Setup

```
clc
clear
close all

addpath("..")% Not needed if +PropPrelib folder is in your current path.

import PropPrelib.*;

units BE;
atmodel Standard;
dragmodel FutureFighter;
enginemodel LBTF;

%mcfg is a struct with variables that will be the same throughout the mission.
mcfg.TR = 1.07; %Throttle Ratio
mcfg.TLto = 1.24; %Wing Loading (Takeoff)
mcfg.WLto = 70; %Thrust Loading (Takeoff)
beta(1) = 0.9;
```

Maneuver 1

```
[PI(1), stats{1}] = Maneuver.E('beta', beta(end),...
                               'M'   , 0.9,...
                               'alt' , 42000,...
                               'D'   , 200*5280,...
                                mcfg); %Rest of mission parameters
beta(end+1) = PI(end)*beta(end);
fprintf('\nManeuver %d: (%s)\n', length(PI), Maneuver.E.name);
ppstruct(stats{end}, 4);
```

```
Maneuver 1: (E - Constant Altitude/Speed Cruise)
    Alpha_req: 0.07566
        Alpha: 0.1706
       AB_req: 0
           AB: 0
         TFSC: 0.0002818
       Beta_1: 0.9
       Beta_2: 0.8685
           CD: 0.03256
           CL: 0.3123
        CDdCL: 0.1042
         Time: 1212
           PI: 0.965
```

Maneuver 2

```
[PI(end+1), stats{end+1}] = Maneuver.E('beta', beta(end),...
                                       'M'   , 1.6    ,...
                                       'alt' , 30000   ,...
                                       'D'   , 100*5280,...
                                       mcfg); %Rest of mission parameters
beta(end+1) = PI(end)*beta(end);
fprintf('\nManeuver %d: (%s)\n', length(PI), Maneuver.E.name);
ppstruct(stats{end}, 4);
if any([stats{2}.AB_req]>0)
    warning('It is impossible subsonic cruise for maneuver 2: (AB_req=%.0f%%)\n',
max([stats{2}.AB_req])*100);
else
    fprintf('It is possible for subsonic cruise for maneuver 2: (AB_req=0)\n');
end
```

```
Maneuver 2: (E - Constant Altitude/Speed Cruise)
    Alpha_req: 0.3742
        Alpha: 0.4453
       AB_req: 0
           AB: 0
         TFSC: 0.0003415
       Beta_1: 0.8685
       Beta_2: 0.8175
           CD: 0.02884
           CL: 0.05399
        CDdCL: 0.5342
         Time: 331.8
           PI: 0.9413
It is possible for subsonic cruise for maneuver 2: (AB_req=0)
```

Maneuver 3

```
[PI(end+1), stats{end+1}] = Maneuver.F('beta', beta(end),...
                                       'M'   , 0.9    ,...
                                       'alt' , 30000   ,...
                                       'n'  , 4.5,...
                                       'Turns', 1,...
                                       mcfg); %Rest of mission parameters
beta(end+1) = PI(end)*beta(end);
fprintf('\nManeuver %d: (%s)\n', length(PI), Maneuver.F.name);
ppstruct(stats{end}, 4);
fprintf('The minimum required afterburner for maneuver 3 is %.0f%%\n',
max([stats{3}.AB_req])*100);
```

```
Maneuver 3: (F - Constant Altitude/Speed Turn)
    Alpha_req: 0.4475
        Alpha: 0.4476
       AB_req: 0.728
           AB: 0.728
         TFSC: 0.0004108
       Beta_1: 0.8175
       Beta_2: 0.8085
           CD: 0.109
           CL: 0.7227
        CDdCL: 0.1508
         Time: 39.85
           PI: 0.9889
The minimum required afterburner for maneuver 3 is 73%
```

