Homework 2 Problem 1

AE370 - Spring 2018

Emilio R. Gordon

Problem: The function

$$f(x) = 2x^3 + 5.875x^2 - 8.625x - 24.75$$

has one root in the interval $1 \leq x \leq 4$. Find it using:

- Bisection
- the Newton-Raphson method
- Secant Method

Solution: Using the algorithms provided in class and the partial matlab code provided from the assignemnt, the function given above produces a single root shown below. As expected, each method produced like results given the specified tollerance of 1×10^{-6}

- Bisection: $\hat{x} = 2.062500$
- Newton-Raphson: $\hat{x} = 2.062500$
- Secant method: $\hat{x} = 2.062500$

Note: If the tolerance were to be decreased, each method would produce different results depending on how loose the tolerance had become.

Bisection Method:

Recall that the basic idea of the bisection method is to bracket the root until the interval size is small enough for us to define the root. The algorithm below does just that by finding the middle point between the lower interval and higher interval. After which, we check to see if the root is above or below that middle point. If it is above, then our lower inteval becomes the mid-point. If it is lower, then our higher interval becomes our midpoint. This is repeated until the difference of our high and low intervals is within tolerance. This is shown in Figure 1 below.

```
function x = bisection(f, x_low, x_high, x_tol)
% Isolate a root of f(x) using bisection.
while x_high-x_low > x_tol
    c = (x_low+x_high)/2;
    if fn(x_low)*fn(c) > 0
        x_low = c;
    else
        x_high = c;
    end
end
x = c;
end
```

Figure 1: Root Finding via Bisection Method

Newton-Raphson Method:

Recall the basic procedure for the Newton-Raphson method:

- Start from an initial guess of the root x_1
- For each iteration step, $k = 1, 2, 3, ...$
  - check for convergence : $||f(x_k)|| < Tolerance$
  - if so, exit: you have found the approximate root
  - if not, compute the correction increment : $\Delta x_k = -\frac{f(x_k)}{f'(x_k)}$
  - then update the solution: $x_{k+1} = x_k + \Delta x_k$

```
function x = newton_raphson(f, fp, x_0, tol)
% Isolate a root of f(x) using Newton-Raphson iteration.
x = x_0;
while abs(fn(x))>tol
    dx = -fn(x)/fnp(x);
    x = x+dx;
end
end
```

Figure 2: Root Finding via Newton-Raphson Method

Secant Method:

Recall that the basic idea for the secant method came about due to the problem that the Netwon-Raphson method required the computation of both $f(x)$ and $f'(x)$ at each iteration which was inconvienient and computationally expensive.

Instead, the Secant method allowed for a successive $f(x_k)$ to approach the derivative such that:

$$x_{k+1} = x_k - f(x_k)\,\frac{x_k - x_{k-1}}{f x_k - f(x_{k-1})}$$

```
function x = secant(f, x_0, x_1, tol)
% Isolate a root of f(x) using the secant method.
x = x_0;
while abs(fn(x))>tol
    xx = x_1 - fn(x_1)*((x_1 - x)/(fn(x_1)-fn(x)));
    x = x_1;
    x_1 = xx;
end
end
```

Figure 3: Root Finding via Secant Method

Homework 2 Problem 3

AE370 - Spring 2018

Emilio R. Gordon

Problem: Solve the ODE

$$\dddot{x}(t) + 9\ddot{x}(t) + 26\dot{x}(t) + 24x(t) = 7\sin 3t$$

over the interval $0 \leq t \leq 8$ and subject to the initial conditions

$$x(0) = 0.2, \qquad \dot{x}(0) = 0, \qquad \ddot{x}(0) = 0$$

- First use the forward Euler method with a time step of 0.01 s
- Then use backward Euler, again with a step of 0.01 s. Solve the algebraic equations at each step using fixed-point iteration, stopping when the norm of the change in y_k^i in one subiteration is reduced to at most 10^{-6}. Begin the iteration with the result of a Forward Euler step
- Plot both numerical solutions along with the analytical solution,

  $$y_a(t) = 1.02e^{-4t} - 2.76667e^{-3t} + 2.00769e^{-2t} - 0.0610256\cos(3t) - 0.0682051\sin(3t)$$

  and note any difference between them.

