```
% Will Kramlinger
% CE 3101
% Gonella
% 4/22/14
% HW7

%% PROBLEM 1

edit
```

```
function dydx = HW7(x,y)
dydx = .1 * y *(100 * (1 - x/100) - y);
end
```

```
function [x,y] = odeEULER(ODE,a,b,h,yINI)
% Will Kramlinger; 4/17/14
% Code appropriated from the textbook.

% odeEULER solves a first-order initial value ODE using Euler's explicit
% method.
% Input variables:
% ODE   Name for the function that calculates dy/dx.
% a     The first value of x.
% b     The last value of x.
% h     Step size.
% yINI  The value of the solution y at the first point (initial value).
% Output variables:
% x     A vector with the x coordinate of the solution points.
% y     A vector with the y coordinate of the solution points.

x(1) = a; y(1) = yINI;
N = (b-a)/h;
for i = 1:N
    x(i+1) = x(i) + h;
    y(i+1) = y(i) + h*ODE(x(i),y(i));
end

% plot(x,y);
end
```

```
function [x,y] = odeModEuler(ODE,a,b,h,yINI)
% Will Kramlinger; 4/17/14
% Code appropriated from textbook.

% odeModEuler solves a first order ODE using the modified Euler method.
% Input variables:
% ODE   Name for the function that calculates dy/dx.
% a     The first value of x.
% b     The last value of x.
% h     Step size.
% yINI  The value of the solution y at the first point (initial value).
% Output variables:
% x     A vector with the x coordinate of the solution points.
% y     A vector with the y coordinate of the solution points.

x(1) = a; y(1) = yINI;
N = (b - a)/h;
```

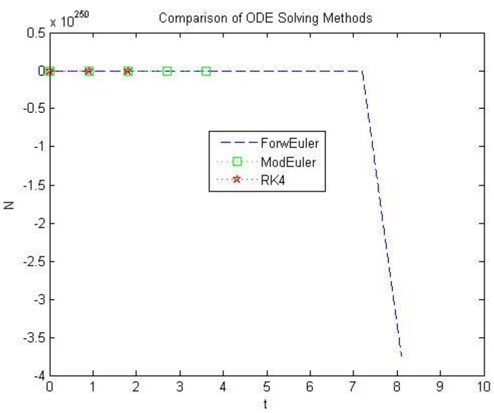
```
for i = 1:N
    x(i+1) = x(i) + h;
    SlopeEu = ODE(x(i),y(i));
    yEu = y(i) + SlopeEu*h;
    SlopeEnd = ODE(x(i+1),yEu);
    y(i+1) = y(i) + (SlopeEu + SlopeEnd) * h/2;
end

% plot(x,y);
end
```

```
function [x,y] = odeRK4(ODE,a,b,h,yINI)
% Will Kramlinger; 4/17/14
% Code appropriated from textbook.

% odeRK4 solves a first order initial value ODE using Runge-Kutta fourth
% order method.
% Input variables:
% ODE   Name for the function that calculates dy/dx.
% a     The first value of x.
% b     The last value of x.
% h     Step size.
% yINI  The value of the solution y at the first point (initial value).
% Output variables:
% x     A vector with the x coordinate of the solution points.
% y     A vector with the y coordinate of the solution points.

x(1) = a; y(1) = yINI;
N = (b - a)/h;
for i = 1:N
    x(i+1) = x(i) + h;
    K1 = ODE(x(i),y(i));
    xhalf = x(i) + h/2;
    yK1 = y(i) + K1*h/2;
    K2 = ODE(xhalf,yK1);
    yK2 = y(i) + K2*h/2;
    K3 = ODE(xhalf,yK2);
    yK3 = y(i) + K3*h;
    K4 = ODE(x(i+1),yK3);
    y(i+1) = y(i) + (K1 + 2*K2 + 2*K3 + K4)*h/6;
end

% plot(x,y);
end
```