Maneuver 4

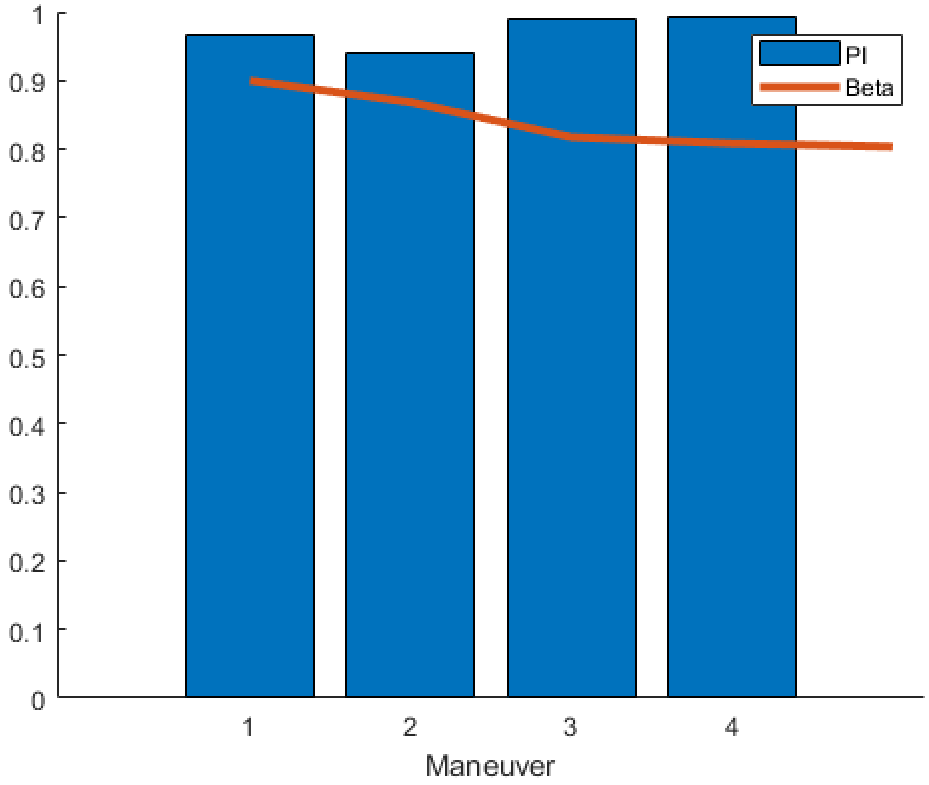
```
%Parameters for single maneuvers can also be stacked in structs
m4cfg.beta = beta(end);
m4cfg.M1 = 0.9;
m4cfg.M2 = 1.6;
m4cfg.alt = 30000;
m4cfg.AB = 1;

%Trying maneuver 4 with 1 interval
PI1int = Maneuver.B('Intervals', 1,...
                    m4cfg,... %Rest of Manuever 4 parameters
                    mcfg); %Rest of mission parameters

%Trying maneuver 4 with 1 interval
[PI(end+1), stats{end+1}] = Maneuver.B('Intervals', 3,...
                                       m4cfg,... %Rest of Manuever 4 parameters
                                       mcfg); %Rest of mission parameters
beta(end+1) = PI(end)*beta(end);
fprintf('\nManeuver %d: (%s)\n', length(PI), Maneuver.B.name);
ppstruct(stats{end}, 4);
fprintf('Maneuver 4 (1 Interval(s)): PI = %.5f\nManeuver 4 (3 Interval(s)): PI = %.5f\n', PI1int,
PI(end));

hold on
bar(PI)
plot(beta,'LineWidth',3)
xlabel('Maneuver')
xticks(1:length(PI))
legend('PI', 'Beta')
```

```
Maneuver 4: (B - Horizontal Acceleration)
     Alpha: 0.7157
        AB: 1
      TFSC: 0.0004795
    Beta_1: 0.8085
    Beta_2: 0.8037
        PI: 0.994
Maneuver 4 (1 Interval(s)): PI = 0.98490
Maneuver 4 (3 Interval(s)): PI = 0.99405
```


