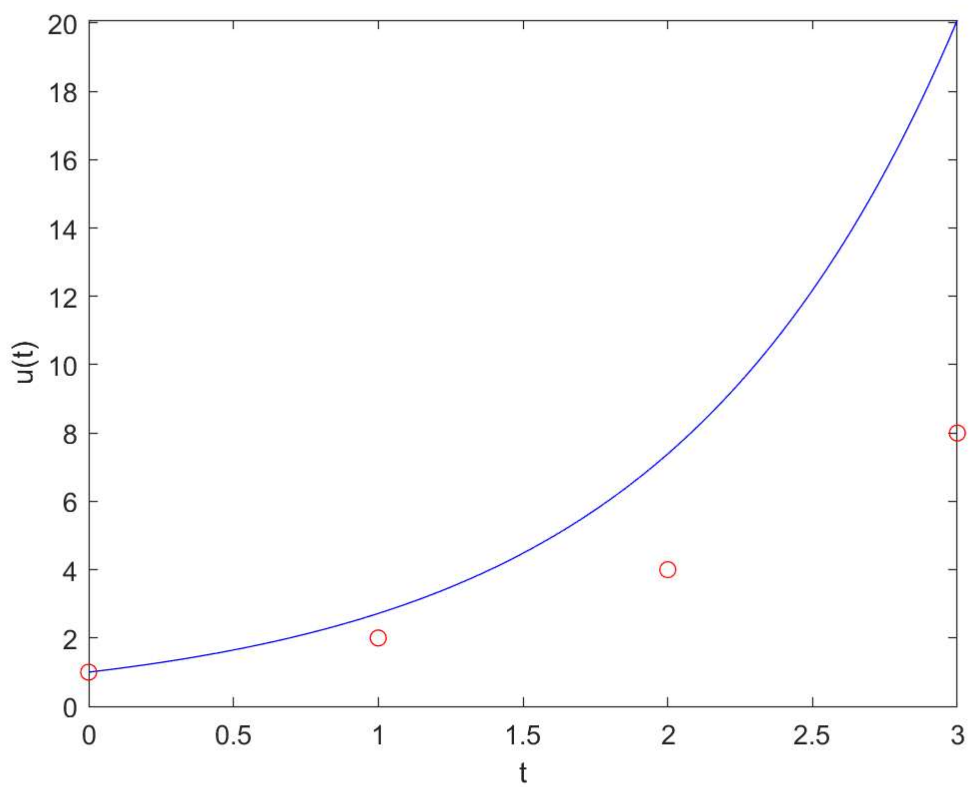
Assignment 1 of 2

4.1 - Geometric construction of the Forward Euler method

```
% Forward Euler for u' = u
a = 0; b = 3;
dudt = @(u) u;
u_exact = @(t) exp(t);

u = zeros(4, 1);
u(1) = 1;
dt = 1;

for i = 1:3
    u(i+1) = u(i) + dt*dudt(u(i));
end


tP = [0 1 2 3];
time = linspace(a, b, 100);
u_true = u_exact(time);
plot(time, u_true, 'b-', tP, u, 'ro');
xlabel('t');
ylabel('u(t)');
```



