```
clc;close all;clear
format long g
%% Constants
% Only change this section for different configuration
Conventional_airfoil = 0; % 0 for supercritical airfoil
Advanced_technology = 0; % Adjust weight after weight loop: 1 for composite
material, 2 for aluminum/lithium structure
Debug = 0;
Swept_angle = 0;
AR = 3;
AR_max = 20;
AR_step_size = 1;
% C_L loop constants
Velocity_approach = 135; % knot
Mach_cruise = 0.80;
Range = 3500; % Nmi
max_percent_fuel_at_landing = 45/100; % max % fuel at landing, 75% for sample
calculation
% TOFL constants
Number_of_engine = 2;
Takeoff_field_length = 6900;
% Weight constants
K_w = 1.01; % 1.03 for fuselage engine
Eta = 1.5*2.5; % Ultimate load factor
Constant_Weight_fuselage = 1; % 1.1 if 3-class international 1 if not
Taper_ratio = 0.35;
K_f = 11.5; % constant for PAX > 135
PAX = 210; % slang for passenger
N_seats_abreast = 6;
```

```matlab
N_aisles = 2; % standard
K_ts = 0.17; % 0.17 for wing engine, 0.25 for fuselage engine
Weight_cargo = 8000; % lb
N_flight_crew = 2;
N_cabin_attendants = 6;
% Drag calculation constants
Fuselage_length = 179; % from tip to tail [ft]
Fuselage_diameter = 20;
% Climb cosntants
Initial_cruise_altitude = 35000; % [ft]
% Thrust check at top climb constant
JT8D = 0; % 1 for JT8D engine and 0 for JT9D engine
%% Loops
fprintf("Swept angle = %.4f degree.\n", Swept_angle)
AR_number_of_steps = (AR_max - AR) / AR_step_size;
AR_list = [];
DOC_list = [];
for i = 1:AR_number_of_steps
   Adjustment_Weight_to_Thrust_ratio = 0;
   fail = 1;
   while fail == 1
       fail = 0;
       % Initalize loop variable
       Ajustment_Factor_Fuel = 0.02;
       Range_all_out_guess = 0;
       Range_all_out = 1E10;
       Range_iteration_limit = 5000;
       % if AR > 13
       %     Range_iteration_limit = 5000;
```

```
        % end
        Range_iteration = 0;
        while (abs(Range_all_out-Range_all_out_guess) > 100) && Range_iteration
< Range_iteration_limit
            % abs(Range_all_out-Range_all_out_guess)
            Range_iteration = Range_iteration + 1;
            if Debug == 1
                fprintf("Debug. Currently in Range loop.\n")
            end
            %% C_L loop
            % Initalize loop variable
            C_L = 0.58;
            C_L_final = 0.1;
            C_L_iteration_limit = 5000;
            C_L_iteration = 0;
            % While loop boundary and iteration taken from Psuedo_Code.m
            while (abs(C_L_final-C_L) > .005) && (C_L_iteration <
C_L_iteration_limit)
                C_L_iteration = C_L_iteration + 1;
                if Conventional_airfoil == 1
                    Delta_Mach_div = -0.348*C_L + 0.191;
                elseif Conventional_airfoil == 0
                    Delta_Mach_div = -0.179 + 1.07*C_L + -1.84*C_L^2 +
0.873*C_L^3;
                else
                    fprintf("Airfoil must be either conventional or
supercritical.")
                end
                % 3. Calculate Divergent Mach number
```

```
                Mach_div = (Mach_cruise + 0.004) - Delta_Mach_div;
                % 4. Use Figure 1a to find t/c
                if Conventional_airfoil == 1
                    t_c_0 = -0.634*Mach_div + 0.572;
                    t_c_10 = -0.616*Mach_div + 0.563;
                    t_c_15 = -0.593*Mach_div + 0.551;
                    t_c_20 = -0.565*Mach_div + 0.537;
                    t_c_25 = -0.533*Mach_div + 0.519;
                    t_c_30 = -0.504*Mach_div + 0.505;
                    t_c_35 = -0.468*Mach_div + 0.486;
                    t_c_40 = -0.428*Mach_div + 0.464;
                    if (Swept_angle >= 0) && (Swept_angle <= 10)
                        t_c = ((Swept_angle - 0) / (10 - 0)) * (t_c_10 - t_c_0)
+ t_c_0;
                    elseif (Swept_angle >= 10) && (Swept_angle < 15)
                        t_c = ((Swept_angle - 10) / (15 - 10)) * (t_c_15 -
t_c_10) + t_c_10;
                    elseif (Swept_angle >= 15) && (Swept_angle < 20)
                        t_c = ((Swept_angle - 15) / (20 - 15)) * (t_c_20 -
