THE COLLEGE OF WILLIAM AND MARY

Programming Assignment #4

Alexander Powell

December 3, 2014

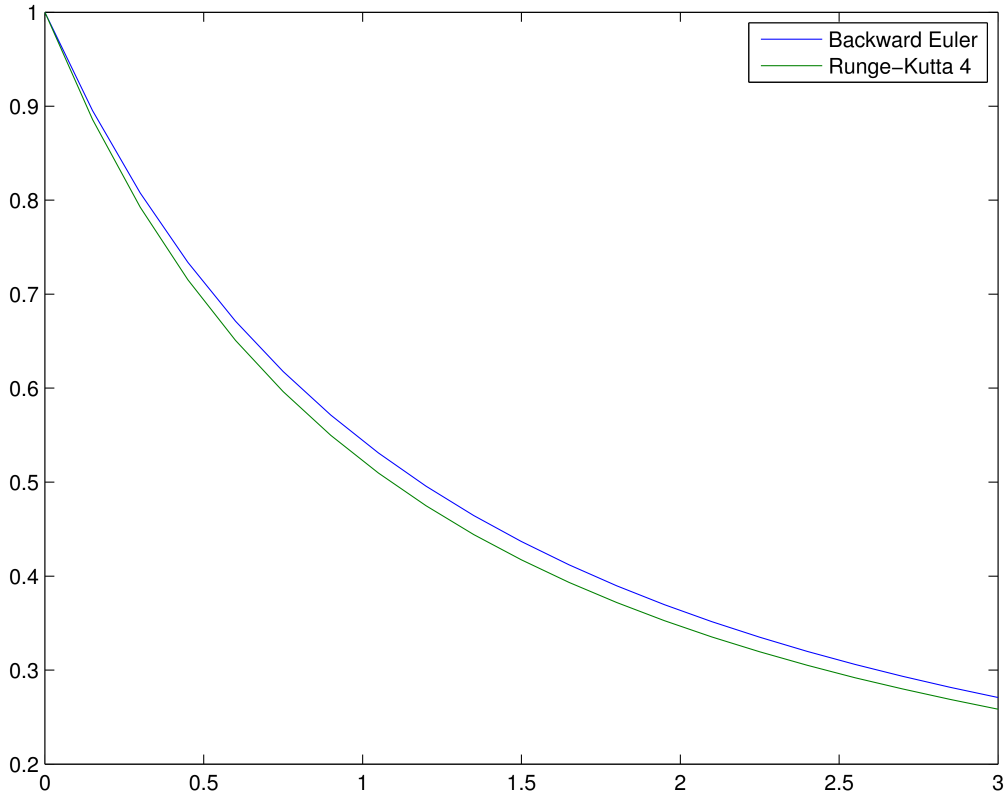
1 INITITAL VALUE PROBLEM

1. Newton's method can be an extremely fast method of finding the roots of a function. It is stated as follows:
$$P_n = P_{n-1} - \frac{f(P_{n-1})}{f'(P_{n-1})}$$
where the initial P value is some guess of where the root of the function is located. Now, we know from the backwards Euler method that $w_0 = \alpha$ and $w_{i+1} = w_i + h \cdot f(t_{i+1}, w_{i+1})$. Following from Newton's method we can write $F(w) = w - w_i - h \cdot f(t_{i+1}, w) = 0$ and $F'(w) = 1 - h \cdot f_y(t_{i+1}, w)$. Also, $w_{i+1}^{(0)} = w_i$. From this we can fill in the equation to state the following:
$$w_{i+1}^{(k)} = w_{i+1}^{(k-1)} - \frac{w_{i+1}^{(k-1)} - w_i - h \cdot f(t_{i+1}, w_{i+1}^{(k-1)})}{1 - h \cdot f_y(t_{i+1}, w_{i+1}^{(k-1)})}$$

2. My implemenation of *backeuler.m* is displayed below. It uses a slight modification of *newton.m* from the previous programming assignments as well as the built-in MATLAB function *zeros()*.

```
function [t, w] = backeuler(f,dfdy,a,b,alpha,N,maxiter,tol)
h = (b - a)/N;
y = zeros(N,1);
ti = zeros(N,1);
y(1) = alpha;
ti(1) = a;
for i = 1:N
    th = ti(i) + h;
    init = y(i);
    f1 = @(x) x - init - h.*f(th,x);
    dfdy1 = @(x) 1 - h.*dfdy(th,x);
    y(i+1) = newton(f1,dfdy1,init,tol,maxiter);
    ti(i+1) = th;
end
t = ti;
w = y;
end
```

The plot of The Backward Euler vs. Runge-Kutta 4 method is displayed below. It is evident that the two approximations are similar.