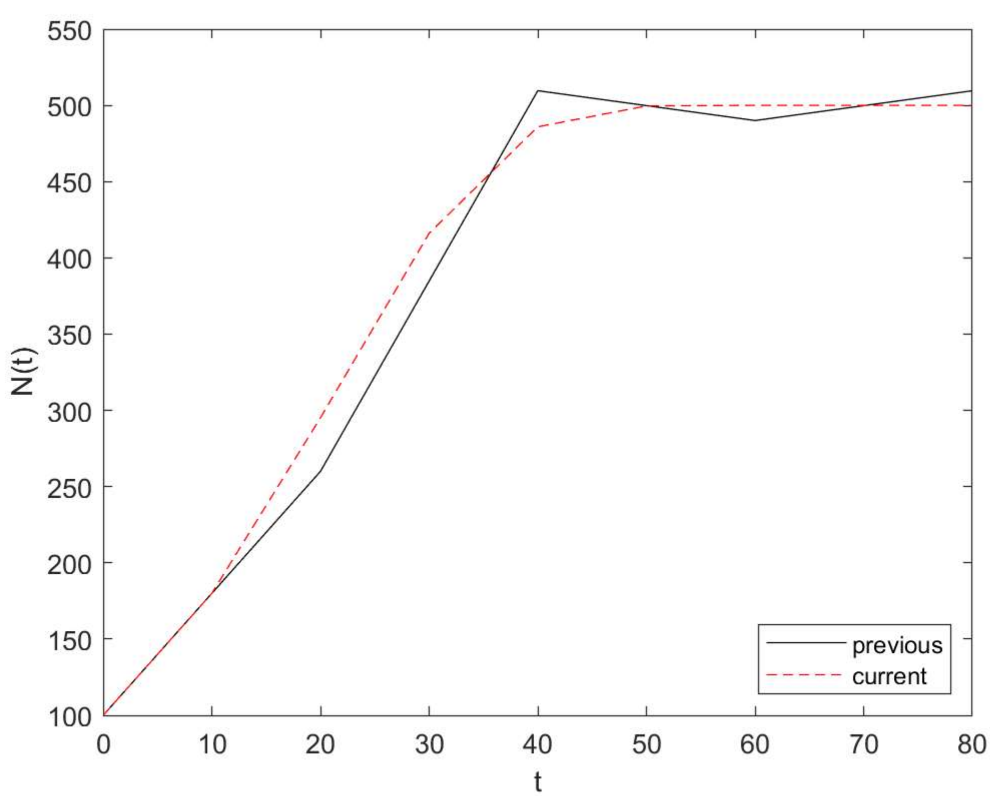
4.2 - Make test functions for the Forward Euler method

```
test_ode_FE_1();
```