t_c_15) + t_c_15;
                    elseif (Swept_angle >= 20) && (Swept_angle < 25)
                        t_c = ((Swept_angle - 20) / (25 - 20)) * (t_c_25 -
t_c_20) + t_c_20;
                    elseif (Swept_angle >= 25) && (Swept_angle < 30)
                        t_c = ((Swept_angle - 25) / (30 - 25)) * (t_c_30 -
t_c_25) + t_c_25;
                    elseif (Swept_angle >= 30) && (Swept_angle < 35)
                        t_c = ((Swept_angle - 30) / (35 - 30)) * (t_c_35 -
t_c_30) + t_c_30;
```

```
                    elseif (Swept_angle >= 35) && (Swept_angle <= 40)
                        t_c = ((Swept_angle - 35) / (40 - 35)) * (t_c_40 -
t_c_35) + t_c_35;
                    end
                elseif Conventional_airfoil == 0
                    t_c_0 = 3.49 + -8.26*Mach_div + 4.98*Mach_div^2;
                    t_c_5 = 3.36 + -7.87*Mach_div + 4.71*Mach_div^2;
                    t_c_10 = 3.17 + -7.32*Mach_div + 4.32*Mach_div^2;
                    t_c_15 = 3.08 + -6.97*Mach_div + 4.04*Mach_div^2;
                    t_c_20 = 2.86 + -6.29*Mach_div + 3.55*Mach_div^2;
                    t_c_25 = 3.07 + -6.64*Mach_div + 3.68*Mach_div^2;
                    t_c_30 = 3.47 + -7.34*Mach_div + 3.96*Mach_div^2;
                    t_c_35 = 5.35 + -11.3*Mach_div + 6.11*Mach_div^2;
                    t_c_40 = 11.8 + -24.9*Mach_div + 13.3*Mach_div^2;
                    if (Swept_angle >= 0) && (Swept_angle <= 5)
                        t_c = ((Swept_angle - 0) / (5 - 0)) * (t_c_5 - t_c_0) +
t_c_0;
                    elseif (Swept_angle >= 5) && (Swept_angle < 10)
                        t_c = ((Swept_angle - 5) / (10 - 5)) * (t_c_10 - t_c_5)
+ t_c_5;
                    elseif (Swept_angle >= 10) && (Swept_angle < 15)
                        t_c = ((Swept_angle - 10) / (15 - 10)) * (t_c_15 -
t_c_10) + t_c_10;
                    elseif (Swept_angle >= 15) && (Swept_angle < 20)
                        t_c = ((Swept_angle - 15) / (20 - 15)) * (t_c_20 -
t_c_15) + t_c_15;
                    elseif (Swept_angle >= 20) && (Swept_angle < 25)
                        t_c = ((Swept_angle - 20) / (25 - 20)) * (t_c_25 -
t_c_20) + t_c_20;
```

```
            elseif (Swept_angle >= 25) && (Swept_angle < 30)
                t_c = ((Swept_angle - 25) / (30 - 25)) * (t_c_30 -
t_c_25) + t_c_25;
            elseif (Swept_angle >= 30) && (Swept_angle < 35)
                t_c = ((Swept_angle - 30) / (35 - 30)) * (t_c_35 -
t_c_30) + t_c_30;
            elseif (Swept_angle >= 35) && (Swept_angle <= 40)
                t_c = ((Swept_angle - 35) / (40 - 35)) * (t_c_40 -
t_c_35) + t_c_35;
            end
        else
            fprintf("Conventional_airfoil has to be either 1 or 0.\n")
        end
        % 5. Constant: cos^2 t/c AR. Use constant and Fig 3 to find
CL_max
        temp = cosd(Swept_angle)^2 * t_c^2 * AR;
        C_L_max_landing = 2.19 + 11.1*temp + -23.2*temp^2;
        C_L_max_takeoff = 1.18 + 12.9*temp + -30.8*temp^2;
        % 6. Calculate wing loading at landing
        sigma = 0.953; % some kind of ratio related to altitude
        WL_landing =
(Velocity_approach/1.3)^2*(sigma*C_L_max_landing/296);
        % 7. Crusing velocity and All out range
        Velocity_cruise = Mach_cruise * 576.4; % [kts] sqrt(gamma R T) =
576.4
        Range_all_out = Range + 200 + 0.75*Velocity_cruise;
        % 8. Use Figure 4 (Engine JT8D) to find fuel weight to take off
weight ratio
```

```
                Weight_fuel_takeoff_JT8D = 0.0209 + 1.04E-04*Range_all_out +
-5.51E-09*Range_all_out^2 + Ajustment_Factor_Fuel;
                % 9. Engine type is JT9D, not JT8D
                if JT8D == 0 % Engine is JT9D
                    SFC_JT9D = 0.61;
                    SFC_JT8D = 0.78;
                    Weight_fuel_takeoff = Weight_fuel_takeoff_JT8D *
(SFC_JT9D/SFC_JT8D) + 0.0257;
                elseif JT8D == 1
                    Weight_fuel_takeoff = Weight_fuel_takeoff_JT8D;
                else
                    fprintf("JT8D must be either 0 or 1.")