3. The combustion model equation is given by

$$y' = y^2(1-y), 0 \le t \le 2000, y(0) = 0.9$$

   a) We know the number of steps required is given by $h < 2\sqrt{2} \approx 2.828427$ and

   $$N > \frac{2000-0}{2\sqrt{2}} \approx 707.107$$

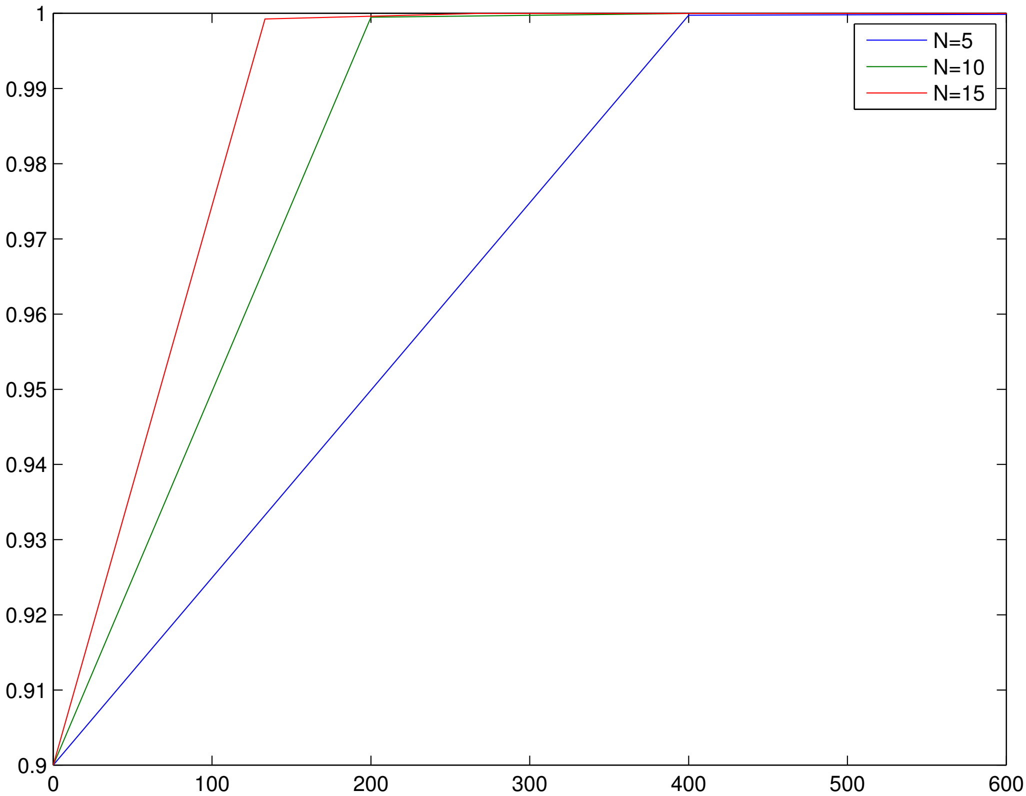
   Solving using *rk4.m* with N set to 707 results in t approaching a value of 2 and w approaching a value of 0.9138102.

b) The following code was used to solve the ODE with *backeuler.m*.

```
f = @(t,y) y*y*(1-y);
df = @(t,y) 2*y-3*y*y;
a = 0; b = 2000;
alpha = 0.9;
maxiter = 50; tol = 1e-12;
[t1,w1] = backeuler(f,df,a,b,alpha,5,maxiter,tol);
[t2,w2] = backeuler(f,df,a,b,alpha,10,maxiter,tol);
[t3,w3] = backeuler(f,df,a,b,alpha,15,maxiter,tol);
plot(t1,w1,t2,w2,t3,w3)
```

It appears that N can be significantly large but of course it can't be smaller than 1. Therefore, h can be as large as 2000.

The plot generated by the code above is displayed below.



Furthermore, the backward euler method appears to approach 1 no matter the step size so it is stable and is suitable for the solution of stiff differential equations.

2 MONTE CARLO INTEGRATION

1. The monte carlo simulation of the integral from problem 2 was implemented using the code given below. The built-in MATLAB functions *zeros()* and *sum()* were used.

```
function [ integral ] = monte_carlo( N )
a = zeros([1,N]);
for i = 1:N
    a(i) = 2*rand - 1;
end
b = zeros([1,N]);
for i = 1:N
    b(i) = 2*rand - 1;
end
x = a;
y = b;
temp = zeros([1,N]);
for j = 1:N
    if ((x(j).^2 + y(j).^2) < 1)
        temp(j) = 1;
    else
        temp(j) = 0;
    end
end
c = temp;
total = sum(c);
integral = 4.*(1/N).*(total);
end
```

To implement step 4 of the Monte Carlo simulation, I calculated the integral 1000 times and took the average using the following shell commands.

```
>> t = zeros([1,1000]);
>> for i = 1:1000
>>     t(i) = monte_carlo(100000);
>> end
>> mean(t)
```

2. The following table presents the computed values of the integral for the given values of N with their respective relative errors.

Integrals and their Relative Errors

N	Integral Approx	Relative Error
2	3.20600000	0.0205014951
5	3.16320000	0.0068778319
10	3.13120000	0.0033080843
100	3.13736000	0.0013472954
1000	3.14117200	0.0001338981
10000	3.14101240	0.0001847004
100000	3.14139904	0.0000616291

It is clear that as N becomes larger the relative errors become smaller, thus giving you a more accurate approximation to the actual value of π.

The figure below is a log-log plot of N versus the respective errors. This further shows how using a larger N returns a smaller relative error.