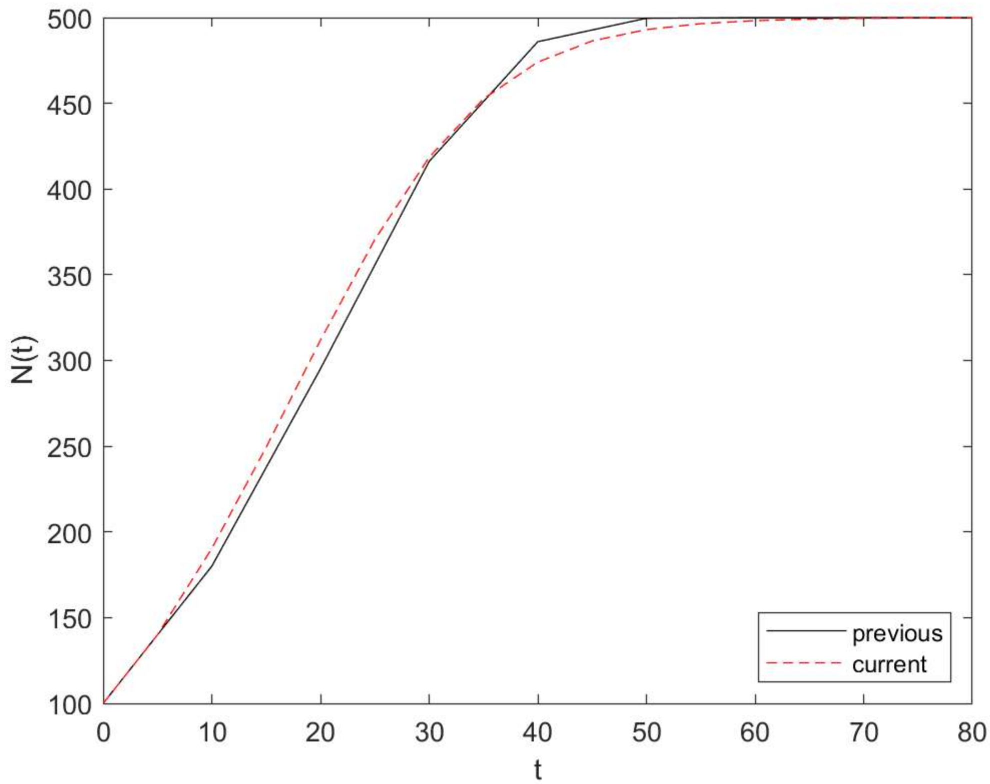
```
test_ode_FE_2();
```

4.4 Find an appropriate time step; logistic model

```
% Repetition of solution from last task
% Extension of logistic.m found on page 99

dt = 20;
T = 80;
f = @(u,t) 0.1*(1-u/500)*u;

U_0 = 100;

[u, t] = ode_FE(f, U_0, dt, T);

k = 1;
while true
    dt_k = 2^(-k)*dt;
    [u_current, t_current] = ode_FE(f, U_0, dt_k, T);
    graph_result(t,u,t_current,u_current, dt_k);
    fprintf("Previous timestep was: %0.3f \n", dt_k)
    if (strcmp(input("Continue with a higher dt value [y/n]? ",'s'),'y'))
       u = u_current;
       t = t_current;
       k = k + 1;
    else
        break; % The interval is okay so we're stopping the loop
    end

end
```



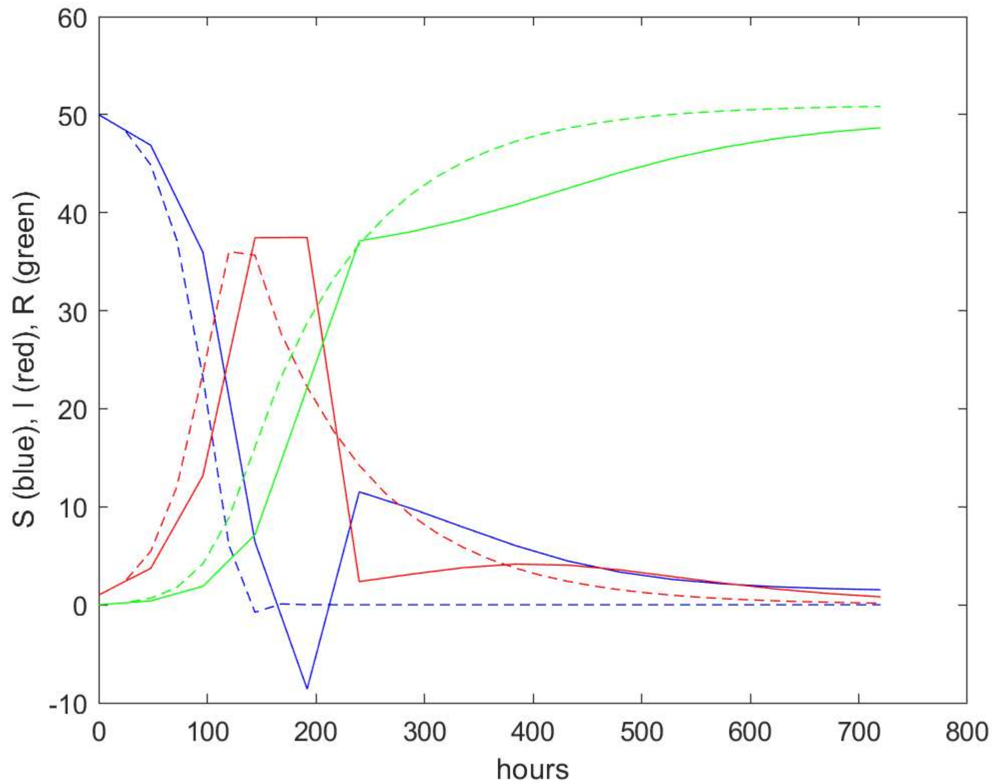
```
Previous timestep was: 10.000
```




```
Previous timestep was: 5.000
```

