flappingangles

This script evaluates a rotor blade's flapping angles as a function of the azimuth angle, and its derivative. In this scope, the first harmonic flapping motion has been determined with the assumptions of small flapping angles and linear aerodynamics. [1]

$$\beta = \beta_0 + \beta_{1c}\cos(\psi) + \beta_{1s}\sin(\psi) \tag{1}$$

syntax

The default syntax of the function is shown below; all inputs are scalar values.

```
[psi,beta,beta_dot] = flappingangles(V,alpha,W,gamma,sigma,theta_tw,cla,
    omegaR,eR,R)
```

Where:

INPUT

V	Airspeed [m/s]
alpha	Angle of attack [deg]
W	Helicopter *(Rotor)* weight [N]
gamma	Rotor Lock number
sigma	Rotor solidity
theta_tw	Rotor blade twist *(Assumed linear)* [deg]
cla	Rotor blade airfoil lift curve slope [1/rad]
omegaR	Rotor blade tip speed ΩR [m/s]
eR	Adimensional flapping hinge eccentricity e/R
R	Rotor blade radius [m]

OUTPUT

psi	Azimuth angles [deg]
beta	Rotor blade flapping angle [deg]
beta_dot	Rotor blade flapping angle derivative $\frac{d\beta}{d\psi}$

By default ψ is an array of 200 equally spaced points from $0°$ to $360°$.The user may also choose to specify the query points through the *'sample'* keyword.

```
[~,beta,beta_dot] = flappingangles(V,alpha,W,gamma,sigma,theta_tw,cla,omegaR,
    eR,R,'sample',psi)
```

By adding *'output','coefficients'* to the function call, the output will change to the flapping angle coefficients used in 1. That is, respectively:

beta_0	Rotor blade coning [deg]
beta_1c	Rotor blade longitudinal flapping [deg]
beta_1s	Rotor blade lateral flapping [deg] $\frac{d\beta}{d\psi}$

```
[beta_0,beta_1c,beta_1s] = flappingangles(V,alpha,W,gamma,sigma,theta_tw,cla,
    omegaR,eR,R,'output','coefficients')
```

example

A test case for the function is shown below, where it is called multiple times in a cylce in order to obtain the flapping coefficients β_0,β_{1c} and β_{1s} as functions of the advance ratio μ.

```
[b0,b1c,b1s,V] = deal(linspace(0,120,200));
mi = v/213;

for i = 1:length(v)
  [b0(i),b1c(i),b1s(i)] =   flappingangles(V(i)
    ,0,33523.27,8,0.1,-8,5.7,213,0,4,...
    'output','coefficients');
end
```

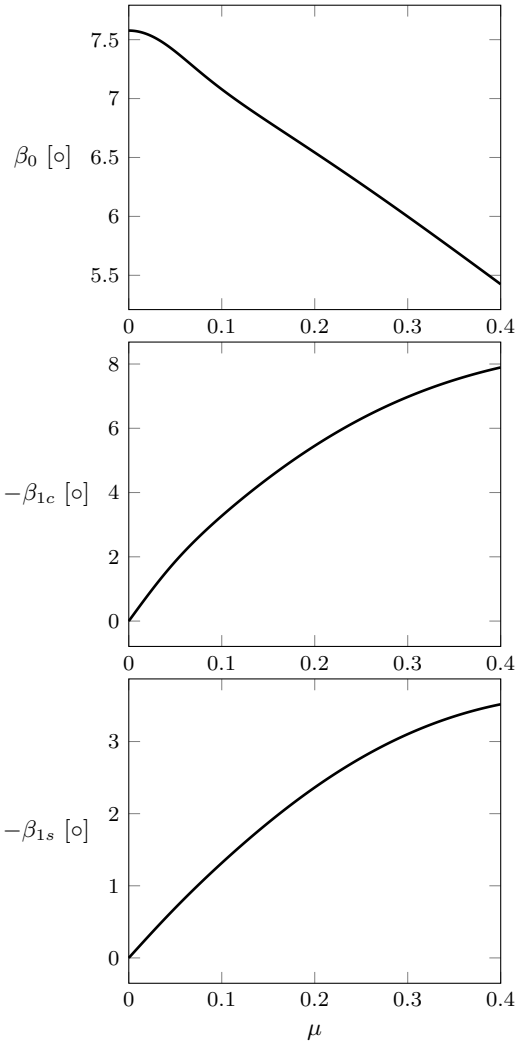

Figure 1: Flapping coefficients as functions of advance ratio [2]

