Student ID: 8444507 Name: M. Kashif Hussain

Scientific Computing Coursework 3

Q1)

We are given the following ODE:

$$\frac{dy}{dx} = xe^{3x} - 2y, \qquad 0 \le x \le 1\,, \qquad y(0) = 0.$$

By using the Euler method with the following algorithm:

$$\tilde{y}_{i+1} = \tilde{y}_i + hf(x_i, \tilde{y}_i), \qquad i = 0, 1, \dots, n-1, \qquad \tilde{y}_0 = \propto,$$

where $\tilde{y}_i$ is an approximation to $y(x_i)$, where $x_i = a + ih$ and $h = (b-a)/n$ is the step size we can integrate the above ODE.

The C++ program below outputs a table which we can use to calculate the approximation to the above problem using the Euler method.

```
#include <iostream>
#include <math.h>
#include <fstream>
#include <vector>
using namespace std;

//Below function uses the Euler method to calculate the integral of an ODE
void eulerMethod(vector<double> &y, vector<double> &xVals, int n, double a, double b)
{
	//Sets the stepsize
	double stepSize = (b - a) / n;
	//Stores the actual equation (for clarity)
	double eqnVal;
	//Loops through all steps and stores the x and y values at each step
	for (int i = 0; i <= n; i++)
	{
		xVals[i] = a + i*stepSize;
		eqnVal = xVals[i] * exp(3*xVals[i]) - 2*y[i];
		y[i+1] = y[i] + stepSize*eqnVal;
	}

}

int main()
{
	//Stores value of n
	int nValue = 10;
	//Minmum x value from range of values
	double xMin = 0;
	//Maximum x value from range of values
	double xMax = 1;
	//Stores all values for x in a vector
	vector<double> xValues(nValue + 1);
	//Stores all values for y in a vector (ie. the integration values)
	vector<double> integrationValues(nValue + 2);
	//sets the initial condition  ie. y(0) = 0.
	integrationValues[0] = 0;
	//Runs the function which produces and stores the values of the Euler method at each step
	eulerMethod(integrationValues, xValues, nValue, xMin, xMax);

	//variable of type ofsteam called eulerFile
	ofstream eulerFile;
	//Opens specified text document for writing
	eulerFile.open("Euler Approximations Tabulated.txt");
	//if attempt to open file failed, returns 1
	if (!eulerFile) return 1;
```

```
	// Will return 11 vals (as we include 0 and increasing in stepsize value)
	for (int i = 0; i <= nValue; i++)
	{
		//Write 3 columns to file with values listed below
		eulerFile.width(10); eulerFile << i;
		eulerFile.width(10); eulerFile << xValues[i];
		eulerFile.width(15); eulerFile << integrationValues[i] << endl;
	}

	return 0;
}
```

Below are the tabulated results of running the program above over the range $0 \leq x \leq 1$.

Value of i (Step Number)	Value of x_i	Value of $\tilde{y}_i$
0	0	0
1	0.1	0
2	0.2	0.0134986
3	0.3	0.0472412
4	0.4	0.111581
5	0.5	0.22207
6	0.6	0.40174
7	0.7	0.684371
8	0.8	1.11913
9	0.9	1.77716
10	1	2.7609

Q2)

By modifying my original program with the following code:

```
	//Generates tabulated results of Eulers methods for n = 10 to 50 in steps of 10
	while (nValue <= 50)
	{
		eulerMethod(integrationValues, xValues, nValue, xMin, xMax);
		//Incrementing i by 1 results in all values from 1 to nValue
		//Inceremented by nValue/10 allows us to directly compare.
		for (int i = 0; i <= nValue; i += nValue/10)
		{
			//Write 3 columns to file with values listed below
			eulerFile.width(10); eulerFile << i;
			eulerFile.width(10); eulerFile << xValues[i];
			eulerFile.width(15); eulerFile << integrationValues[i] << endl;
		}
		eulerFile << endl;
		//Increment nValue by 10
		nValue += 10;
		//Resize all vectors as necessary
		xValues.resize(nValue + 1);
		integrationValues.resize(nValue + 2);
	}
```

we can analyse the effect of increasing the value of n (decreasing the step size). Below are tables produced by the above code which we can directly compare with our original table with $n = 10$.

