+x^2} - 2y^2$ over the range $[0, 10]$ and initial value $y(0) = 0$, four separate step values will again be examined. First with a step size of h=0.1, then h=0.05, next h=0.01 and lastly h=0.005. See figure 2 for a graph showing all step sizes.

When h = 0.1, the maximum error is calculated to be 0.0004. It took 0.00022 seconds to calculate this. When the step size is decreased to h=0.05, as expected the error has decreased to 0.0002. This change is error is significantly less than what was seen in the first ode. The time taken as increased to 0.00039 seconds. The additional time taken to calculate is insignificant. When the step size is decreased to h=0.01 and then to h=0.005 the respective maximum errors are 0.00004 and 0.00002. The time taken for each is 0.0019 seconds and 0.0037 seconds. The error continues to decrease as expected with a smaller step size.

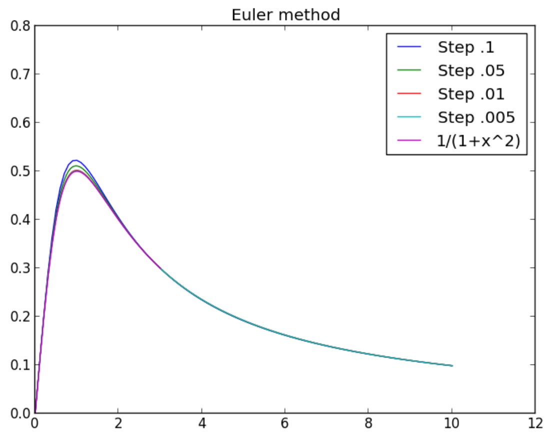

Figure 2:

Overall Euler's method appears to be fast but the accuracy decreases the further the iteration is taken. This is seen as the maximum error increases at a quadratic pace. The next logical step would seem to be

3.2 Midpoint Method

The general method is given by $y_{n+1} = h * f((x_n + 0.5 * h), (y_n + 0.5 * h * f(x_n, y_n)))$, where $y' = f(x, y)$. As with each method of numerical ode solution methods this is an iterative process. In order to map out the solution each x and y coordinate are stored in a list and then draw on the graph.

As always when estimation is used, the error value is important. How good and how close is our estimations? For the purposes of this paper the maximum error given by $|f(x_n) - y_n|$ will be used.

For the function $y' = 3y$ over the range $[0, 3]$ and initial value $y(0) = 1$, four separate step values will be examined. As with Euler's method first with a step size of h=0.1, then h=0.05, next h=0.01 and lastly h=0.005 with be used. See figure 3 for a graph showing all four step sizes and their respective graphs.

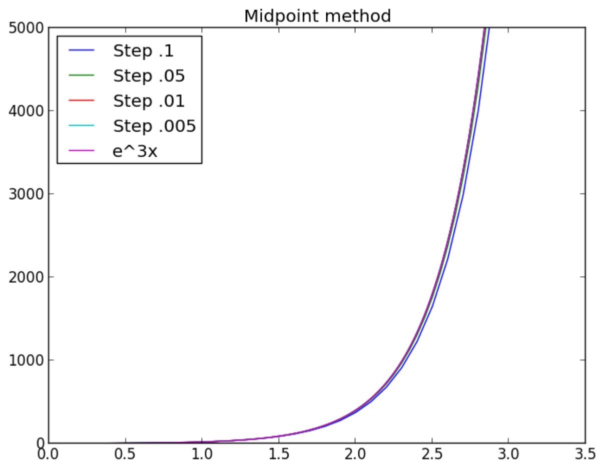

Figure 3:

When h = 0.1, the maximum error is calculated to be 830.8. It took a swift 0.00007 seconds to calculate this. Already this midpoint method is close to the level of accuracy which took the Euler's method a step size of h=0.005 to reach. When the step size is decreased to h=0.05, the error is now below the best Euler could attain at 240.9. The additional time taken to calculate is two magnitudes higher at 0.0013 but remains insignificant. When the step size is decreased to h=0.01 the maximum error becomes 10.7, and again the time taken is insignificant. At h=0.005 the maximum error is only 2.7. At 2.7, nearly a 200th of what Euler's method yielded. The additional time taken is still insignificant for the large gains in accuracy.