4.5 - Find an appropriate time step; SIR model

```
demo_SIR
```

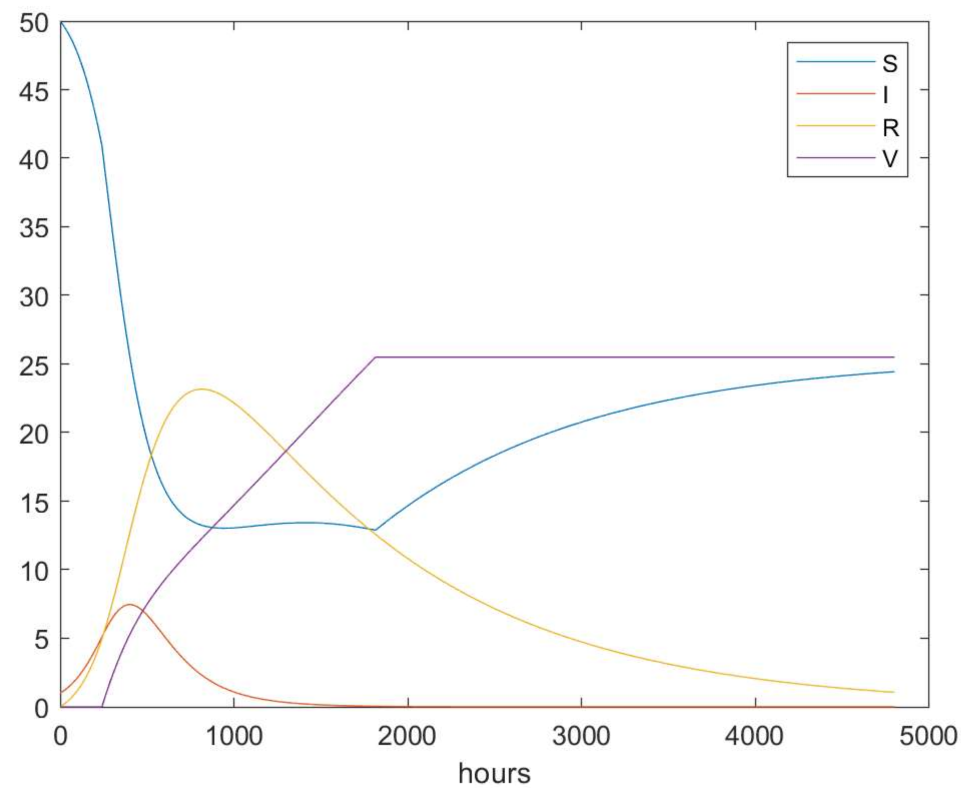



```
Finest timestep was: 24
```

4.6 - Model an adaptive vaccination campaign

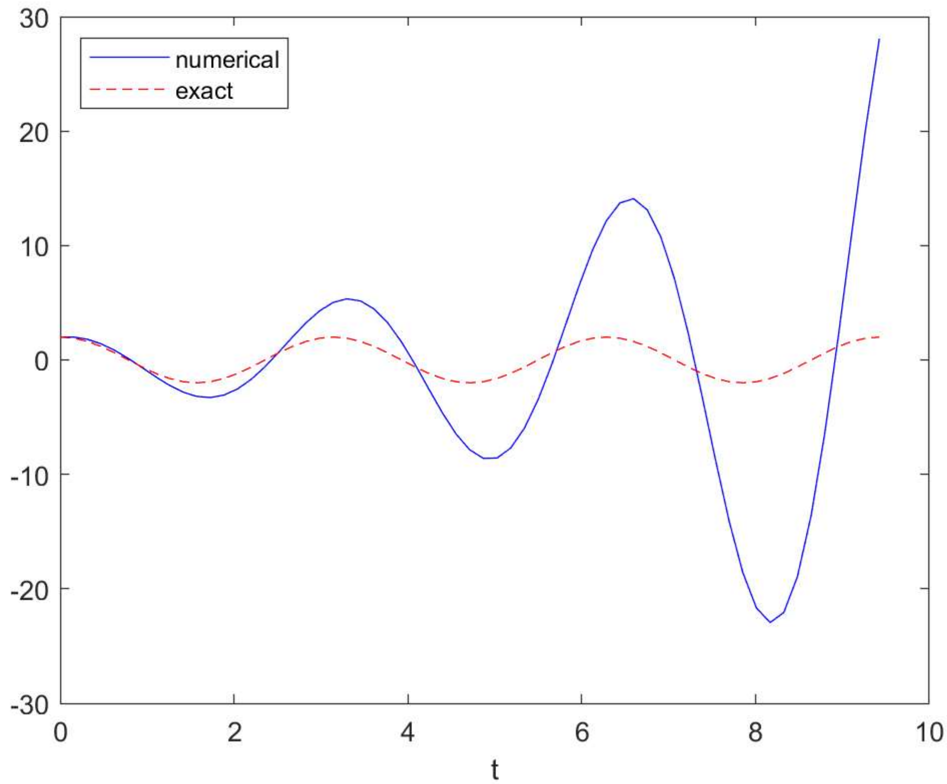
```
SIRV_p_adapt
```

```
beta: 0.000325521
```



