

**Problem 1**

Find the solution to the following initial value problem.

$$\frac{d^2u}{dt^2} + 2u = 0 \quad \text{for } t \in [0, 10] \quad \text{with } u(0) = 1 \quad \text{and} \quad \frac{du}{dt}(0) = 0 \quad (1)$$

The analytical solution, as provided in class, is...

$$u(t) = \cos(\sqrt{2}t) \quad (2)$$

$$u'(t) = -\sqrt{2}\sin(\sqrt{2}t) \quad (3)$$

Before starting on the different finite difference methods, the initial conditions must be properly constructed.

With  $u(0) = 1$  and  $\frac{du}{dt}(0) = 0$ , let

$$u_1 = u(t) \quad \text{and} \quad u_2 = u'(t)$$

To establish the function,  $F(t) = \frac{du}{dt}$ , that will be used in the finite difference methods, the equation  $\frac{d^2u}{dt^2} + 2u = 0$  is used to determine...

$$u'_1 = u_2 \quad \text{and} \quad u'_2 = -2u_1 \quad \text{because } u''(t) = -2u(t)$$

$$\frac{d}{dt} \begin{bmatrix} u(t) \\ u'(t) \end{bmatrix} = \frac{d}{dt} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} u_2 \\ -2u_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -2 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = F(t)$$

With  $u(0) = 1$  and  $\frac{du}{dt}(0) = 0$ , the initial condition at  $t = 0$  is...

$$u^{i=0} = \begin{bmatrix} u_1 = u_1(0) \\ u_2 = u_2(0) \end{bmatrix}^{i=0} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}^{i=0}$$

1. **Forward Euler:**

The Forward Euler Method is defined as...

$$F(u^i) = \frac{u^{i+1} - u^i}{\Delta t} \longrightarrow u^{i+1} = u^i + \Delta t F(u^i) \quad (4)$$

Rewritten as a system of matrices to be iterated over each time-step,  $\Delta t$ ...

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^{i+1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^i + \begin{bmatrix} 0 & \Delta t \\ -2\Delta t & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^i \longrightarrow \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^{i+1} = \begin{bmatrix} 1 & \Delta t \\ -2\Delta t & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^i$$

This process is displayed in the function, forward\_euler:

```

1 function [U, Analytic, Error] = forward_euler(dt, time)
2     steps = time/dt;
3     t_vec = [0 : dt : time];
4     U = zeros(2, steps+1);
5     Analytic = zeros(2, steps+1);
6
7     % Set initial condition
8     U(:, 1) = [1 ; 0];
9
10    % finite difference Matrix
11    F = [1      dt ; ...
12         -2*dt   1];
13
14    % loop over all time steps
15    for ii=2:steps
16        U(:, ii) = F * U(:, ii-1);
17    end
18
19    Analytic(1,:) = analytic(t_vec);
20    Analytic(2,:) = analyticdt(t_vec);
21
22    Error = abs( U(1, (end+1)/2) - Analytic(1, (end+1)/2) );
23 end

```

forward\_euler

For this assignment, I decided to use the time-steps...

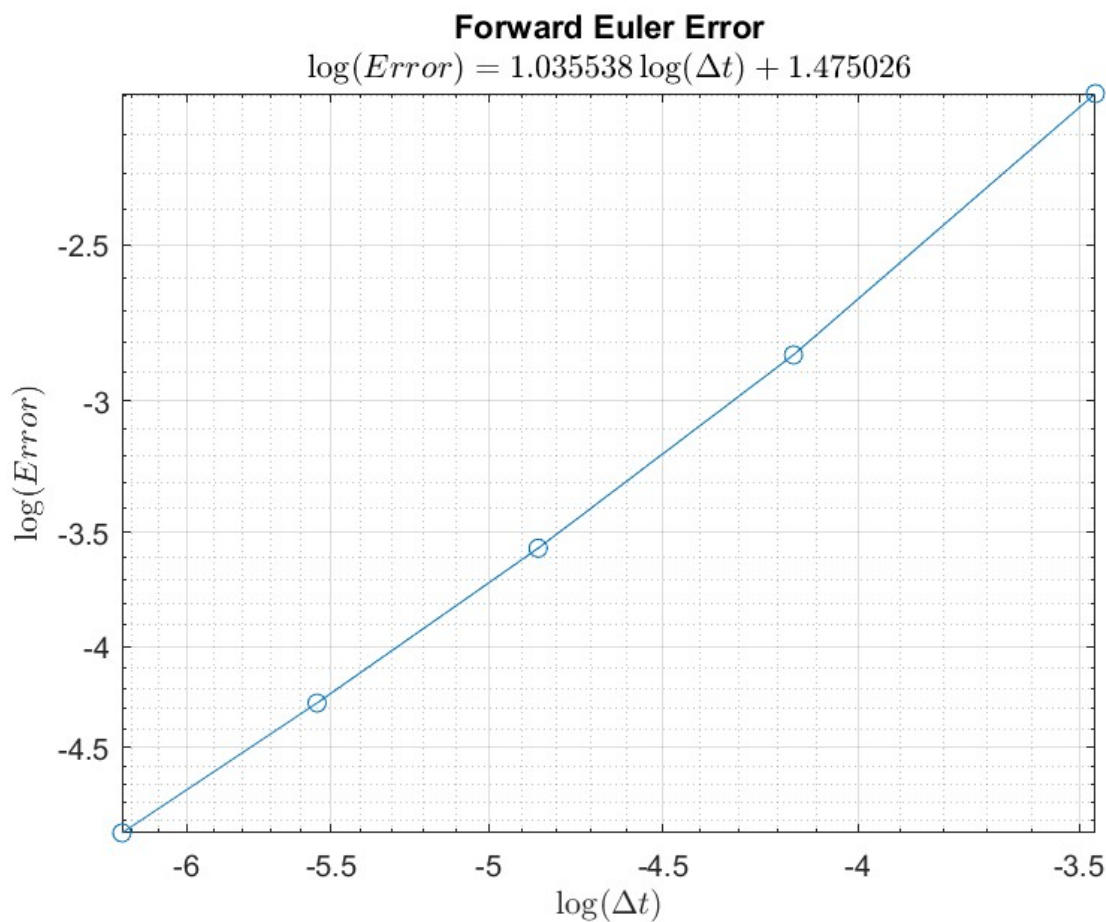
$$\Delta t = \left[ \left(\frac{1}{2}\right)^5, \left(\frac{1}{2}\right)^6, \left(\frac{1}{2}\right)^7, \left(\frac{1}{2}\right)^8, \left(\frac{1}{2}\right)^9 \right]$$