When examining the second initial value problem of $y' = \frac{1}{1+x^2} - 2y^2$ over the range $[0, 10]$ and initial value $y(0) = 0$, the same four step sizes will be used and graphed in figure 4.

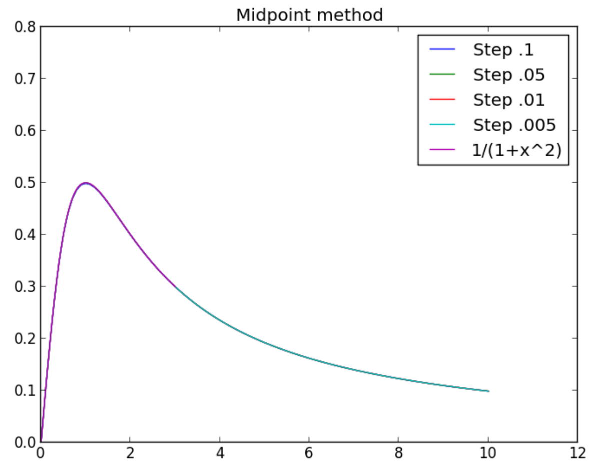

Figure 4:

When h = 0.1, the maximum error is calculated to be 0.00007. It took 0.00035 seconds to calculate this. When the step size is decreased to h=0.05, as expected the error has decreased to 0.0000017. This change is error is significantly less than what was seen in the first ode. The time taken as increased to 0.00069 seconds. The additional time taken to calculate is insignificant. When the step size is increased to h=0.01 and then to h=0.005 the respective maximum errors are $6.797e-8$ and $1.697e-8$. The time taken for

each is 0.0033 seconds and 0.0066 seconds. The error continues to decrease as expected with a smaller step size.

The accuracy of the midpoint method is vastly superior to the Euler method. The increase in calculation time is insignificant with respect to modern computing power. With a the relative speed and accuracy of this method it remains to be seen if the trapezoid method can yield any vast improvements over it.

3.3 Trapezoid Method

The iterative trapezoid method is given by $y_{n+1} = 0.5 * h * f(x_n, y_n) + 0.5 * h * f(x_{n+1}, y_{n+1})$, where $y' = f(x, y)$. Since $y_{n+1}$is on both side of the equations we can estimate y_{n+1}using Euler's method, then plug our value new y_{n+1}back into the formula to obtain a better approximation. This yields, $y_{n+1} = 0.5 * h * f(x_n, y_n) + 0.5 * h * f(x_{n+1}, y_n + h * f(x_n, y_n))$ In order to map out the solution each x and y coordinate are stored in a list and then draw on the graph.

As always when estimation is used, the error value is important. How good and how close is our estimations? For the purposes of this paper the maximum error given by $|f(x_n) - y_n|$ will be used.

For the function $y' = 3y$ over the range $[0, 3]$ and initial value $y(0) = 1$, four separate step values will be examined. As with previous methods the first step size used is h=0.1, then h=0.05, next h=0.01 and lastly h=0.005 with be used. See figure 5 for a graph showing all four step sizes and their respective graphs.