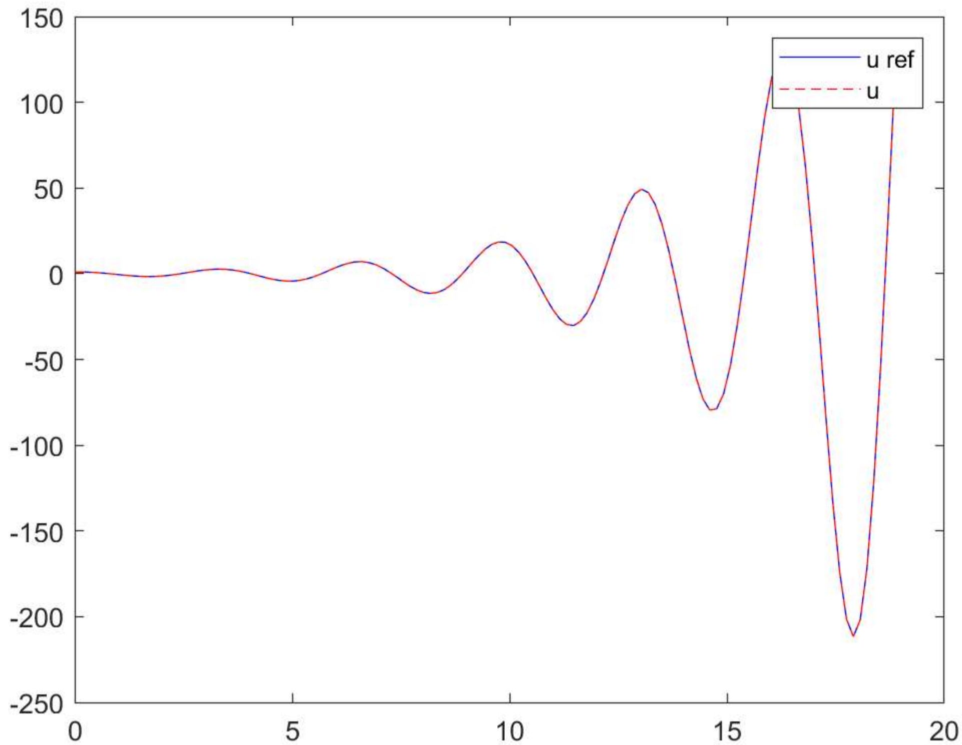
4.8 - Refactor a flat program

```
demo_osc_FE
```



4.9 - Simulate oscillations by a general ODE solver

```
test_osc_ode_FE
```



Utility Graphing Function

```
function graph_result(t,u,t_current, u_current,dt_k)
    figure
    plot(t,u,'k',t_current,u_current,'r--')
    legend('previous','current','Location','southeast')
    xlabel('t');
    ylabel('N(t)');
    saveas(gcf,sprintf("output_logistic_dt_%d.png",dt_k))
end
```