Solution: Using the algorithms provided in class and the partial Matlab code provided from the assignment, the problem can be solved. Before going into the solution, it is important to state the function for calculating the derivative given in the assignment as it is used in both, the Forward-Euler function and the Backward-Euler function.

```
function yp = yprime(t, y)
 %% Evaluate the derivative y'(t, y).
 yp = zeros(3, 1);
 yp(1) = y(2);
 yp(2) = y(3);
 yp(3) = − 24.0 ∗ y(1) − 26.0 ∗ y(2) − 9.0 ∗ y(3) + 7.0 ∗ sin(3.0 ∗ t);
end
```

Figure 4: Function for Calculating the Derivative

Forward-Euler Scheme: Part 1

Recall that for the Forward Euler Scheme, we denote h to be the time step size. Using Taylor Series Expansion for y(t+h) we have the expression:

$$y(t+h) = y(t) + h\,\frac{\mathrm{d}y(t)}{\mathrm{d}t}$$

which results in the following iterative scheme:

- Start from t_0 and $y(t_0) = y_0$
- $y(h) = y_1 = y_0 + h\,f(t_0, y_0)$
- $y(2h) = y_2 = y_1 + h\,f(t_1, y_1)$
  ...
- $y((k+1)h) = y_{k+1} = y_k + h\,f(t_k, y_k)$

```
function y = forward_euler(t, y_0, yp, h)
 %% Integrate the ODE given by yp using the forward Euler method.
 y = zeros(length(y_0), length(t)); %Create empty matrix to store all Y's
 y(:, 1) = y_0; %Intitial Condition
 for k = 1 : length(t) - 1
     y(:, k + 1) = y(:, k)+h*yp(t(k), y(:, k));
 end
end
```

Figure 5: Forward Euler Function

Note: I added an additional parameter to the function, h. This allows me to declare the step-size for both functions rather than stating it within the function. This is especially important for testing for convergence and examining results under different step sizes though was not necessary for this assignment.

Backward-Euler Scheme: Part 2

Recall that the Backward-Euler Scheme was an implicit scheme meaning we use information from a later state and then work backwards. This approach is more complex and more time consuming but is more stable than a Forward-Euler approach. The Backward-Euler Scheme is expressed by:

$$y(k+1) = y_k + h\, f(t_{k+1}, y_{k+1})$$

We are also asked to solve each algebraic equation at each step using the fixed point iteration method which means (from slide 81)

Use Forward Euler for initial guess

$$y_{k+1}^{(0)} = y_k + h\, f(t_k, y_k)$$

Then iterate using the fixed-point scheme until convergence

$$y_{k+1}^{(i)} = y_k + h\, f(t_{k+1}, y_{k+1}^{(i-1)})$$

Once convergence is achieved, move to t_{k+2}.