Student ID: 8444507 Name: M. Kashif Hussain

Table for n = 20:

Value of i (Step Number)	Value of x_i	Value of $\tilde{y}_i$
0	0	0
2	0.1	0.00290459
4	0.2	0.0201894
6	0.3	0.059215
8	0.4	0.131178
10	0.5	0.252808
12	0.6	0.448805
14	0.7	0.755304
16	0.8	1.22482
18	0.9	1.93324
20	1	2.98972

Table for n = 30:

Value of i (Step Number)	Value of x_i	Value of $\tilde{y}_i$
0	0	0
3	0.1	0.00386033
6	0.2	0.0224061
9	0.3	0.0631997
12	0.4	0.137717
15	0.5	0.26308
18	0.6	0.464542
21	0.7	0.779026
24	0.8	1.26016
27	0.9	1.98542
30	1	3.06619

Table for n = 40:

Value of i (Step Number)	Value of x_i	Value of $\tilde{y}_i$
0	0	0
4	0.1	0.00433578
8	0.2	0.0235113
12	0.3	0.0651894
16	0.4	0.140985
20	0.5	0.268216
24	0.6	0.472413
28	0.7	0.790891
32	0.8	1.27784
36	0.9	2.01151
40	1	3.10443

Student ID: 8444507 Name: M. Kashif Hussain

Table for n = 50:

Value of i (Step Number)	Value of x_i	Value of $\tilde{y}_i$
0	0	0
5	0.1	0.00462025
10	0.2	0.0241733
15	0.3	0.0663822
20	0.4	0.142945
25	0.5	0.271298
30	0.6	0.477135
35	0.7	0.798011
40	0.8	1.28844
45	0.9	2.02717
50	1	3.12737

Using the following MATLAB code, we can generate graphs to compare the tables above:

```
function compareExp()
%Split one file into 5 with the respective tables
data1 = load('Euler Approximations Tabulated 50 - 1.txt');
data2 = load('Euler Approximations Tabulated 50 - 2.txt');
data3 = load('Euler Approximations Tabulated 50 - 3.txt');
data4 = load('Euler Approximations Tabulated 50 - 4.txt');
data5 = load('Euler Approximations Tabulated 50 - 5.txt');
x = 0:0.1:1;
%Actual Solution:
y = 1/25.*exp(-2.*x).*(5.*exp(5.*x).*x - exp(5.*x) + 1);
%Graphs:
plot(data1(:,2), y, 'k--');
hold on;
plot(data1(:,2), data1(:,3), 'g-o');
plot(data2(:,2), data2(:,3), 'r-*');
plot(data3(:,2), data3(:,3), 'b-s');
plot(data4(:,2), data4(:,3), 'c-d');
plot(data5(:,2), data5(:,3), 'm-+');
legend('Exact Solution','n = 10', 'n = 20', 'n = 30', 'n = 40', 'n = 50');
xlabel('Values of x');
ylabel('Values for y (approximations)');

end
```

The graphs below shows that as n increases, our approximation for $\tilde{y}_n$ grows closer to the true solution. This is because our value for $h = \frac{b-a}{n}$ gets closer and closer to 0. As a result we have the following:

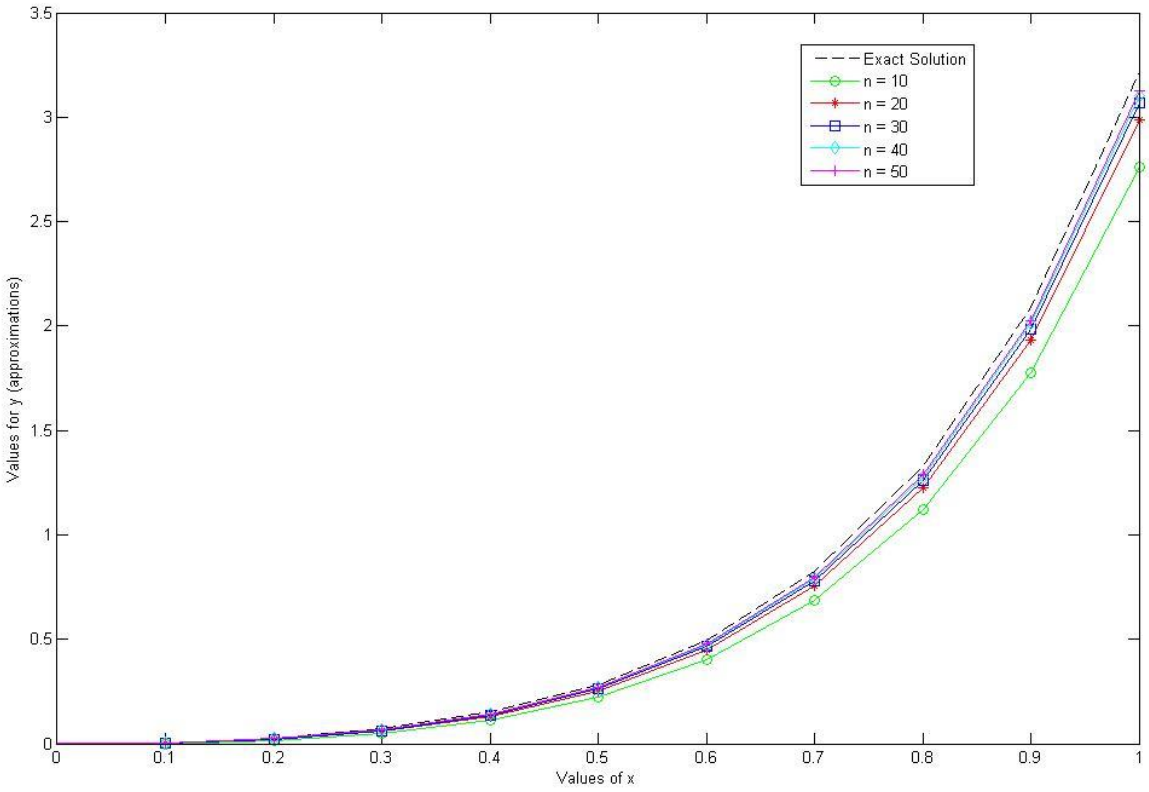
$$\tilde{y}_i \to y(x_e) \ \ as\ h \to 0$$

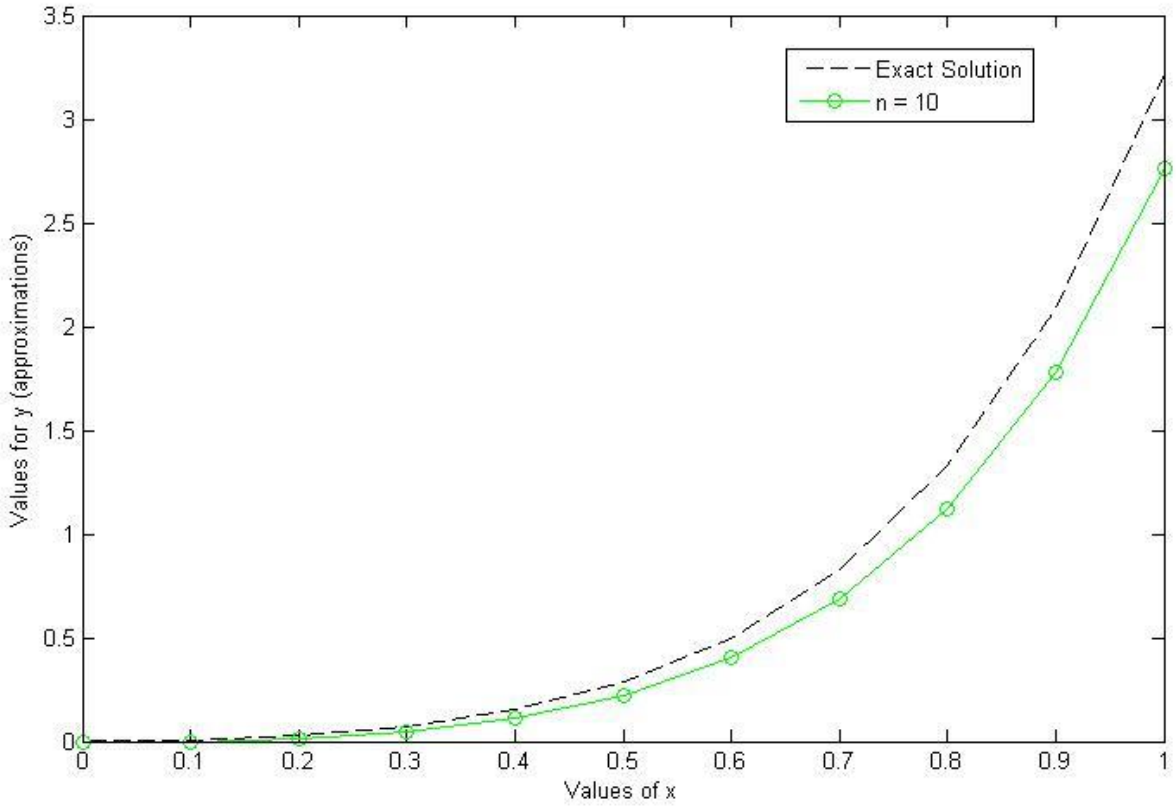
Where $x_e = ih$ for the ith step.
Remembering that the Taylor series approximation is the following:

$$\tilde{y}_{i+1} = \tilde{y}_i + hf(x_i, \tilde{y}_i) + O(h^2),$$

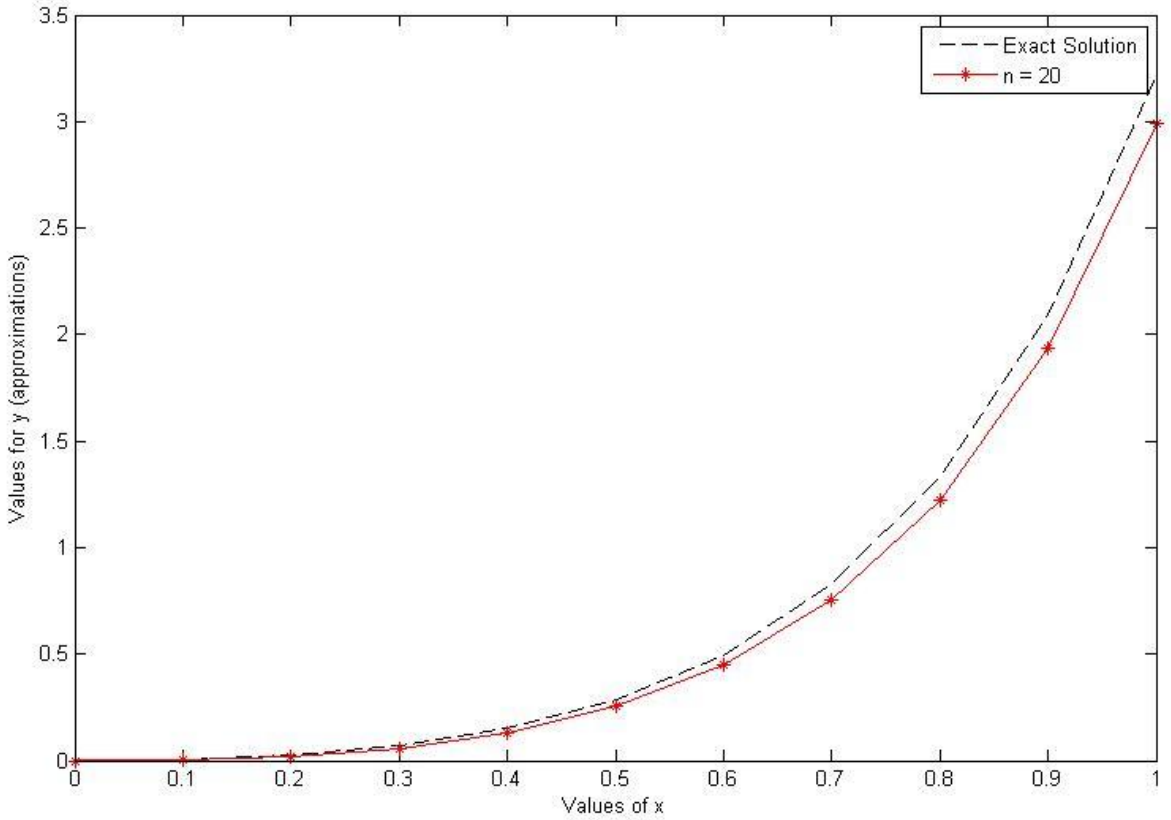
We can see that as $h \to 0$, all terms of $O(h^2)$ and higher become very small and negligible and we are left with $\tilde{y}_{i+1} = \tilde{y}_i + hf(x_i, \tilde{y}_i)$