4.2 Tests For Forward Euler Method

```
function test_ode_FE_1()
    function res = f(u, ~)
        res = u;
    end
    hand = [1 2 4 8];
    T = 3;
    dt = 1;
    U_0 = 1.0;
    [u, ~] = ode_FE(@f, U_0, dt, T);
    tol = 1E-14;
    for i = 1:length(hand)
        err = abs(hand(i) - u(i));
        assert(err < tol, 'i=%d, err=%g', i, err);
    end
end
```

```
function test_ode_FE_2()
    function res = f(u, ~)
        r = 1;
        res = r*u;
    end
    function res = u_exact(U_0, dt, n)
        r = 1;
        res = U_0*(1+r*dt)^n;
    end
    T = 3;
    dt = 0.1;       % Tested first dt = 1.0, ok
    U_0 = 1.0;

    [u, ~] = ode_FE(@f, U_0, dt, T);
    tol = 1E-12;     % Relaxed from 1E-14 to get through
    for n = 1:length(u)
        err = abs(u_exact(U_0, dt, n-1) - u(n));
        assert(err < tol, 'n=%d, err=%g', n, err);
    end
end
```

4.5 - Demo SIR Functions

```
function demo_SIR()
    % Find appropriate time step for an SIR model
    function res = f(u,~)
        beta = 10/(40*8*24);
        gamma = 3/(15*24);
        S = u(1); I = u(2); R = u(3);
        res = [-beta*S*I beta*S*I - gamma*I gamma*I];
    end

    dt = 48.0;   % 48 h
    D = 30;      % Simulate for D days
    N_t = floor(D*24/dt);    % Corresponding no of hours
    T = dt*N_t;              % End time
    U_0 = [50 1 0];

    [u_old, t_old] = ode_FE(@f, U_0, dt, T);

    S_old = u_old(:,1);
    I_old = u_old(:,2);
    R_old = u_old(:,3);
    k = 1;
    one_more = true;
    while one_more == true
        dt_k = 2^(-k)*dt;
        [u_new, t_new] = ode_FE(@f, U_0, dt_k, T);
        S_new = u_new(:,1);
        I_new = u_new(:,2);
        R_new = u_new(:,3);

        plot(t_old, S_old, 'b-', t_new, S_new, 'b--',...
```

```
            t_old, I_old, 'r-', t_new, I_new, 'r--',...
            t_old, R_old, 'g-', t_new, R_new, 'g--');
        xlabel('hours');
        ylabel('S (blue), I (red), R (green)');
        fprintf('Finest timestep was: %g\n', dt_k);
        answer = input('Do one more with finer dt (y/n)? ','s');
        if strcmp(answer, 'y')
            S_old = S_new;
            R_old = R_new;
            I_old = I_new;
            t_old = t_new;
            k = k + 1;
        else
            one_more = false;
        end
    end
end
```

4.6 - Model an adaptive vaccination campaign

```
function SIRV_p_adapt()
    % Time-dependent vaccination.
    % Time unit: 1 h

    function res = p(t, n)
        if ( V(n) < 0.5 * ( S(1) + I(1) ) && t > Delta * 24)
            res = p_0;
        else
            res = 0;
        end
    end

    beta = 10 /( 40 * 8 * 24 );
    beta = beta/4;          % Reduce beta compared to SIR1.py
    fprintf('beta: %g\n', beta);
    gamma = 3/(15*24);
    dt = 0.1;               % 6 min
    D = 200;                % Simulate for D days
    N_t = floor(D*24/dt);  % Corresponding no of hours
    nu = 1/(24*50);         % Average loss of immunity
    Delta = 10;             % Start point of campaign (in days)
    p_0 = 0.001;

    t = linspace(0, N_t*dt, N_t+1);
    S = zeros(N_t+1, 1);
    I = zeros(N_t+1, 1);
    R = zeros(N_t+1, 1);
    V = zeros(N_t+1, 1);

    % Initialize first conditions
    S(1) = 50;
    I(1) = 1;
    R(1) = 0;
    V(1) = 0;
```

```
    epsilon = 1e-12;
    % Iterate equations
    for n = 1:N_t
        S(n+1) = S(n) - dt*beta*S(n)*I(n) + dt*nu*R(n) - dt*p(t(n),n)*S(n);
        V(n+1) = V(n) + dt*p(t(n),n)*S(n);
        I(n+1) = I(n) + dt*beta*S(n)*I(n) - dt*gamma*I(n);
        R(n+1) = R(n) + dt*gamma*I(n) - dt*nu*R(n);
        loss = (V(n+1) + S(n+1) + R(n+1) + I(n+1)) - (V(1) + S(1) + R(1) + I(1));
        if loss > epsilon
            fprintf('loss: %g\n', loss);
        end
    end

    figure();
    plot(t, S, t, I, t, R, t, V);
    legend('S', 'I', 'R', 'V', 'Location', 'northeast');
    xlabel('hours');
    % saveas() gives poor resolution in plots, use the
    % third party function export_fig to save the plot?

end
```