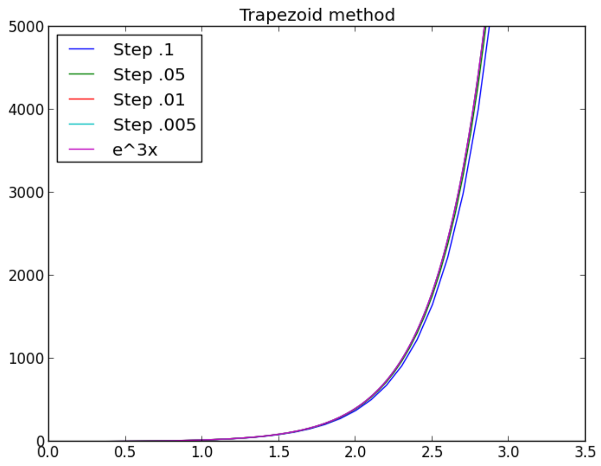

Figure 5:

When h = 0.1, the maximum error is calculated to be 830.8. This is the same level of error that the midpoint method yielded. It took 0.000087 seconds to calculate this, which is a little over twice as long as for the midpoint method. When the step size is decreased to h=0.05, the error remains the same as the midpoint method at 240.9. The additional time taken to calculate is higher at 0.0009 but remains insignificant. When the step size is decreased to h=0.01 the maximum error becomes 10.7, and again the time taken is insignificant. At h=0.005 the maximum error is only 2.7. Both of these remain nearly

exactly what the midpoint method yielded. The time taken to calculate each however was longer but insignificant.

When examining the second initial value problem of $y' = \frac{1}{1+x^2} - 2y^2$ over the range $[0, 10]$ and initial value $y(0) = 0$, the same four step sizes will be used and graphed in figure 6.

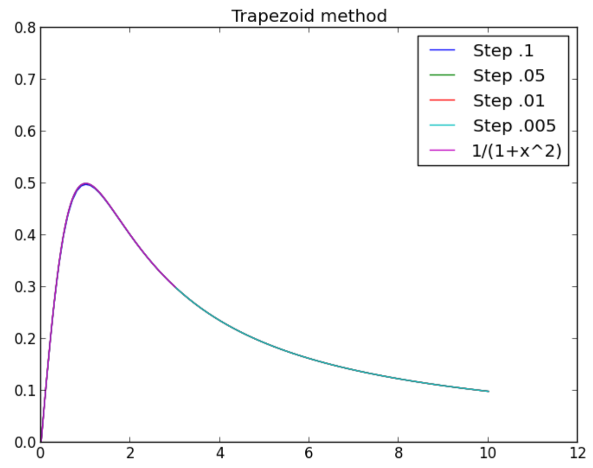

Figure 6:

When h = 0.1, the maximum error is calculated to be $9.307e - 8$. It took 0.00045 seconds to calculate this. When the step size is decreased to h=0.05, as expected the error has decreased to $2.306e-8$. This change is error is significantly less than what was seen in the first ode and remains different than the midpoint method. The time taken as increased to 0.00091 seconds. The additional time taken to calculate is insignificant. When the step size is increased to h=0.01 and then to h=0.005 the respective maximum errors are $9.15e - 8$ and $2.285e - 8$. The time taken for each is 0.004 seconds and 0.009 seconds.

We see that using this trapezoid method time it takes to calculate each ode at the different step sizes is longer than the time taken for midpoint. Although the accuracy is no better.

3.4 Comparison

The three unique methods implemented in this paper for ordinary differential equation solving each yield different results in terms of speed an accuracy. Overall speed is a non-issue as each took fractions of a second to calculate. However in comparison to each other the time take yields clues to efficiency. The measure of accuracy was the maximum error, or $f(x_n) - y_n$.