in order to make the Forward Euler method sufficiently stable.

Graphs comparing  $\frac{du}{dt}$  vs  $u$  using the different finite difference methods are located towards the end of the paper.

The Forward Euler Method tends to trail off from the analytical result as the error in each progression through time compounds on itself. In a way it adds more and more error, spiraling it away from the true answer.

A log-log plot of the error compared with the different time-steps,  $\Delta t$ , is...



The slope of the log-log plot regression properly shows the order of accuracy for the Forward Euler method, which is one. This is due to this Forward Euler Method having a schematic error of  $\mathcal{O}(\Delta t)$  and a truncation error of  $\mathcal{O}(\Delta t^2)$

## 2. Backward Euler:

The Backwards Euler Method is defined as...

$$F(u^i) = \frac{u^{i+1} - u^i}{\Delta t} \longrightarrow u^{i+1} = u^i + \Delta t F(u^{i+1}) \quad (5)$$

Rewritten as a system of matrices to be iterated over each time-step,  $\Delta t$ ...

$$\begin{aligned} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^{i+1} &= \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^i + \begin{bmatrix} 0 & \Delta t \\ -2\Delta t & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^{i+1} \longrightarrow \begin{bmatrix} -1 & \Delta t \\ 2\Delta t & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^{i+1} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^i \\ &\longrightarrow \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^{i+1} = \begin{bmatrix} -1 & \Delta t \\ 2\Delta t & 1 \end{bmatrix}^{-1} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^i \end{aligned}$$

This process is displayed in the function, backward\_euler:

```

1 function [U, Error] = backward_euler(dt, time)
2     steps = time/dt;
3     t_vec = [0 : dt : time];
4     U = zeros(2, steps+1);
5     Analytic = U;
6
7     % Set initial condition
8     U(:, 1) = [1 ; 0];
9
10    % finite difference Matrix
11    F = [1      -dt; ...
12         2*dt    1];
13
14    % loop over all time steps
15    for ii=2:steps
16        U(:, ii) = inv(F)*U(:, ii-1);
17    end
18
19    Analytic(1,:) = analytic(t_vec);
20    Analytic(2,:) = analyticdt(t_vec);
21
22    Error = abs( U(1,(end+1)/2) - Analytic(1,(end+1)/2) );
23 end

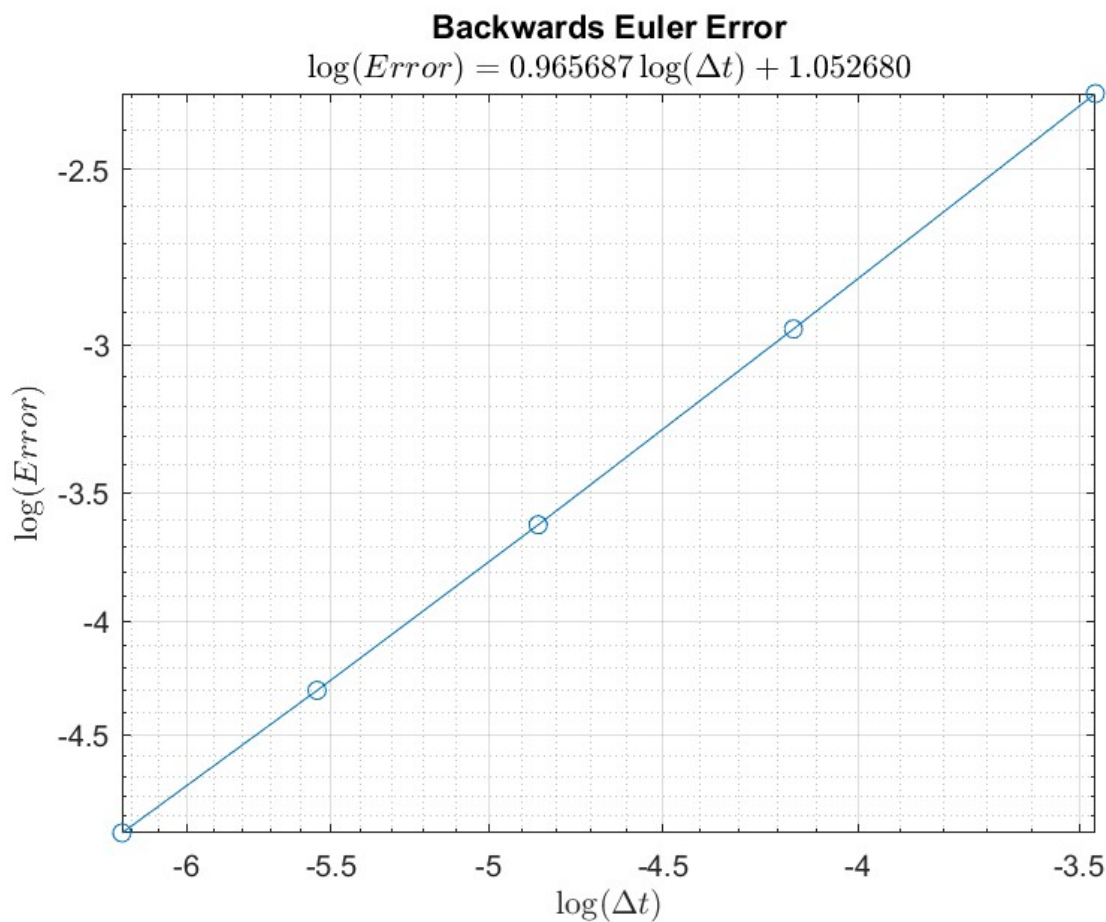
```