Note, that we will define convergence to be when,

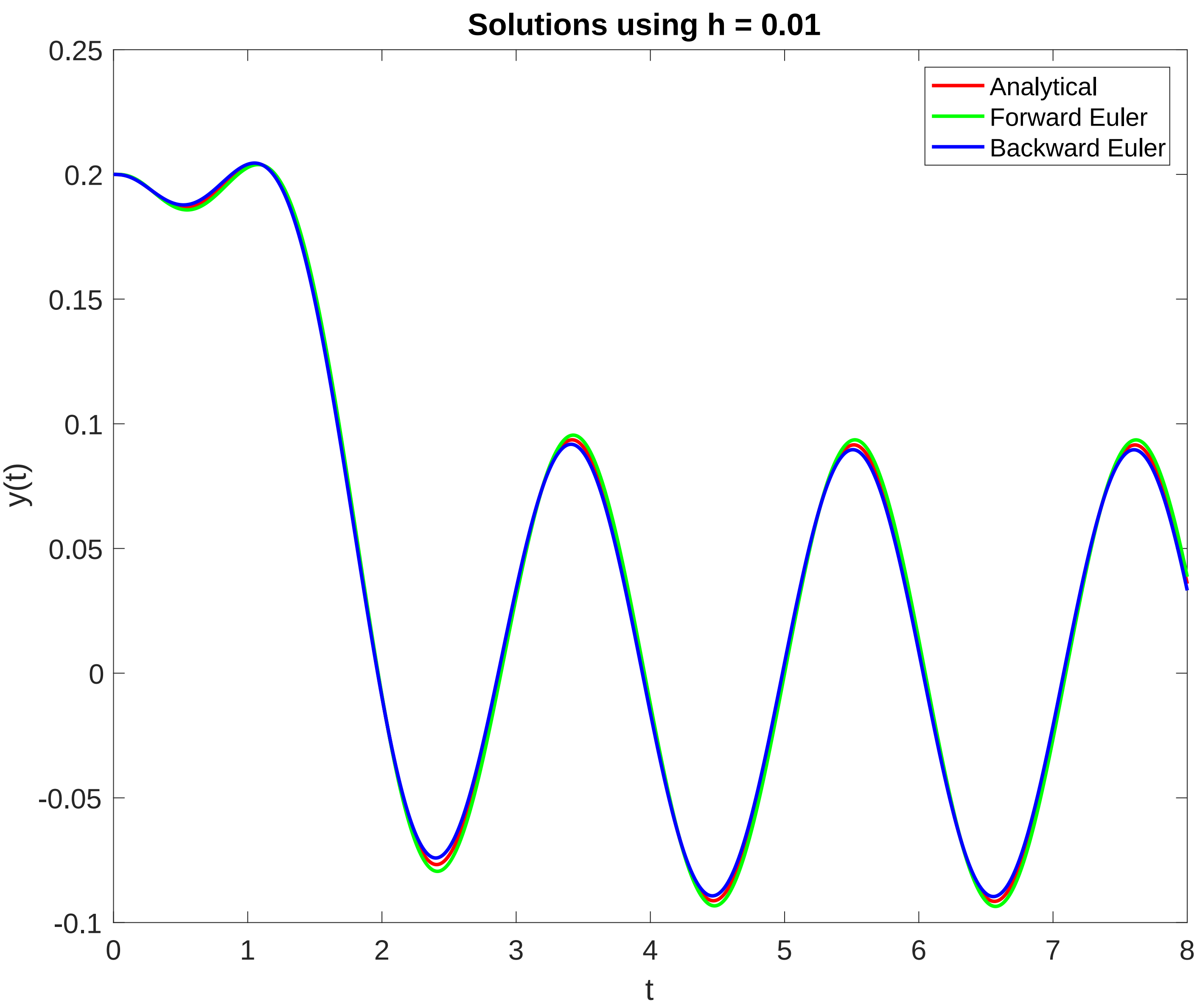
$$||y_{k+1}^{(i)} - y_{k+1}^{(i-1)})|| \leq 10^{-6}$$