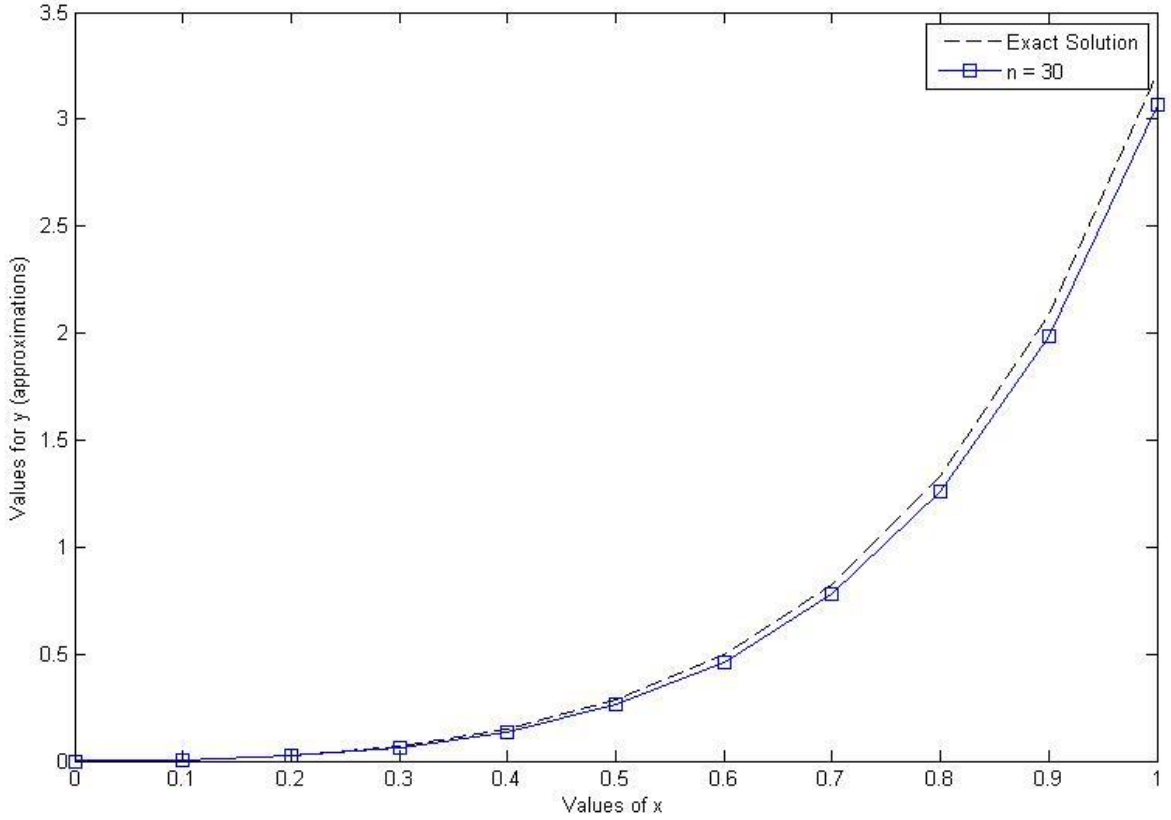
Student ID: 8444507 Name: M. Kashif Hussain