                end
                % 10. Take off wing loading
                WL_takeoff =  WL_landing / (1 - max_percent_fuel_at_landing *
Weight_fuel_takeoff);
                % 11. Initial crusing wing loading
                WL_initial_crusing = 0.965 * WL_takeoff;
                % 12. Calculate lift coefficient for initial crusing
                C_L_initial_crusing = WL_initial_crusing / (1481 * 0.2360 *
Mach_cruise^2);
                C_L_final = C_L_initial_crusing;
                % Conditional Statement to determine whether guess is high or
low
                if C_L_final>C_L
                    C_L = C_L + 0.01;
                else
                    C_L = C_L - 0.01;
                end
```

```
            end
            if Debug == 1
                fprintf('Debug. End C_L loop...\n');
            end
            %% TOFL
            % Weight to Thurst ratio at 0.7 lift off velocity. Figure 5
            if Number_of_engine == 2
                temp = 28.3*Takeoff_field_length*10^(-3) + -9.09;
            elseif Number_of_engine == 3
                temp = 31.5*Takeoff_field_length*10^(-3) - 7.45;
            elseif Number_of_engine == 4
                temp = 32.5*Takeoff_field_length*10^(-3) + 1.41;
            else
                fprintf("Number_of_engine must be 2-4.")
            end
            Weight_Thrust_0_7_Velocity_liftoff = temp * sigma * C_L_max_takeoff
/ WL_takeoff;
            % 0.7 of Mach number at lift off
            Velocity_liftoff = 1.2 * sqrt((296*WL_takeoff) /
(sigma*C_L_max_takeoff));
            Mach_liftoff = Velocity_liftoff / (661*sqrt(sigma));
            Mach_liftoff_0_7 = 0.7 * Mach_liftoff;
            % Weight to Thrust ratio
            Thrust_JT9D_sea_level_static = 45500; % Sea Level Static Thrust
            Thrust_at_0_7_Mach_liftoff = -24567*Mach_liftoff_0_7 + 42600; % JT9D
            Weight_Thrust = Weight_Thrust_0_7_Velocity_liftoff *
Thrust_at_0_7_Mach_liftoff / Thrust_JT9D_sea_level_static +
Adjustment_Weight_to_Thrust_ratio;
            %% Weight
```

```
            % Weight Wing = Weight_Wing * Weight_takeoff^1.195
            Weight_wing = 0.00945 * AR^0.8 * (1 + Taper_ratio)^0.25 * K_w *
Eta^0.5 / ( (t_c + 0.03)^0.4 * cosd(Swept_angle) * WL_takeoff^0.695 );
            % Weight Fuselage = Weight_fuselage * Weight_takeoff^0.235
            l = (3.76*PAX / N_seats_abreast + 33.2) * Constant_Weight_fuselage;
            d = (1.75 * N_seats_abreast + 1.58 * N_aisles + 1) *
Constant_Weight_fuselage;
            Weight_fuselage = 0.6727 * K_f * l^0.6 * d^0.72 * Eta^0.3;
            % Weight landing gear = 0.04 * Weight_takeoff
            Weight_landing_gear = 0.04;
            % Weight nacelle + Weight pylon = Weight_nacelle_pylon *
Weight_takeoff
            Weight_nacelle_pylon = 0.0555 / Weight_Thrust;
            % Weight tail surface = 0.1967 * Weight_wing
            Weight_tail_surface = (K_ts + 0.08/Number_of_engine);
            % Weight tail surface + wing = Weight_tail_surface_wing *
Weight_takeoff^1.195
            Weight_tail_surface_wing = (Weight_tail_surface + 1) * Weight_wing;
            % Weight power plant = Weight_power_plant * Weight_takeoff
            Weight_power_plant = 1/(3.58*Weight_Thrust);
            % Weight fuel = Weight_fuel * Weight_takeoff
            Weight_fuel = 1.0275 * Weight_fuel_takeoff;
            % Weight payload = Weight_payload [lb]
            Weight_payload =  215*PAX + Weight_cargo;
            % Weight fixed equipment = Weight_fixed_equipment +
0.035*Weight_takeoff
            Weight_fixed_equipment = 132 * PAX + 300 * Number_of_engine + 260 *
N_flight_crew + 170* N_cabin_attendants;
            % Construct Weight polynomial. a*x^1.195 + b*x^0.235 + c*x + d = 0
```

```
            a = Weight_tail_surface_wing;
            b = Weight_fuselage;
            c = Weight_landing_gear + Weight_nacelle_pylon + Weight_power_plant
+ Weight_fuel + 0.035 - 1;