backward\_euler

Graphs comparing  $\frac{du}{dt}$  vs  $u$  using the different finite difference methods are located towards the end of the paper.

The Backwards Euler Method tends to swirl inside the analytical result as the error in each progression through time draws itself closer to zero. The error kind of subtracts itself further from the correct answer.

A log-log plot of the error compared with the different time-steps,  $\Delta t$ , is...



The slope of the log-log plot regression properly shows the order of accuracy for the Backwards Euler method, which is one. This is due to this Backwards Euler Method having a schematic error of  $\mathcal{O}(\Delta t)$  and a truncation error of  $\mathcal{O}(\Delta t^2)$

**3. The Trapezoidal method:**

The Trapezoidal Method is defined as...

$$u^{i+1} = u^i + \frac{\Delta t}{2} (F(u^i) + F(u^{i+1})) \quad (6)$$

Rewritten as a system of matrices to be iterated over each time-step,  $\Delta t$ ...

$$\begin{aligned} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^{i+1} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^i + \begin{bmatrix} 0 & \frac{\Delta t}{2} \\ -\Delta t & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^i + \begin{bmatrix} 0 & \frac{\Delta t}{2} \\ -\Delta t & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^{i+1} \\ \longrightarrow \begin{bmatrix} 1 & -\frac{\Delta t}{2} \\ \Delta t & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^{i+1} &= \begin{bmatrix} 1 & \frac{\Delta t}{2} \\ -\Delta t & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^i \\ \longrightarrow \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^{i+1} &= \begin{bmatrix} 1 & -\frac{\Delta t}{2} \\ \Delta t & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & \frac{\Delta t}{2} \\ -\Delta t & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^i \end{aligned}$$