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
3
4
Maneuver
PI
Beta

Important External Functions

```
function [PI, stats] = ConstantAltitudeSpeedCruise(varargin)
    [beta, WLto, TLto, alt, M, TR, D, CDR] = parsevars(varargin);
    import PropPrelib.*

    [~, a, P] = atmos(alt);
    [theta, delta] = atmos_nondimensional(alt);
    [theta_0, delta_0] = adjust_atmos(theta, delta, M);
    q = dynamic_pressure(P, M);
    [K1, CD0, K2] = drag_constants(M);

    CL = beta/q*WLto;
    CD = K1*CL^2 + K2*CL + CD0;
    CDdCL = (CD+CDR)/CL;

    alpha_req = CDdCL*beta/TLto;
    %Find afterburner setting where alpha = alpha_req
    [AB_req, alpha_avail] = required_AB(alpha_req, theta_0, delta_0, TR);

    tfsc_m = tfsc('theta', theta,...
                  'M0'   , M,...
                  'AB'   , AB_req);

    dt = D/(a*M);

    PI = exp(-tfsc_m*CDdCL*dt);

    stats.Alpha_req = alpha_req;
    stats.Alpha = alpha_avail;
    stats.AB_req = AB_req;
    stats.AB = AB_req;
    stats.TFSC = tfsc_m;
    stats.Beta_1 = beta;
    stats.Beta_2 = PI*beta;
    stats.CD = CD;
    stats.CL = CL;
    stats.CDdCL = CDdCL;
    stats.Time = dt;
    stats.PI = PI;
end
```

```
function [PI, stats] = ConstantAltitudeSpeedTurn(varargin)
    [beta, WLto, TLto, alt, M, TR, n, Turns, CDR] = parsevars(varargin);
    import PropPrelib.*

    [~, a, P] = atmos(alt);
    [theta, delta] = atmos_nondimensional(alt);
    [theta_0, delta_0] = adjust_atmos(theta, delta, M);
    q = dynamic_pressure(P, M);
    [K1, CD0] = drag_constants(M);
    K2 = 0;

    CL = n*beta/q*WLto;
    CD = K1*CL^2 + K2*CL + CD0;
    CDdCL = (CD+CDR)/CL;

    alpha_req = CDdCL*n*beta/TLto;
    %Find afterburner setting where alpha = alpha_req
    [AB_req, alpha_avail] = required_AB(alpha_req, theta_0, delta_0, TR);

    tfsc_m = tfsc('theta', theta,...
                  'M0'   , M,...
                  'AB'   , AB_req);

    V = a*M;
    dt = 2*pi*Turns*V/(g0*sqrt(n^2-1));

    PI = exp(-tfsc_m*CDdCL*n*dt);

    stats.Alpha_req = alpha_req;
    stats.Alpha = alpha_avail;
    stats.AB_req = AB_req;
    stats.AB = AB_req;
    stats.TFSC = tfsc_m;
    stats.Beta_1 = beta;
    stats.Beta_2 = PI*beta;
    stats.CD = CD;
    stats.CL = CL;
    stats.CDdCL = CDdCL;
    stats.Time = dt;
    stats.PI = PI;
end
```

```
function [PI, stats] = HorizontalAcceleration(varargin)
    [beta, WLto, TLto, alt, M1, M2, TR, CDR, AB, Intervals] =
parsevars(varargin);
    import PropPrelib.*

    %Loop Prep
    PI = 1;
    Mr = linspace(M1, M2, Intervals+1);
    stats.Alpha = 0;
    stats.AB = 0;
    stats.TFSC = 0;
    stats.Beta_1 = beta;

    %This stays the same every iteration, only calculate it once
    [~, a, P, ~, theta, delta] = atmos(alt);
    for i = 1:Intervals
        M = [Mr(i), Mr(i+1)];
        [theta_0, delta_0] = adjust_atmos(theta, delta, M);
        q = dynamic_pressure(P, M);
        [K1, CD0, K2] = drag_constants(M);

        CL = beta./q.*WLto;
        CD = K1.*CL.^2 + K2.*CL + CD0;
        CDdCL = (CD+CDR)./CL;

        alpha = thrustLapse('theta_0', theta_0,...
                            'delta_0', delta_0,...
                            'TR', TR,...
                            'AB', AB);

        tfsc_m = mean(tfsc('theta', theta,...
                           'M0'   , M,...
                           'AB'   , AB));

        V = a*M;
        V1 = a*M(1);
        V2 = a*M(2);
        V = mean(V);

        dZe = (V2^2-V1^2)/(2*g0);
        u = mean(CDdCL.*(beta./alpha)*(1./TLto));

        PI = exp(-tfsc_m/(V*(1-u))*dZe);

        stats.Alpha = stats.Alpha + (mean(alpha) - stats.Alpha)/i; %Averaging
Alpha
        stats.AB = max([stats.AB, AB]); %Important AB is max
        stats.TFSC = stats.TFSC + (tfsc_m - stats.TFSC)/i; %Averaging TFSC
    end
    stats.Beta_2 = stats.Beta_1 * PI;
    stats.PI = PI;
end
```

Published with MATLAB® R2019a