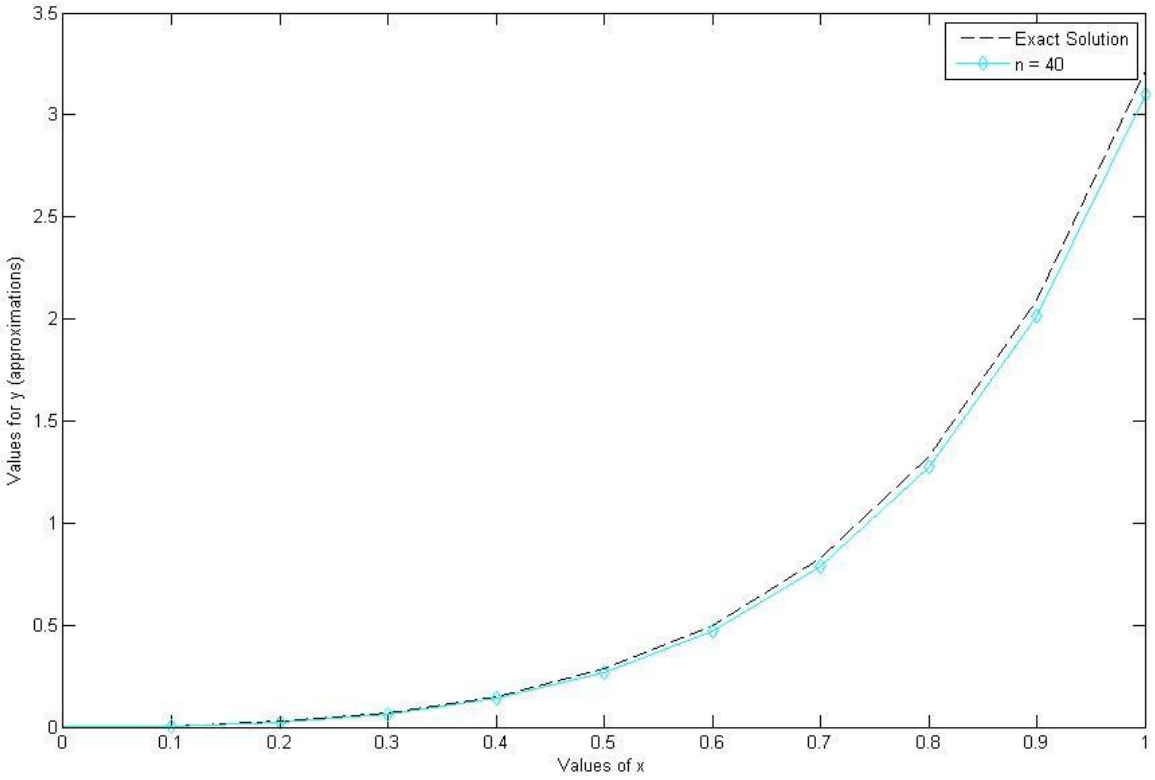


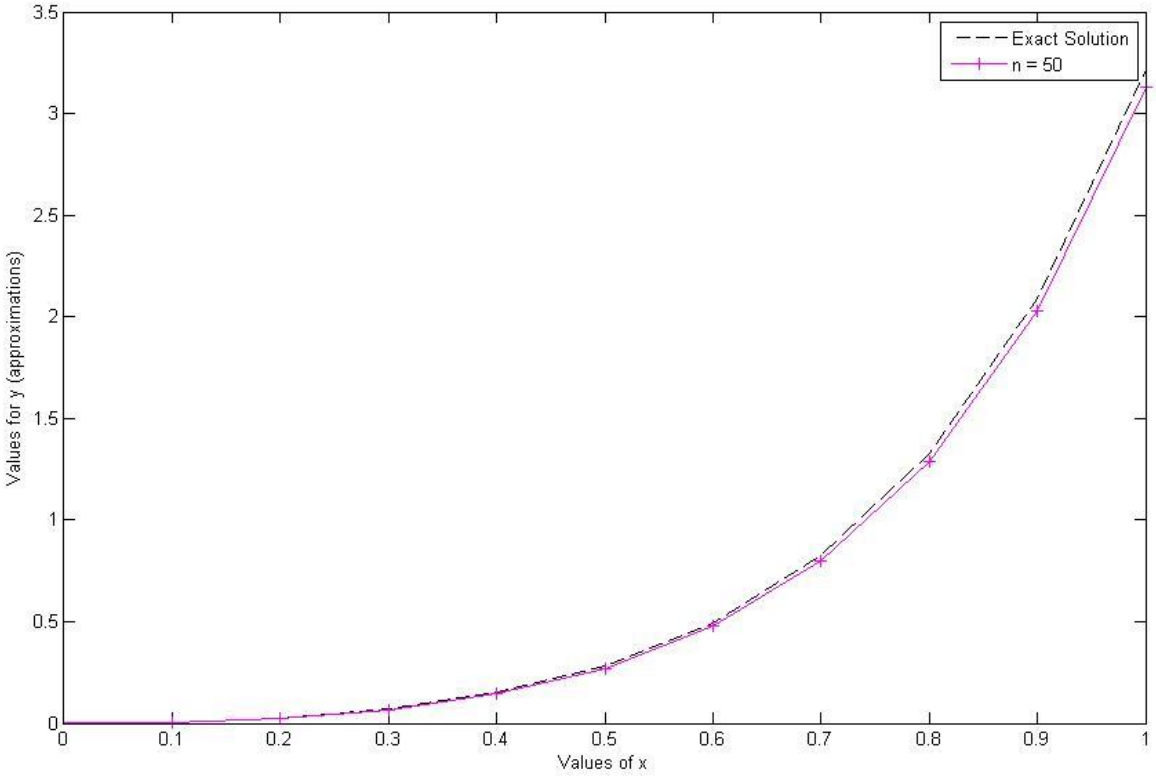
Student ID: 8444507 Name: M. Kashif Hussain





Student ID: 8444507 Name: M. Kashif Hussain





The above graphs display the results of the table as listed.

Student ID: 8444507 Name: M. Kashif Hussain

Q3)

By editing our initial program with the following:

```
//Stores value of n
int nValue = 100;
//Minmum x value from range of values
double xMin = 0;
//Maximum x value from range of values
double xMax = 1;
//Stores all values for x in a vector
vector<double> xValues(nValue + 1);
//Stores all values for y in a vector (ie. the integration values)
vector<double> integrationValues(nValue + 2);

ofstream eulerFile;
//Opens specified text document for writing
eulerFile.open("Euler Approximations Tabulated Q3.txt");
//if attempt to open file failed, returns 1
if (!eulerFile) return 1;
//Number of values for y(0)
int noOfInitialValues = 5;
//Setting y(0) = -1
integrationValues[0] = -1;
for (int i = 1; i <= noOfInitialValues; i++)
{
	eulerMethod(integrationValues, xValues, nValue, xMin, xMax);
	for (int i = 0; i <= nValue; i += nValue/10)
	{
		//Write 3 columns to file with values listed below
		eulerFile.width(10); eulerFile << i;
		eulerFile.width(10); eulerFile << xValues[i];
		eulerFile.width(15); eulerFile << integrationValues[i] << endl;
	}
	eulerFile << endl;
	// Resize all vectors as necessary
	xValues.resize(nValue + 1);
	integrationValues.resize(nValue + 2);
	//Increment value for y(0) by 0.5
	integrationValues[0] += 0.5;

}
```