This process is displayed in the function, trapezoid:

```

1 function [U, Error] = trapezoid(dt, time)
2     steps = time/dt;
3     t_vec = [0 : dt : time];
4     U = zeros(2, steps+1);
5     Analytic = U;
6
7     % Set initial condition
8     U(:, 1) = [1 ; 0];
9
10    % finite difference Matrices
11    F1 = [1      dt/2; ...
12         -dt     1];
13    F2 = [1      -dt/2; ...
14         dt      1];
15
16    % loop over all time steps
17    for ii=2:steps
18        U(:, ii) = inv(F2)*F1*U(:, ii-1);
19    end
20
21    Analytic(1,:) = analytic(t_vec);
22    Analytic(2,:) = analyticdt(t_vec);
23
24    Error = abs( U(1, (end+1)/2) - Analytic(1, (end+1)/2) );
25 end

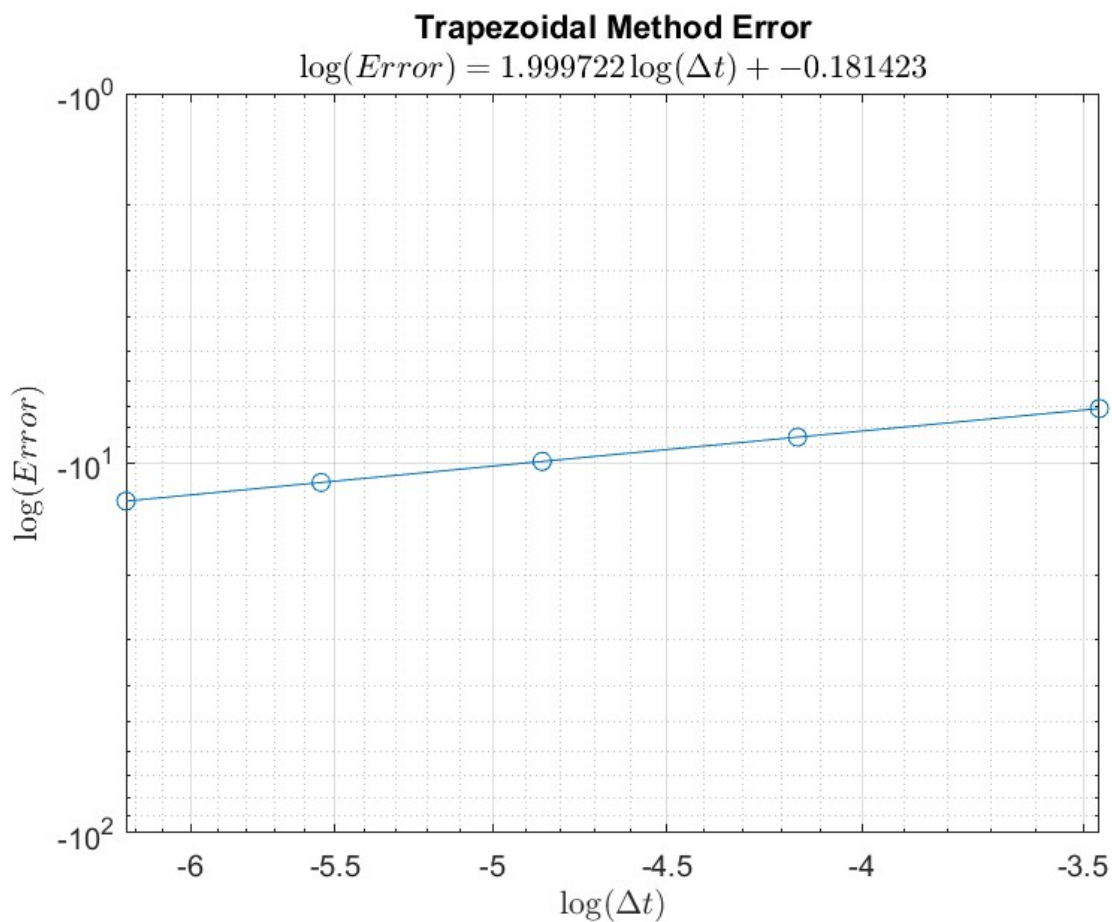
```