4.8 - OSC_FE Function

```
function [u, v, t] = osc_FE(X_0, omega, dt, T)
    N_t = floor(T/dt);
    u = zeros(N_t+1, 1);
    v = zeros(N_t+1, 1);
    t = linspace(0, N_t*dt, N_t+1);

    % Initial condition
    u(1) = X_0;
    v(1) = 0;

    % Step equations forward in time
    for n = 1:N_t
        u(n+1) = u(n) + dt*v(n);
        v(n+1) = v(n) - dt*omega^2*u(n);
    end
end

function demo_osc_FE()
    omega = 2;
    P = 2*pi/omega;
    dt = P/20;
    T = 3*P;
    X_0 = 2;
    [u, ~, t] = osc_FE(X_0, omega, dt, T);

    plot(t, u, 'b-', t, X_0*cos(omega*t), 'r--');
    legend('numerical', 'exact', 'Location', 'northwest');
    xlabel('t');
end
```

4.9 - OSC_FE 2 file

```
function test_osc_ode_FE()
    function res = f(sol, ~)
        u = sol(1);
        v = sol(2);
        res = [v -omega^2*u];
    end

    % Set and compute problem dependent parameters
    omega = 2;
    X_0 = 1;
    number_of_periods = 6;
    time_intervals_per_period = 20;

    P = 2*pi/omega;                    % Length of one period
    dt = P/time_intervals_per_period;  % Time step
    T = number_of_periods*P;           % Final simulation time
    U_0 = [X_0 0];                     % Initial conditions

    % Produce u data on file that will be used for reference
    file_name = 'osc_FE_data';
    osc_FE_2file(file_name, X_0, omega, dt, T);

    [sol, t] = ode_FE(@f, U_0, dt, T);
    u = sol(:,1);

    % Read data from file for comparison
    infile = fopen(file_name, 'r');
    u_ref = fscanf(infile, '%f');
    fclose(infile);

    tol = 1E-5;                 % Tried several stricter ones first
    for n = 1:length(u)
        err = abs(u_ref(n) - u(n));
        assert(err < tol, 'n=%d, err=%g', n, err);
    end

    % Choose to also plot, just to get a visual impression of u
    plot(t, u_ref, 'b-', t, u, 'r--');
    legend('u ref', 'u');
end

function osc_FE_2file(filename, X_0, omega, dt, T)
    N_t = floor(T/dt);
    u = zeros(N_t+1, 1);
    v = zeros(N_t+1, 1);

    % Initial condition
    u(1) = X_0;
    v(1) = 0;

    outfile = fopen(filename, 'w');
```

```
    fprintf(outfile,'%10.5f\n', u(1));

    % Step equations forward in time
    for n = 1:N_t
        u(n+1) = u(n) + dt*v(n);
        v(n+1) = v(n) - dt*omega^2*u(n);
        fprintf(outfile,'%10.5f\n', u(n+1));
    end
    fclose(outfile);
end
```