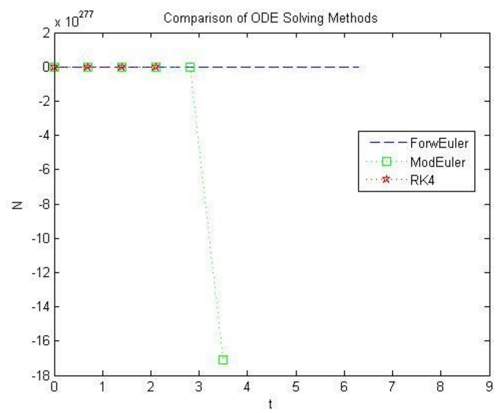
```
function graph = superimpose(a,b,h,yINI)
% Will Kramlinger; 4/17/14
% The function plots the solutions of a differential equation, in this
% case the equation HW7, using previously written forward Euler,
% modified Euler, and RK4 functions.
% Input variables:
% a = The first value of x.
% b = The last value of x.
% h = Step size.
% yINI = The value of the solution y at the first point (initial value).
% Output variables:
```

```
% graph = instructs user that this variable is irrelevant and to look
%         at the graph

[xFE,yFE] = odeEULER(@HW7,a,b,h,yINI);
plot(xFE,yFE,'--');

hold on

[xMod,yMod] = odeModEuler(@HW7,a,b,h,yINI);
plot(xMod,yMod,'g:s');

[xRK4,yRK4] = odeRK4(@HW7,a,b,h,yINI);
plot(xRK4,yRK4,'r:p');

title('Comparison of ODE Solving Methods');
xlabel('t'); ylabel('N');
legend('ForwEuler','ModEuler','RK4','Location','best');

graph = 'Look at the graph, doggy.';

hold off
end
```

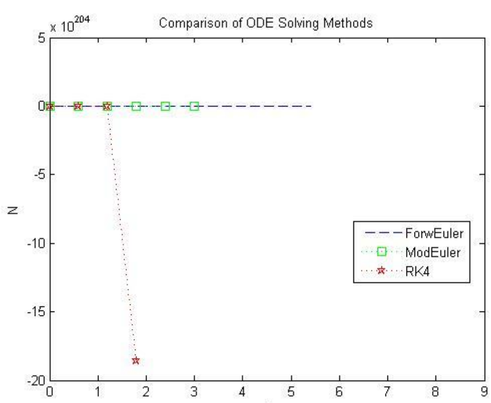
```
graph = superimpose(0,9,.9,90) % ...and repeated for smaller step sizes
```



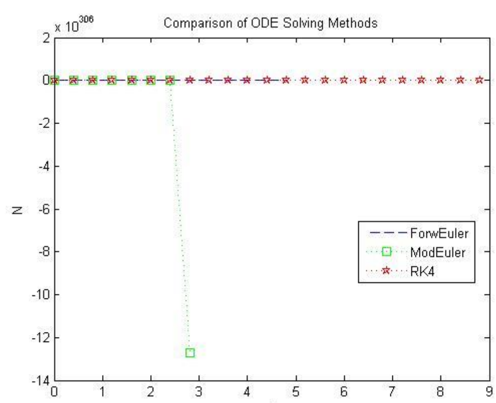
h = .9



h = .7

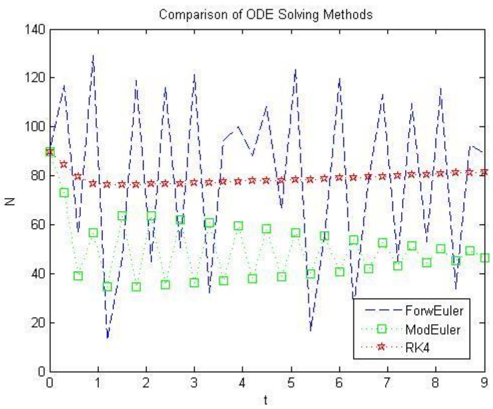


h = .6

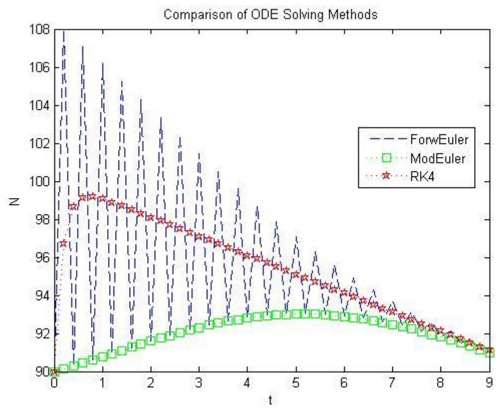


h = .4

```
% In the approximate range .3 < h < .9, all three methods tend to produce
% solutions of N ~= 0 (or presumably some constant on the order of 10^1),
% then often tend to -infinity; the step size is too large.  It is
% interesting to note that at about h = .4, the RK4 method is the first of
% the methods not to tend to -infinity for 0 < t < 9.  Then, Modified
% Euler and Forward Euler follow suit, respectively.
```