backward\_euler

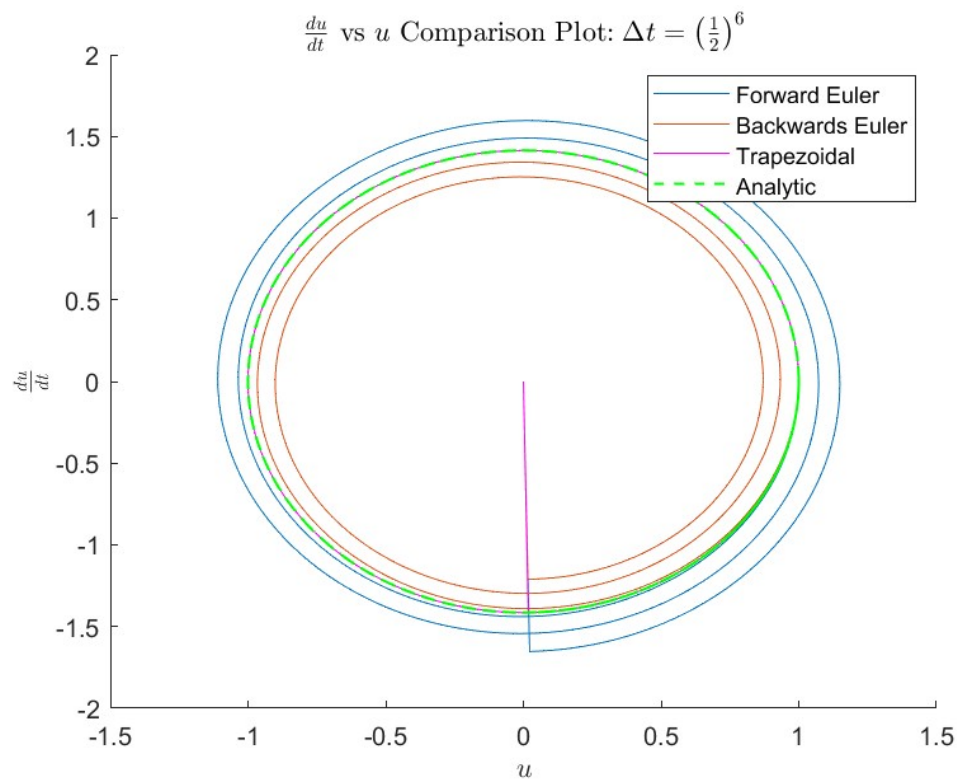
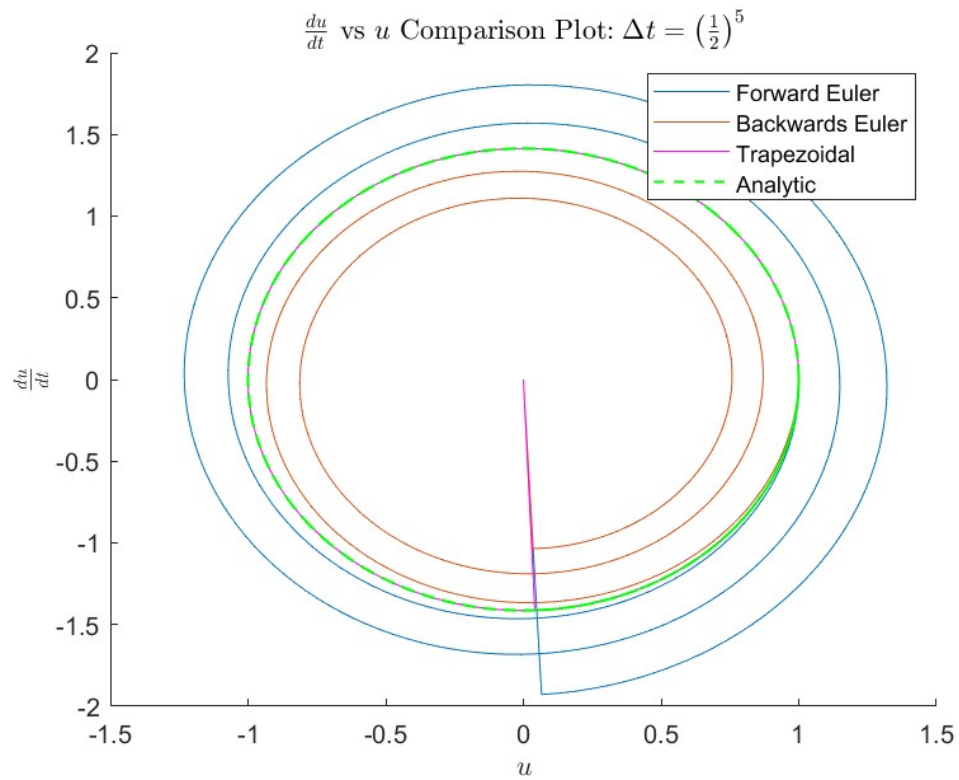
Graphs comparing  $\frac{du}{dt}$  vs  $u$  using the different finite difference methods are located towards the end of the paper.

The Trapezoidal Method tends to track well with the analytical answer. Being an average of the Forward and Backwards Euler Methods, the error of the Forward Euler Method kind of cancels out the error of the Backwards Euler method. However, that is not to mean that there isn't any error, which is discussed below.