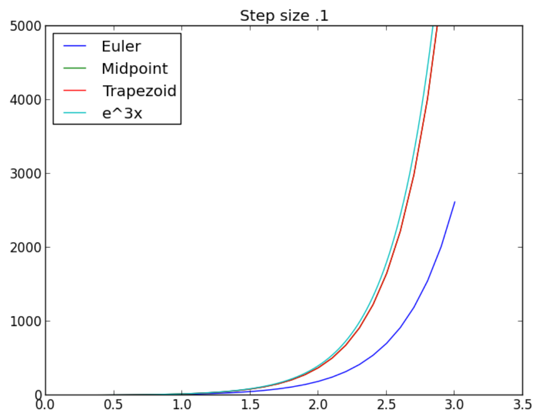

Figure 7:

When looking at the error of these methods with respect to the first ode

of $y' = 3 * y$ it can be seen quite clearly that Euler has the largest error at all step sizes. Midpoint and trapezoid method yield an error are h=0.1 which is comparable to the Euler error at h=0.005. Euler's method is magnitude of 10 slower at h=0.005 compared to midpoint when h=0.1. The midpoint method and trapezoid method is significantly more efficient that Euler's method. In figure 7 a comparison of the graphs of the three methods for h=0.1.

When examining the second ode of $y' = \frac{1}{1+x^2} - 2y^2$similar results to the first ode can be seen. With the midpoint and trapezoid methods at h=0.1 it is comparable to the Euler method when the step size is .005. At h=0.05 both the midpoint and trapezoid surpass the accuracy of the Euler method at h=0.005. In a similar manor to the first ode again, the midpoint and trapezoid methods are close in their maximum errors. However the midpoint method continues to be faster. As a result the most efficient method, where efficiency is (time taken)/(maximum error), is midpoint. With further optimizations or use of in-line c, the trapezoid method might gain the advantage in speed and become more efficient as is expected. However in pure python it can be concluded the midpoint method is the best in most cases.

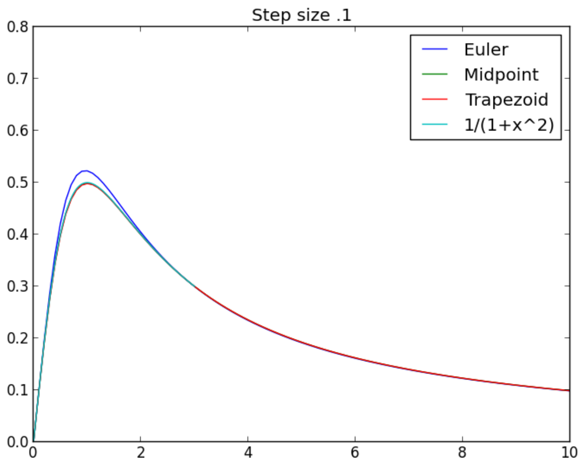

Figure 8:

4 Conclusion

The goal of this paper was to implement various methods of numerical analysis. It sought to study the factor and difficulties in implementing these methods and formulas in the language of python. First numerical integration methods were studied for both error and convergence. It was found that Simpson's rule yielded the fastest convergence. Riemann sums was the quickest in terms of raw speed to calculate but it had a large error and slow convergence. The time for each method's computation was well under a second so speed was not the most important factor.

The second field examined was numerical solutions to ordinary differential equations. In the field of ordinary differential equations there were three methods examined and studied. First classic Euler's method was used to solve and graph

the ode. As Euler's method relies on the previous term only the resulting error was large and slow to converge to the actual value of the function. The midpoint method improved on this and greatly reduced the maximum error. The trapezoid method interestingly produced the same maximum error or slightly higher errors than the midpoint method. This is likely a result of the iteration method and having to estimate for y_{n+1}first. As a result the midpoint method was found to be the most efficient. With errors at or less than those of the trapezoid the calculation time was also significantly less in comparison. Overall the times were well below a second, and are not significant. However comparing each method to itself, it seems the midpoint method is the better method for these two ode functions which were approximated.

When implementing numerical analysis methods of any topic it is highly important to examine and cater to the platform that is being used. One must take into account any optimizations if possible, and what level of error one is will to except. With modern computers and simplistic problems speed is not of great importance, however with more complex problems, one might wish to have a faster method, such as trapezoid method over Simpson's rule if the slight increase in error is not of importance.

5 Appendix A