            d = Weight_payload + Weight_fixed_equipment;
            % Initalize loop variable
            Weight_takeoff = 545000;
            max_iterations = 5e5;
            iteration = 0;
            while abs(a*Weight_takeoff^1.195 + b*Weight_takeoff^0.235 +
c*Weight_takeoff + d) > 100 && iteration < max_iterations
                % Conditional Statement to determine whether guess is high or
low
                if (a*Weight_takeoff^1.195 + b*Weight_takeoff^0.235 +
c*Weight_takeoff + d) > 0
                    Weight_takeoff = Weight_takeoff + 1000;
                else
                    Weight_takeoff = Weight_takeoff - 1000;
                end
                iteration = iteration + 1;
            end
            if Debug == 1
                fprintf('Debug. End Weight loop...\n\n');
            end
            % Weight calculation for advanced technology
            if Advanced_technology == 1
                Weight_takeoff = Weight_takeoff -
Weight_tail_surface_wing*Weight_takeoff^1.195*0.3 -
```

```
Weight_fuselage*Weight_takeoff^0.235*0.15 - (Weight_fixed_equipment +
0.035*Weight_takeoff)*0.1 - Weight_nacelle_pylon * Weight_takeoff * 0.2;
           elseif Advanced_technology == 2
               Weight_takeoff = Weight_takeoff -
Weight_tail_surface_wing*Weight_takeoff^1.195*0.06 -
Weight_fuselage*Weight_takeoff^0.235*0.06;
           end
           % Reference area
           S_ref = Weight_takeoff / WL_takeoff;
           % Span
           Span = sqrt(AR * S_ref);
           % Mean aerodynamic cord
           MAC = S_ref / Span;
           % Thrust
           Thrust = Weight_takeoff / Weight_Thrust;
           Thrust_per_engine = Thrust / Number_of_engine;
           %% Drag calculation
           % Mach = 5 for this section
           % Reynold_over_Lenglth
           Velocity_at_0_5_Mach = 0.5 * 576.4 * 1.688; % convert [kts] to
[ft/s]
           Reynolds_over_Length = 2.852E6 * 0.5; % [1/ft]
           % Copy and paste from drag project Triet code in MAE 158
           % ----- Wing Parasite Drag -----
           Reynolds_wing = Reynolds_over_Length * MAC;                %
Reynolds number (wing)
           C_f_wing = 0.0798 * Reynolds_wing^-0.195;      % Skin friction
coefficient (wing)
```

```
            Z = ((2 - 0.5^2) * cosd(Swept_angle)) / sqrt(1 - 0.5^2 *
cosd(Swept_angle)^2);
            K_wing = 1 + Z * t_c + 100 * t_c^4;  % Form factor
            S_wet_wing = 2*(S_ref - 20*30)*1.02;        % Wetted area from
model (ft^2)
            f_wing = K_wing * C_f_wing * S_wet_wing; % Equivalent profile drag
area
            % ----- Fuselage Parasite Drag -----
            Reynolds_f = Reynolds_over_Length * Fuselage_length;
% Reynolds number (fuselage)
            C_f_f = 0.0798 * Reynolds_f^(-0.195);          % Skin friction
coefficient (fuselage)
            Lf_Df = Fuselage_length / Fuselage_diameter;                       %
Length-to-diameter ratio
            K_f = 2.29 - 0.353 * Lf_Df + 0.038 * Lf_Df^2 - 1.48E-03 * Lf_Df^3; %
Form factor (fuselage)
            S_wet_f = 0.9*pi*Fuselage_diameter*Fuselage_length;
            f_fuselage = K_f * C_f_f * S_wet_f ;            % Equivalent profile
drag area
            % ----- Other Parasite Drag -----
            f_tail_surface = 0.38*f_wing;
            S_wet_nacelle = 2.1*sqrt(Thrust_per_engine)*Number_of_engine;
            f_nacelle = 1.25* C_f_wing *S_wet_nacelle;
            f_pylon = 0.2*f_nacelle;
            % ----- Total Parasite Drag -----
            f_total = 1.06*(f_wing + f_fuselage + f_tail_surface + f_nacelle +
f_pylon);
            C_D_parasite = f_total / S_ref;
            Oswald_Efficiency = 1/ (1.035 + 0.38 * C_D_parasite * pi * AR);
```

```
Oswald_Efficiency = vpa(Oswald_Efficiency); % format
%% Climb
% Calculate climb velocity
Weight_climb = 0.9825*Weight_takeoff;
Density_ratio_at_20_35th_cruise_height = 0.5702;
Velocity_L_D_max = ( 12.9/(f_total*Oswald_Efficiency)^0.25 ) * sqrt(Weight_climb / (Density_ratio_at_20_35th_cruise_height * Span));