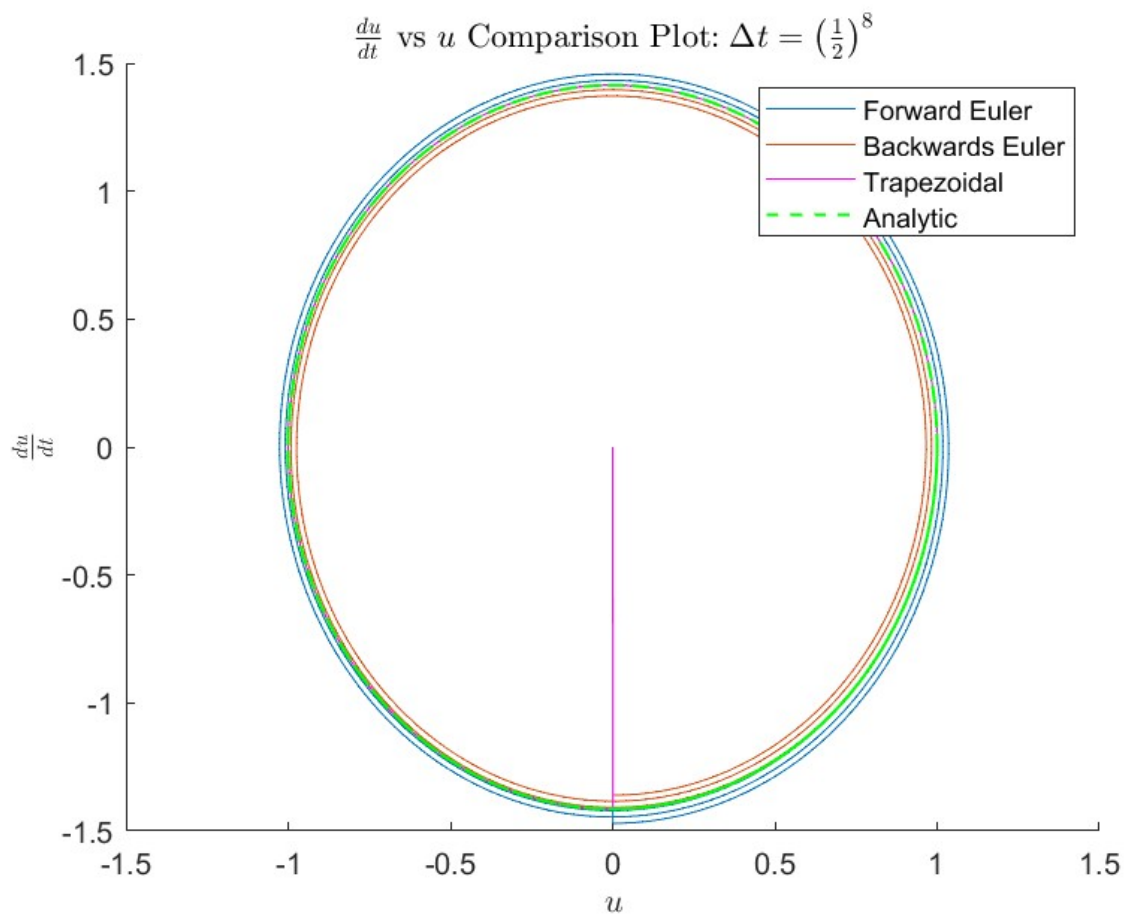
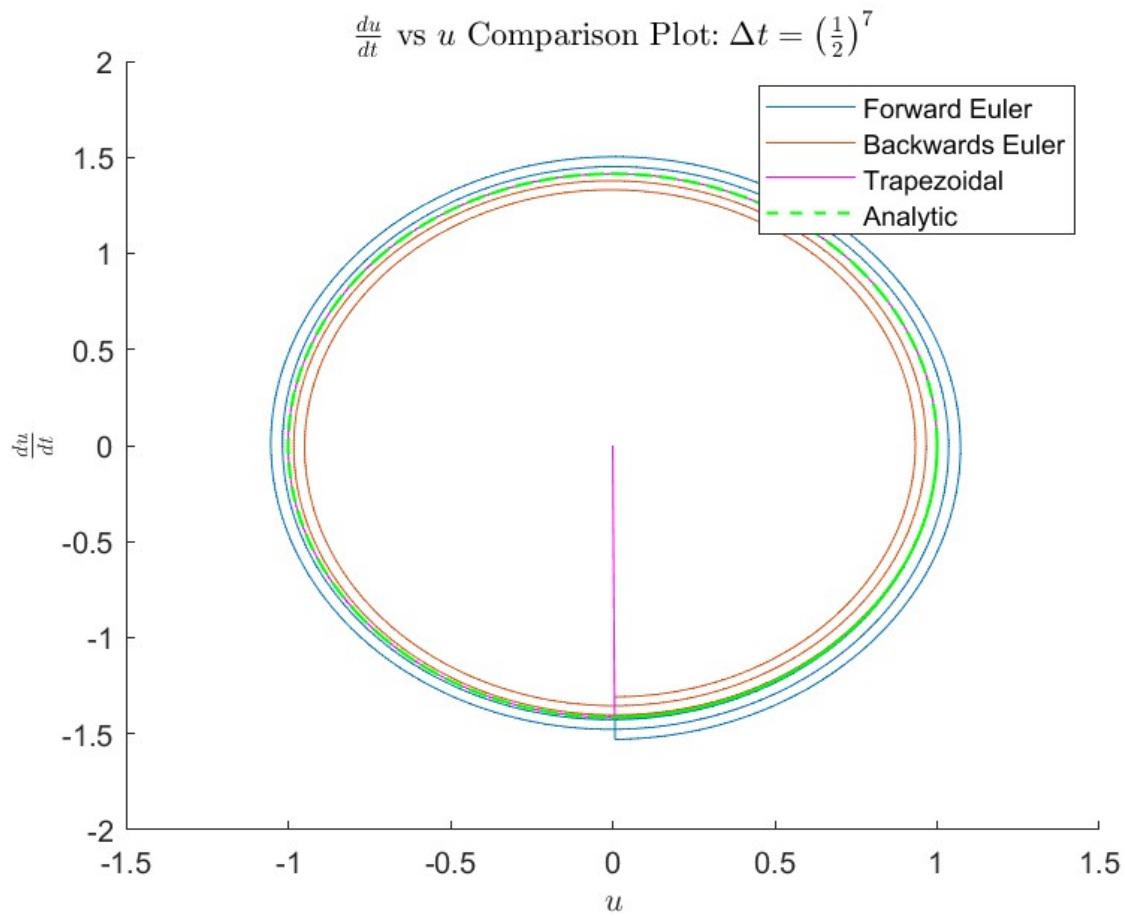
A log-log plot of the error compared with the different time-steps,  $\Delta t$ , is...