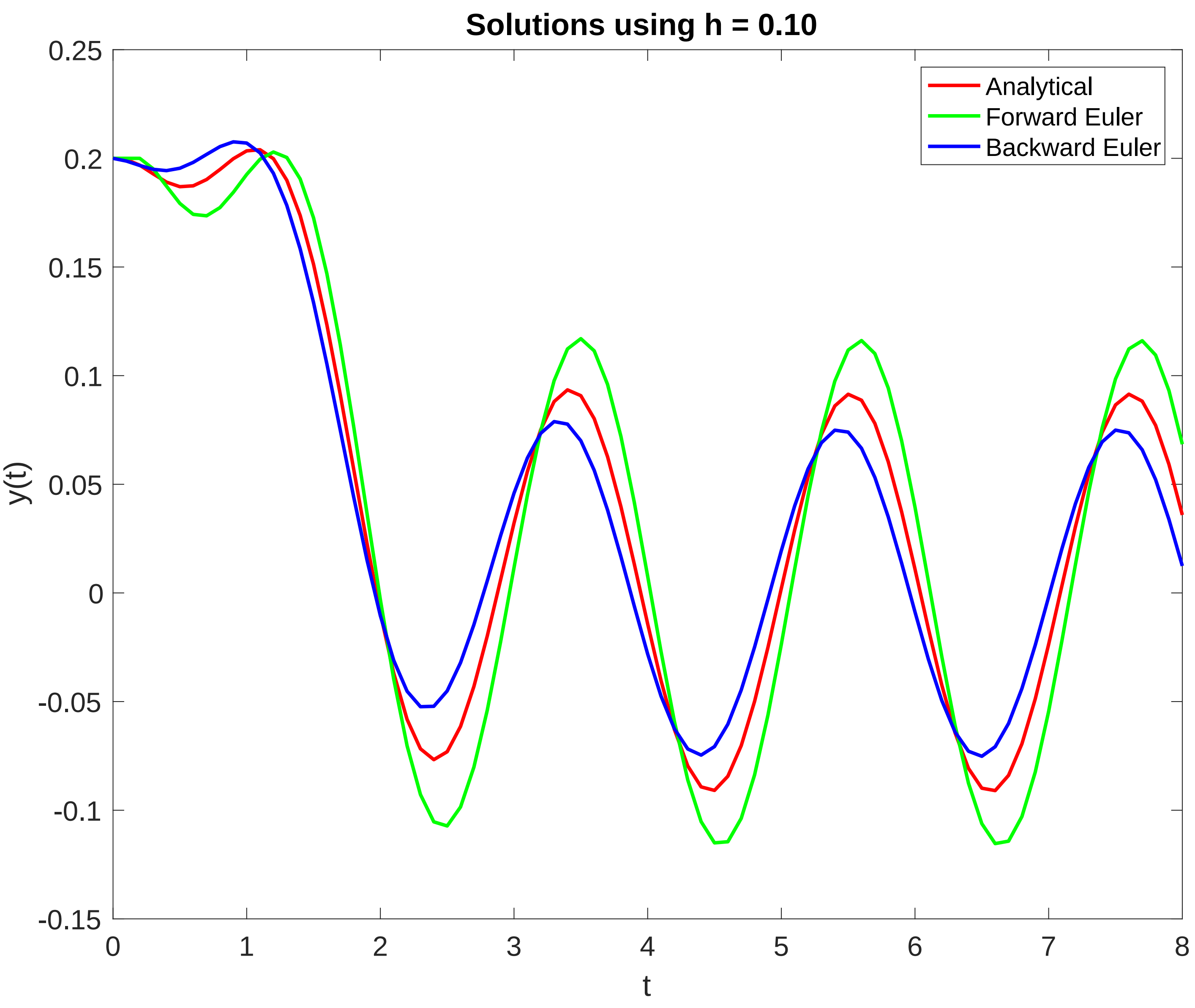
as mentioned in the problem statement.

```matlab
function y = backward_euler(t, y_0, yp,h)
 %% Integrate the ODE given by yp using the backward Euler method.
 tol = 1.0e-6;
 y = zeros(length(y_0), length(t)); % Create empty matrix to store all Y's
 y(:, 1) = y_0;                     % Initial Condition
 for k = 1 : length(t) - 1
     y_k = y(:, k);                    % y_k
     y_kp1_im1 = zeros(size(y_k));     % y_{k+1}^{i-1}
     y_kp1_i = y(:, k) + h * yp(t(k), y(:, k));   % y_{k+1}^{i}
     while norm(y_kp1_i - y_kp1_im1) > tol            % Test for Convergence
         y_kp1_im1 = y_kp1_i;                         % Update y_{k+1}^{i-1}
         y_kp1_i = y_k + h * yp(t(k + 1), y_kp1_im1); % Update y_{k+1}^{i}
     end
     y(:, k + 1) = y_kp1_i;
     end
end
```

Figure 6: Backward Euler Function

The resulting graphs from the Forward Euler and Backward Euler are matched with the analytical solutions shown below. I decided to produce two plots, one of h=0.01 and another with h=0.1 to show how the step-size greatly affects the solutions.




Solutions using h = 0.10
Analytical
Forward Euler
Backward Euler
y(t)
t
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
0
1
2
3
4
5
6
7
8