Velocity_climb = 1.3 * Velocity_L_D_max;
Mach_climb = Velocity_climb/576.4;
% Calculate Thrust required
Drag_compressibility = 0; % no compressibility drag in climbing
Drag_parasite = Density_ratio_at_20_35th_cruise_height * f_total * Velocity_climb^2 / 296;
Drag_induced = (94.1/(Density_ratio_at_20_35th_cruise_height*Oswald_Efficiency)) * (Weight_climb / Span)^2 * (1/Velocity_climb^2);
Thrust_required_climb = Drag_parasite + Drag_induced + Drag_compressibility;
% Thrust available per engine
Thrust_available_at_20k = 15400;
Specific_fuel_consumption_at_20k = 0.65; % constant varied with altitude and engine type
Thrust_available = ( Thrust_per_engine / Thrust_JT9D_sea_level_static ) * Thrust_available_at_20k;
Rate_of_climb = 101 * (Number_of_engine * Thrust_available - Thrust_required_climb) * Velocity_climb / Weight_climb;
Time_climb = ( Initial_cruise_altitude / Rate_of_climb ) / 60; % [minutes]
Range_climb = Velocity_climb * Time_climb; % [nmi]
```

```
            Weight_fuel_climb = Number_of_engine * Thrust_available *
Specific_fuel_consumption_at_20k * Time_climb;
            %% Range
            % Lift coefficient
            Weight_0 = Weight_takeoff - Weight_fuel_climb;
            Weight_1 = (1-Weight_fuel_takeoff)*Weight_takeoff;
            C_L_average_cruise = ( (Weight_0 + Weight_1) / (2*S_ref) ) / (1481 *
0.2360 * Mach_cruise^2);
            % Drag coefficient
            C_D_induced = C_L_average_cruise*C_L_average_cruise/
(pi*AR*Oswald_Efficiency);
            Delta_C_D_compressibility = 0.001;
            C_D_total = C_D_parasite + C_D_induced + Delta_C_D_compressibility;
            % Lift over Drag
            Lift_Drag = C_L_average_cruise/C_D_total;
            % Thurst
            Thrust_required_range = 0.5*(Weight_0 + Weight_1) / Lift_Drag;
            Thrust_required_JT9D = (Thrust_required_range * (
Thrust_JT9D_sea_level_static / Thrust_per_engine ))/ Number_of_engine ;
            % Engine graph at 35k
            Specific_fuel_consumption_at_35k = 0.392*Mach_cruise + 0.30856;
            % Range [nmi]
            Range_cruise = (Velocity_cruise/Specific_fuel_consumption_at_35k) *
Lift_Drag * log(Weight_0 / Weight_1);
            Range_all_out_guess = Range_climb + Range_cruise;
            if Range_all_out_guess < Range_all_out
                Ajustment_Factor_Fuel = Ajustment_Factor_Fuel + 0.01;
            else
                Ajustment_Factor_Fuel = Ajustment_Factor_Fuel - 0.01;
```

```
            end
        end
        if Debug == 1
            fprintf("Debug. Range.\n")
        end
        %% Thrust Check at top of climb
        C_L_initial_crusing_thrust_check =
(Weight_0/S_ref)/(1481*0.2360*Mach_cruise^2);
        C_D_induced_thrust_check = C_L_initial_crusing_thrust_check^2/
(pi*AR*Oswald_Efficiency);
        C_D_parasite_thrust_check = f_total/S_ref;
        C_D_total_thrust_check = C_D_parasite_thrust_check +
C_D_induced_thrust_check + Delta_C_D_compressibility;
        Lift_Drag_thrust_check =
C_L_initial_crusing_thrust_check/C_D_total_thrust_check;
        Thrust_required_thrust_check = (Weight_0/Lift_Drag_thrust_check) /
Number_of_engine;
        % Now scale down the above value to that of JT9D:
        Thrust_required_JT9D_thrust_check = Thrust_required_thrust_check * (
Thrust_JT9D_sea_level_static / Thrust_per_engine );
        if JT8D == 1
            Thrust_available_at_35k = 1381*Mach_cruise + 2675;
        else
            Thrust_available_at_35k = 3570*Mach_cruise + 7380;
        end
        if Thrust_required_JT9D_thrust_check > Thrust_available_at_35k
            if Debug == 1
                fprintf('NOT ENOUGH THRUST TOP OF CLIMB\n')
            end
```

```
            fail = 1;
        end
        % Climb gradients
        %% 1st Segment
        C_L_takeoff_segment_1 = C_L_max_takeoff / 1.2^2;
        % C_L_takeoff and Figure 6 to get Delta_C_D_parasite