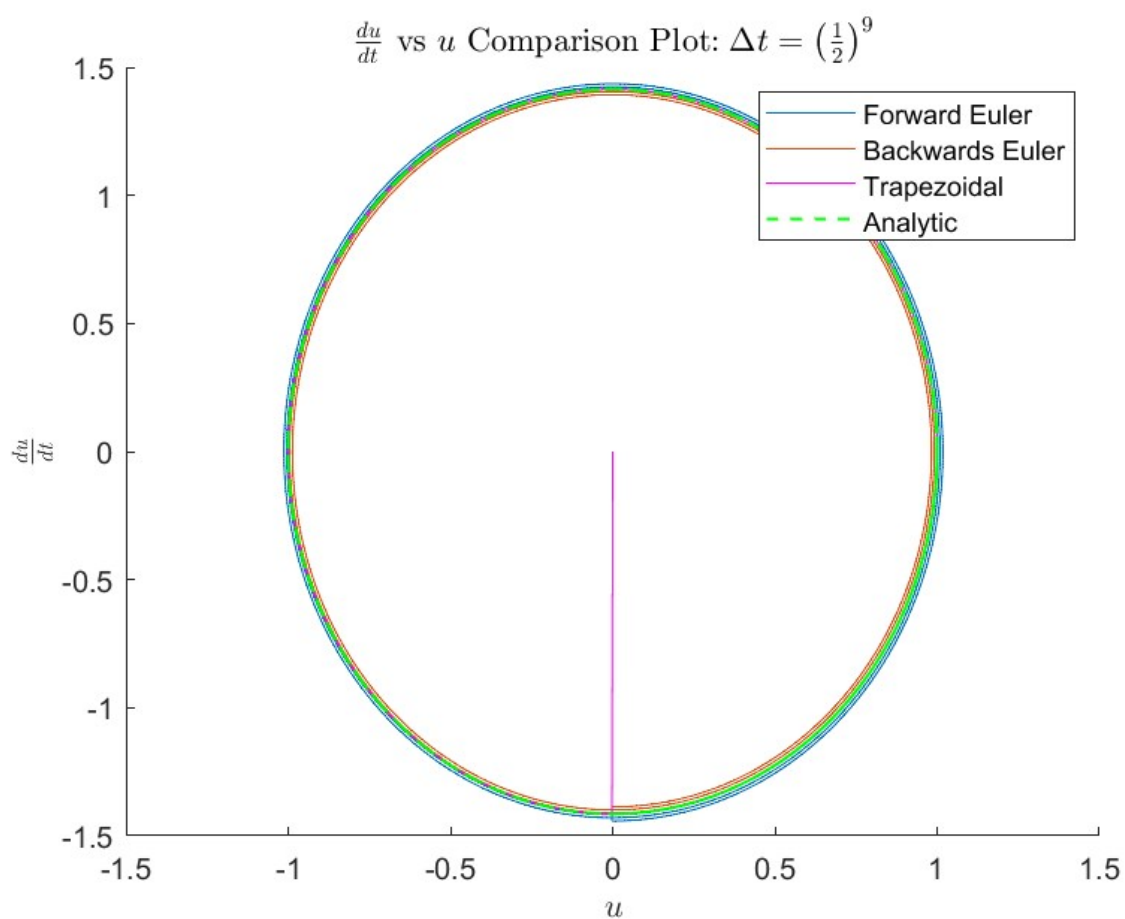


The slope of the log-log plot regression properly shows the order of accuracy for the Trapezoidal Method, which is two. This is due to this Trapezoidal Method having a schematic error of  $\mathcal{O}(\Delta t^2)$

$\frac{du}{dt}$  vs  $u$  Comparison Plots







## Matlab Code

```

1 %% Zack Humphries
2 % COMP 521
3 % HW6
4
5 close all;
6 clear;
7 clc;
8
9 % Analytical solution == u(t) = cos(sqrt(2) * t)
10 % u(0) = 1
11 % u'(t) = 0
12
13 time = 10;
14 expo = [5 6 7 8 9];
15 time_steps = 1./2.^expo;
16
17 %%
18 forward_euler_error = zeros(length(time_steps),1);
19 backward_euler_error = zeros(length(time_steps),1);
20 trapezoid_error = zeros(length(time_steps),1);
21
22 for kk = 1:length(time_steps)
23
24     dt = time_steps(kk);
25     t_vec = [0 : dt : time];
26     [forward_euler_U, Analytic, Error_forward] = forward_euler(dt, time);
27     forward_euler_error(kk,1) = Error_forward;
28
29     [backward_euler_U, Error_backward] = backward_euler(dt, time);
30     backward_euler_error(kk,1) = Error_backward;
31
32     [trapezoid_U, Error_trapezoid] = trapezoid(dt, time);
33     trapezoid_error(kk,1) = Error_trapezoid;
34
35     figure
36     hold on
37     plot(forward_euler_U(1,:), forward_euler_U(2,:));
38     plot(backward_euler_U(1,:), backward_euler_U(2,:));
39     plot(trapezoid_U(1,:), trapezoid_U(2,:), '-m');
40     plot(Analytic(1,:), Analytic(2,:), '--g', "LineWidth",1);
41     title_name = strcat("\frac{du}{dt}$ vs $u$ Comparison Plot$: \Delta t = \left(\frac{1}{2}\right)^{" , sprintf("%i", expo(kk)), "}");
42     title(title_name, 'interpreter', 'latex')
43     legend("Forward Euler", "Backwards Euler", "Trapezoidal", "Analytic")
44     xlabel("$u$", 'interpreter', 'latex')
45     ylabel("\frac{du}{dt}$", 'interpreter', 'latex')
46     % add a legend, label, etc, etc
47     hold off
48 end
49
50 %% make plot
51
52 % Error plotting