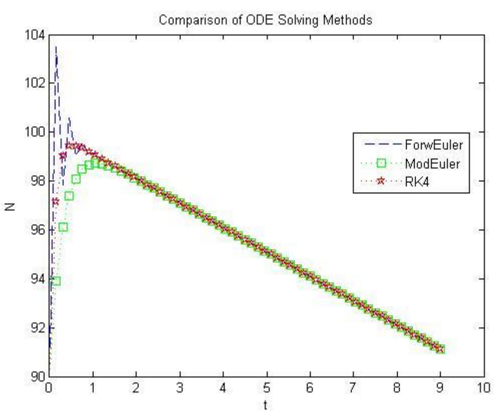


h = .3

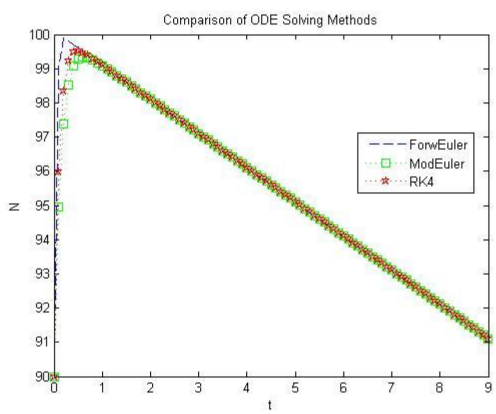


h = .2

```
% At h = ~.3, the solutions begin to take on distinctive characters.  The
% RK4 solution can be approximated by N = 80 (which is why I assume N is
% slightly greater than 0 in the preceding paragraph).  The other 2
% solutions suggest N oscillates for 0 < t < 9.

% At around h = .2, the RK4 method begins to resemble the assumed actual
% solution.  Forward Euler still gives an oscillating function (though
% less erratic) while Modified Euler starts to resemble a curve. For all
% 3, the range of N values get closer to those of the actual solution.
```



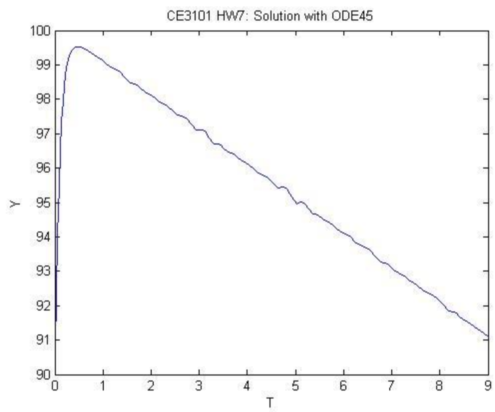
h = .15



h = .1

```
% As h decreases from .2 to .1, Modified Euler and Forward Euler,
% respectively, both begin to converge to the solution given by RK4.
% Slight oscillatory character is still visible for the Forward Euler
% solution at h = .15  At h = .1, all three solutions resemble the actual
% solution.
```

```
% As the step size is decreased past h = .1, there is very little
% discernable change in the solutions and the differences between them.

% The trends seen in the solutions due to decreasing step size agree with
% those which can be intuitively predicted.  In terms of expected global
% error, RK4 < Modified Euler < Forward Euler.  Thus, the solutions given
% by the three methods should begin to converge (if they do) to the actual
% solution in that order, which is seen above.  The lack of any predictor-
% corrector characteristics for Forward Euler results in it providing a
% misleading oscillating solution for a relatively wide range of h values.
% None of the methods are immune to a step size which is too large.

[T,Y] = ode45(@HW7,[0 9],90);
plot(T,Y);
```




```
% The solution obtained using ode45 seems to be a reasonable
% representation of the solutions obtained with the other methods, once
% their step sizes are ~.1 or lower.  Given the ease of use of ode45, as
% well as the lack of mandatory step size specification, it appears to be
% a very convenient but still relatively accurate function.
```