References

[1] Raymond W. Prouty. *Helicopter Performance, Stability and Control*. 1986, pp. 467–469.

[2] Wayne Johnson. *Helicopter Theory*. 1980, pp. 194–197.

code listing

```
function [varargout] = flappingangles(V,alpha,W,gamma,...
sigma,theta_tw,cla,omegaR,eR,R,options)

% Argument validation
arguments
V        (1,1) {mustBeNumeric,mustBeReal}
alpha    (1,1) {mustBeNumeric,mustBeReal}
W        (1,1) {mustBeNumeric,mustBeNonnegative}
gamma    (1,1) {mustBeNumeric,mustBeNonnegative}
sigma    (1,1) {mustBeNumeric,mustBeNonnegative}
theta_tw (1,1) {mustBeNumeric,mustBeReal}
cla      (1,1) {mustBeNumeric,mustBeReal}
omegaR   (1,1) {mustBeNumeric,mustBeNonnegative}
eR       (1,1) {mustBeNumeric,mustBeNonnegative}
R        (1,1) {mustBeNumeric,mustBeNonnegative}
options.output  (1,1) string {mustBeMember(options.output,{'angles','
  coefficients'})} = 'angles'
options.sample  (:,1) {mustBeNumeric,mustBeReal} = linspace(0,360,200)'

end

%% Preliminary calculations

% Converting to radians
theta_tw = theta_tw*pi/180;

% Thrust coefficient
Tc = W/(1.225*omegaR^2*pi*R^2);

% Advance ratio
mi = V*cosd(alpha)/omegaR;

% Adimensional inflow
lambda_i = sqrt(-.5*V.^2+.5*sqrt(V.^4+4*(W/(2*1.225*pi*R^2)).^2))/omegaR;
lambda   = mi.*tand(alpha)+ lambda_i;

% Collective pitch
theta_0 = 6*Tc./(sigma*cla*(1+3/2*mi.^2)*(1-eR))...
-3/4*theta_tw*(1+mi.^2)./(1+3/2*mi.^2)+1.5*lambda./(1+3/2*mi.^2);

%% Flapping coefficients

% Coning
beta_0 = 1/6*gamma*(3/4*theta_0.*(1+mi.^2)+theta_tw*(3/5+mi.^2/2)-lambda)
  ...
*(1-eR)^2;

% Longitudinal flapping
beta_1c = -2*mi.*(4/3*theta_0+theta_tw-lambda)./(1-mi.^2/2)...
-12*eR./(gamma*(1-eR)^3*(1+mi.^4/4)).*(4/3*Tc/sigma*...
(2/3*mi*gamma/(cla*(1+3/2*eR))));

% Lateral flapping
beta_1s = -4/3*mi.*beta_0.*(1+eR/2)./((1+mi.^2/2)*(1-eR)^2)...
-12*eR*(1+eR/2)./(gamma*(1-eR)^3*(1+mi.^4/4))*...
2.*mi.*(4/3*theta_0+theta_tw-lambda);

%% Output

if strcmpi(options.output,'coefficients')
varargout = {beta_0*180/pi,beta_1c*180/pi,beta_1s*180/pi};
else
psi    = options.sample;
beta = (beta_0 + beta_1c.*cosd(psi) + beta_1s.*sind(psi))*180/pi;
beta_dot = ( -beta_1c.*sind(psi) + beta_1s.*cosd(psi))*180/pi;
varargout = {psi,beta,beta_dot};
end

end
```