```

```

53 figure
54 forward_poly = polyfit(log(time_steps), log(forward_euler_error), 1);
55 loglog(log(time_steps), log(forward_euler_error), "o-"); grid on;
56 title("Forward Euler Error")
57 subtitle_name_forward = strcat("$\log(\text{Error}) = ", sprintf("%2.6f",
    forward_poly(1)), "\log(\Delta t) + ", sprintf("%2.6f", forward_poly(2)),
    "$");
58 subtitle(subtitle_name_forward, 'interpreter', 'latex')
59 xlabel("$\log(\Delta t)$", 'interpreter', 'latex')
60 ylabel("$\log(\text{Error})$", 'interpreter', 'latex')
61 figure
62 backward_poly = polyfit(log(time_steps), log(backward_euler_error), 1);
63 loglog(log(time_steps), log(backward_euler_error), "o-"); grid on;
64 title("Backwards Euler Error")
65 subtitle_name_backward = strcat("$\log(\text{Error}) = ", sprintf("%2.6f",
    backward_poly(1)), "\log(\Delta t) + ", sprintf("%2.6f", backward_poly(2)),
    "$");
66 subtitle(subtitle_name_backward, 'interpreter', 'latex')
67 xlabel("$\log(\Delta t)$", 'interpreter', 'latex')
68 ylabel("$\log(\text{Error})$", 'interpreter', 'latex')
69 figure
70 trapezoid_poly = polyfit(log(time_steps), log(trapezoid_error), 1);
71 loglog(log(time_steps), log(trapezoid_error), "o-"); grid on;
72 title("Trapezoidal Method Error")
73 subtitle_name_trapezoid = strcat("$\log(\text{Error}) = ", sprintf("%2.6f",
    trapezoid_poly(1)), "\log(\Delta t) + ", sprintf("%2.6f", trapezoid_poly(2)),
    "$");
74 subtitle(subtitle_name_trapezoid, 'interpreter', 'latex')
75 xlabel("$\log(\Delta t)$", 'interpreter', 'latex')
76 ylabel("$\log(\text{Error})$", 'interpreter', 'latex')
77
78 % HW asks for global trunc error and last time step
79
80 %% Forward Euler Function
81
82 function [U, Analytic, Error] = forward_euler(dt, time)
83     steps = time/dt;
84     t_vec = [0 : dt : time];
85     U = zeros(2, steps+1);
86     Analytic = zeros(2, steps+1);
87
88     % Set initial condition
89     U(:, 1) = [1 ; 0];
90
91     % finite difference Matrix
92     F = [1      dt; ...
93         -2*dt   1];
94
95     % loop over all time steps
96     for ii=2:steps
97         U(:, ii) = F * U(:, ii-1);
98     end
99
100     Analytic (1,:) = analytic(t_vec);
101     Analytic (2,:) = analyticdt(t_vec);

```

```

102     Error = abs( U(1,(end+1)/2) - Analytic(1,(end+1)/2) );
103
104 end
105
106 %% Backwards Euler Function
107
108 function [U, Error] = backward_euler(dt, time)
109     steps = time/dt;
110     t_vec = [0 : dt : time];
111     U = zeros(2,steps+1);
112     Analytic = U;
113
114     % Set initial condition
115     U(:, 1) = [1 ; 0];
116
117     % finite difference Matrix
118     F = [1      -dt;...
119          2*dt    1];
120
121     % loop over all time steps
122     for ii=2:steps
123         U(:, ii) = inv(F)*U(:, ii-1);
124     end
125
126     Analytic (1,:) = analytic(t_vec);
127     Analytic (2,:) = analyticdt(t_vec);
128
129     Error = abs( U(1,(end+1)/2) - Analytic(1,(end+1)/2) );
130 end
131
132 %% Trapezoid Function
133
134 function [U, Error] = trapezoid(dt, time)
135     steps = time/dt;
136     t_vec = [0 : dt : time];
137     U = zeros(2,steps+1);
138     Analytic = U;
139
140     % Set initial condition
141     U(:, 1) = [1 ; 0];
142
143     % finite difference Matrices
144     F1 = [1      dt/2;...
145          -dt      1];
146     F2 = [1      -dt/2;...
147          dt      1];
148
149     % loop over all time steps
150     for ii=2:steps
151         U(:, ii) = inv(F2)*F1*U(:, ii-1);
152     end
153
154     Analytic (1,:) = analytic(t_vec);
155     Analytic (2,:) = analyticdt(t_vec);
156

```

```
157     Error = abs( U(1,(end+1)/2) - Analytic(1,(end+1)/2) );
158 end
159
160 function result = analytic(t)
161     s2 = sqrt(2);
162     result = cos(s2 .* t);
163 end
164
165 function result = analyticdt(t)
166     s2 = sqrt(2);
167     result = -s2 .* sin(s2 .* t);
168 end
```