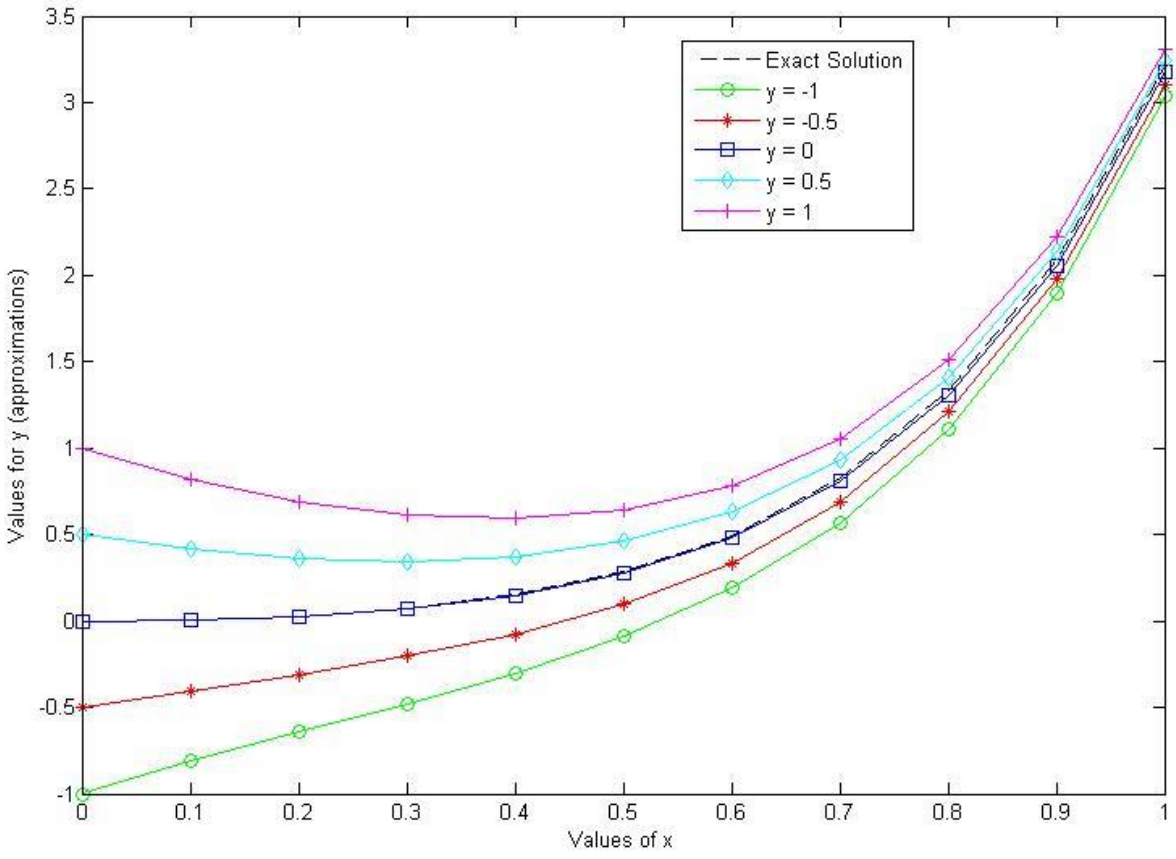
we can generate a table showing how the approximations vary for differing initial values for $y(0)$ in the range

$$-1 \leq y \leq 1.$$

Below is a plot of the results choosing n = 100 (to achieve a result very close to the exact solution)

As you can see, setting y = 0 sets the approximation almost exactly onto the exact solution. Further, setting y to the above specified range, we can see that it converges to the true solution as expected no matter what we set the initial condition as.

Student ID: 8444507 Name: M. Kashif Hussain



The above plot was produced using the following MATLAB code:

```
function compareExp()
%Split one file into 5 with the respective tables
data1 = load('Euler Approximations Tabulated Q3 - 1.txt');
data2 = load('Euler Approximations Tabulated Q3 - 2.txt');
data3 = load('Euler Approximations Tabulated Q3 - 3.txt');
data4 = load('Euler Approximations Tabulated Q3 - 4.txt');
data5 = load('Euler Approximations Tabulated Q3 - 5.txt');
x = 0:0.1:1;
%Actual Solution:
y = 1/25.*exp(-2.*x).*(5.*exp(5.*x).*x - exp(5.*x) + 1);
%Graphs:
plot(data1(:,2), y, 'k--');
hold on;
plot(data1(:,2), data1(:,3), 'g-o');
plot(data2(:,2), data2(:,3), 'r-*');
plot(data3(:,2), data3(:,3), 'b-s');
plot(data4(:,2), data4(:,3), 'c-d');
plot(data5(:,2), data5(:,3), 'm-+');
legend('Exact Solution','y = -1', 'y = -0.5', 'y = 0', 'y = 0.5', 'y = 1');
xlabel('Values of x');
ylabel('Values for y (approximations)');

end
```