        temp = C_L_takeoff_segment_1 / C_L_max_takeoff;
        Delta_C_D_parasite_segment_1 = 0.0327 + -0.0707*temp + 0.0893*temp^2 +
-0.151*temp^3 + 0.163*temp^4;
        Delta_C_D_gear = 0.0145;
        C_D_induced_segment_1 = C_L_takeoff_segment_1^2 /
(pi*AR*Oswald_Efficiency);
        C_D_total_segment_1 = C_D_parasite + Delta_C_D_parasite_segment_1 +
Delta_C_D_gear + C_D_induced_segment_1;
        Lift_Drag_segment_1 = C_L_takeoff_segment_1 / C_D_total_segment_1;
        Thrust_required_segment_1 = Weight_takeoff/Lift_Drag_segment_1;
        % Maximum Limit Operating (MLO) dry takeoff thrust
        Thrust_MLO_dry_takeoff_segment_1 = 45479 + -48077*Mach_liftoff +
38144*Mach_liftoff^2;
        Thrust_available_segment_1 = (Thrust_per_engine /
Thrust_JT9D_sea_level_static) * Thrust_MLO_dry_takeoff_segment_1;
        Gradient_1 = ((((Number_of_engine - 1)*(Thrust_available_segment_1)) -
Thrust_required_segment_1) / Weight_takeoff) * 100;
        %% 2nd Segment
        % No Delta_C_D_gear in C_D_total_segment_2
        C_D_total_segment_2 = C_D_parasite + Delta_C_D_parasite_segment_1 +
C_D_induced_segment_1;
        Lift_Drag_segment_2 = C_L_takeoff_segment_1 / C_D_total_segment_2;
        Thrust_required_segment_2 = Weight_takeoff / Lift_Drag_segment_2;
```

```
        Gradient_2 = ((((Number_of_engine - 1) * (Thrust_available_segment_1)) -
Thrust_required_segment_2) / Weight_takeoff) * 100;
        %% 3rd Segment
        C_L_max_clean = 0.191 + 13.1*t_c + -39.5*t_c^2;
        Velocity_segment_3 = 1.2 * (sqrt((296 * WL_takeoff) / (0.9204 *
C_L_max_clean))); % KTS
        Mach_segment_3 = Velocity_segment_3 / 659;
        C_L_segment_3 = C_L_max_clean / (1.2^2);
        C_D_total_segment_3 = C_D_parasite + C_L_segment_3^2 /
(pi*AR*Oswald_Efficiency);
        Lift_Drag_segment_3 = C_L_segment_3 / C_D_total_segment_3;
        Thrust_required_segment_3 = Weight_takeoff / (Lift_Drag_segment_3);
        Thrust_JT9D_Max_Climb_Condition = 37594 + -36139*Mach_segment_3 +
18246*Mach_segment_3^2;
        Thrust_available_segment_3 = (Thrust_per_engine /
Thrust_JT9D_sea_level_static) * Thrust_JT9D_Max_Climb_Condition;
        Gradient_3 = ((((Number_of_engine - 1) * Thrust_available_segment_3) -
Thrust_required_segment_3) / Weight_takeoff) * 100;
        %% Approach
        C_L_approach = C_L_max_takeoff / 1.3^2;
        temp = C_L_approach / C_L_max_takeoff;
        Delta_C_D_parasite_approach = 0.0327 + -0.0707*temp + 0.0893*temp^2 +
-0.151*temp^3 + 0.163*temp^4;
        C_D_total_approach = C_D_parasite + Delta_C_D_parasite_approach +
C_L_approach^2 / (pi*AR*Oswald_Efficiency);
        Lift_Drag_approach = C_L_approach / C_D_total_approach;
        Weight_landing_approach = WL_landing * S_ref;
        Thrust_required_approach = Weight_landing_approach / Lift_Drag_approach;
```

```
        Velocity_approach_segment = sqrt((296 * WL_landing) / (0.953 *
C_L_approach)); % KTS
        Mach_approach = Velocity_approach_segment / 667;
        Thrust_JT9D_Mach_approach_Max_Climb_Condition = -21428*(Mach_approach)^3
+ 43382*(Mach_approach)^2 - 43523*(Mach_approach) + 37935;
        Thrust_available_approach = (Thrust_per_engine /
Thrust_JT9D_sea_level_static) * Thrust_JT9D_Mach_approach_Max_Climb_Condition;
        Gradient_approach = ((Number_of_engine - 1)*Thrust_available_approach -
Thrust_required_approach)*100/Weight_landing_approach;
        %% Landing
        C_L_landing = C_L_max_landing / 1.3^2;
        temp = C_L_landing / C_L_max_landing;
        Delta_C_D_parasite_landing = 0.0411 + -0.0684*temp + 8.83E-03*temp^2 +
0.0784*temp^3;
        C_D_total_approach = C_D_parasite + Delta_C_D_parasite_landing +
Delta_C_D_gear + C_L_landing^2 / (pi*AR*Oswald_Efficiency);
        Lift_Drag_landing = C_L_landing / C_D_total_approach;
        Thrust_required_landing = Weight_landing_approach / Lift_Drag_landing;
        Velocity_landing = Velocity_approach;
        Mach_landing = Velocity_landing/667;
        Thrust_JT9D_Mach_landing_dry = 45479 + -48077*Mach_landing +
38144*Mach_landing^2;
        Thrust_available_landing = (Thrust_per_engine /
Thrust_JT9D_sea_level_static) * Thrust_JT9D_Mach_landing_dry;
        Gradient_landing =(Number_of_engine*Thrust_available_landing -
Thrust_required_landing) * 100 / Weight_landing_approach;
        %% Gradient_check
        if Number_of_engine == 2
            if (Gradient_1 < 0)
```

```
    if Debug == 1
        fprintf('Gradient 1 fail.\n')
    end
    fail = 1;
end
if (Gradient_2 < 2.4)
    if Debug == 1
        fprintf('Gradient 2 fail.\n')
    end
    fail = 1;
end
if (Gradient_3 < 1.2)
    if Debug == 1
        fprintf('Gradient 3 fail.\n')
    end
    fail = 1;
end
if (Gradient_approach < 2.1)
    if Debug == 1
        fprintf('Gradient approach fail.\n')
    end
    fail = 1;
end
if (Gradient_landing < 3.2)
    if Debug == 1
        fprintf('Gradient landing fail.\n')
    end
    fail = 1;
end
```

```
elseif Number_of_engine == 3
    if (Gradient_1 < 0.3)
        if Debug == 1
            fprintf('Gradient 1 fail.\n')
        end
        fail = 1;
    end
    if (Gradient_2 < 2.7)
        if Debug == 1
            fprintf('Gradient 2 fail.\n')
        end
        fail = 1;
    end
    if (Gradient_3 < 1.5)
        if Debug == 1
            fprintf('Gradient 3 fail.\n')
        end
        fail = 1;
    end
    if (Gradient_approach < 2.49)
        if Debug == 1
            fprintf('Gradient approach fail.\n')
        end
        fail = 1;
    end
    if (Gradient_landing < 3.2)
        if Debug == 1
            fprintf('Gradient landing fail.\n')
        end
```

```
        fail = 1;
    end
elseif Number_of_engine == 4
    if (Gradient_1 < 0.5)
        if Debug == 1
            fprintf('Gradient 1 fail.\n')
        end
        fail = 1;
    end
    if (Gradient_2 < 3.0)
        if Debug == 1
            fprintf('Gradient 2 fail.\n')
        end
        fail = 1;
    end
    if (Gradient_3 < 1.7)
        if Debug == 1
            fprintf('Gradient 3 fail.\n')
        end
        fail = 1;
    end
    if (Gradient_approach < 2.7)
        if Debug == 1
            fprintf('Gradient approach fail.\n')
        end
        fail = 1;
    end
    if (Gradient_landing < 3.2)
        if Debug == 1
```

```
                fprintf('Gradient landing fail.\n')
            end
            fail = 1;
        end
    else
        fprintf("Check number of engine. Only 2-4 engines are allowed.\n")
    end
    if fail == 1
        % fprintf("Debug.Fail after Gradient Check.\n")
        Adjustment_Weight_to_Thrust_ratio =
Adjustment_Weight_to_Thrust_ratio + 0.01;
    end
end
%% Direct Operating Cost (DOC)
% Block velocity:
Distance_block = 1.15 * Range;
Time_ground_maneuvering = 0.25; % Hour
Time_descent = 0 ;
Time_additional_miscellaneous = 0.1;
Time_cruise = (Distance_block + 0.02*Distance_block + 20 -
(Range_climb*1.15) - 0 ) / (1.15 * Velocity_cruise);
Velocity_block = Distance_block / (Time_ground_maneuvering + Time_climb +
Time_descent + Time_cruise + Time_additional_miscellaneous);
% Block time
Time_block = Time_ground_maneuvering + Time_climb + Time_descent +
Time_cruise + Time_additional_miscellaneous;
% Block fuel
Fuel_climb = Weight_fuel_climb;
```

```
   Fuel_cruise = Thrust_required_range * Specific_fuel_consumption_at_35k *
Time_cruise;
   Fuel_additional_miscellaneous = Thrust_required_range *
Specific_fuel_consumption_at_35k * Time_additional_miscellaneous;
   Fuel_block = Fuel_climb + Fuel_cruise + Fuel_additional_miscellaneous;
   % Flying operation cost
   % a. Flight Crew
   Payload_in_tons = Weight_payload / 2000; % tons
   Velocity_cruise_in_mph = 1.15 * Velocity_cruise;
   Cost_over_block_hour = 17.849 * (Velocity_cruise_in_mph *
(Weight_takeoff/10^5) )^0.3 + 40.83;
   Cost_light_crew_per_ton_mile = Cost_over_block_hour / (Velocity_block *
Payload_in_tons);
   % b. Fuel and Oil
   Cost_fuel_per_pound = 0.4 * (1 / 6.4);
   Cost_Oil_per_pound = 2.15;
   Cost_fuel_and_oil_per_ton_mile = (1.02 * Fuel_block * Cost_fuel_per_pound +
Number_of_engine * Cost_Oil_per_pound * Time_block * 0.135) / (Distance_block *
Payload_in_tons);
   % C. Hull Insurance
   Weight_airframe = Weight_takeoff - Weight_fuel * Weight_takeoff -
Weight_payload - Weight_power_plant * Weight_takeoff;
   Cost_airframe = 2.4*10^6 + 87.5 * Weight_airframe;
   Cost_per_engine = 590000 + (16 * Thrust_per_engine);
   Cost_total_aircraft = Cost_airframe + Number_of_engine * Cost_per_engine;
   Insurance_Rate = 1/100;
   Utilization_rate_in_flight_hours_per_year = 630 + 4000 / (1 + 1 /
(Time_block + 0.5));
```

```
   Cost_Hull_insurance_per_ton_mile = (Insurance_Rate * Cost_total_aircraft) /
(Utilization_rate_in_flight_hours_per_year * Velocity_block * Payload_in_tons);
   % Direct maintenance
   % a. Airframe labor
   Coefficient_flight_hour_airframe_labor = (4.9169 * log10(Weight_airframe /
1000)) - 6.425;
   Coefficient_flight_cycle_labor = 0.21256 * (log10(Weight_airframe /
1000))^3.7375;
   Time_maintenance_flight = Time_block - Time_ground_maneuvering;
   Labor_rate_per_hour = 8.6;
   Cost_airframe_labor_per_ton_mile = ((Coefficient_flight_hour_airframe_labor
* Time_maintenance_flight + Coefficient_flight_cycle_labor) / (Velocity_block *
Time_block * Payload_in_tons)) * Labor_rate_per_hour;
   % b. Airframe material
   Coefficient_flight_hour_airframe_material = 1.5994 * Cost_airframe / 10^6 +
3.4263;
   Coefficient_flight_cycle_airframe_material = 1.9220 * Cost_airframe / 10^6 +
2.2504;
   Cost_airframe_material_per_ton_mile =
(Coefficient_flight_hour_airframe_material * Time_maintenance_flight +
Coefficient_flight_cycle_airframe_material) / (Velocity_block * Time_block *
Payload_in_tons);
   % c. Engine labor
   Coefficient_flight_hour_engine_labor = (Number_of_engine *
(Thrust_per_engine / 1000)) / ((0.82715 * (Thrust_per_engine / 1000)) +
13.639);
   Coefficient_flight_cycle_engine_labor = 0.2 * Number_of_engine;
```

```
   Cost_engine_labor_per_ton_mile = (((Coefficient_flight_hour_engine_labor *
Time_maintenance_flight) + Coefficient_flight_cycle_engine_labor) *
Labor_rate_per_hour) / (Velocity_block * Time_block * Payload_in_tons);
   % d. Engine material
   Coefficient_flight_hour_engine_material = ((28.2353 * Cost_per_engine /
10^6) - 6.5176) * Number_of_engine;
   Coefficient_flight_cycle_engine_material = ((3.6698 * Cost_per_engine /
10^6) + 1.3685) * Number_of_engine;
   Cost_engine_material_per_ton_mile = (Coefficient_flight_hour_engine_material
* Time_maintenance_flight + Coefficient_flight_cycle_engine_material) /
(Velocity_block * Time_block * Payload_in_tons);
   % e. Total maintenance
   Cost_total_maintenance = (Cost_airframe_labor_per_ton_mile +
Cost_airframe_material_per_ton_mile + Cost_engine_labor_per_ton_mile +
Cost_engine_material_per_ton_mile)*2;
   % Depreciation
   Cost_total_maintenance_depreciation = ( Cost_total_aircraft + (0.06 *
(Cost_total_aircraft - (Number_of_engine * Cost_per_engine))) + (0.3 *
Number_of_engine * Cost_per_engine) ) / ( Velocity_block * Payload_in_tons * 14
* Utilization_rate_in_flight_hours_per_year );
   % Total Direct Operating Cost (DOC)
   Direct_operating_cost_per_ton_mile = Cost_light_crew_per_ton_mile +
Cost_fuel_and_oil_per_ton_mile + Cost_Hull_insurance_per_ton_mile +
Cost_total_maintenance + Cost_total_maintenance_depreciation;
   Direct_operating_cost_per_passenger_mile =
Direct_operating_cost_per_ton_mile * Payload_in_tons / PAX;
   DOC = Direct_operating_cost_per_ton_mile;
   DOC_list(end+1) = DOC;
   AR = AR + AR_step_size;
```

```
    AR_list(end+1) = AR;
    fprintf("AR = %.4f\n", AR)
    fprintf("DOC = %.4f\n", DOC)
end
figure;
plot(AR_list, DOC_list, '-o', 'LineWidth', 2, 'MarkerSize', 8);
grid on;
xlabel('Aspect Ratio (AR)');
ylabel('Direct Operating Cost (DOC)');
title('Effect of Aspect Ratio on Direct Operating Cost');
legend('DOC vs AR');
```