Inverse drag coefficient by approximation solution (base on time series) 20241010

CHEN, YI-RUI

October 2024

1.Generate Trajectory Data by Numerical Solution

Then we create numerical solutions and add Gaussian noise to quasi-the real Trajectory Data.

$$
\begin{aligned}
\frac{dt}{d\theta} &= -\frac{v(\theta)}{g\cos(\theta)},\\
\frac{dv}{d\theta} &= v\tan\theta + \frac{kv^3(\theta)}{\cos\theta},\\
\frac{d\mathrm{X}}{d\theta} &= -\frac{v^2(\theta)}{g},\\
\frac{d\mathrm{Y}}{d\theta} &= -\frac{v^2(\theta)\tan(\theta)}{g}.
\end{aligned}
$$

2.Approximate Analytical Solution of Trajectory (2-D)

$$v(\theta_k) = \frac{v(\theta_0)\cos\theta_0}{\cos\theta\sqrt{1 + kv^2(\theta_0)\cos^2\theta_0(f(\theta_0) - f(\theta))}},$$

$$f(\theta_k) = \frac{\sin\theta_k}{\cos^2\theta_k} + \ln\tan\left(\frac{\theta_k}{2} + \frac{\pi}{4}\right),$$

$$\beta = k(v^2(\theta_k)\sin\theta_k + v^2(\theta_{k+1})\sin\theta_{k+1}).$$

$$t(\theta_{k+1}) = t(\theta_k) + \frac{2(v(\theta_k)\sin\theta_k - v(\theta_{k+1})\sin\theta_{k+1})}{g(2+\beta)}.$$

$$\mathrm{X}(\theta_{k+1}) = \mathrm{X}(\theta_k) + \frac{v^2(\theta_k)\sin(2\theta_k) - v^2(\theta_{k+1})\sin(2\theta_{k+1})}{2g(1+\beta)},$$

$$\mathrm{Y}(\theta_{k+1}) = \mathrm{Y}(\theta_k) + \frac{v^2(\theta_k)\sin^2(\theta_k) - v^2(\theta_{k+1})\sin^2(\theta_{k+1})}{g(2+\beta)}.$$

3.Set the Cost Function

(Least Square Method)

$$J(k) = [(X_{\text{est}}(k) - X_{\text{data}})^2 + (Y_{\text{est}}(k) - Y_{\text{data}})^2]$$

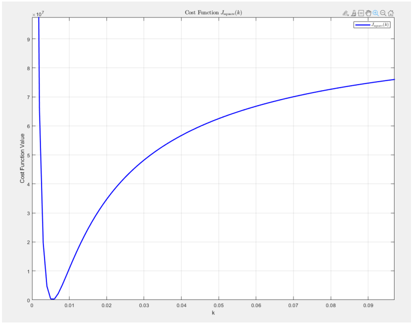

Figure: $J_squre(k)$

4.Minimizing the Cost Function I

(Golden Section Search Method) Define

- the golden ratio $GR = \frac{\sqrt{5}-1}{2}$
- a be the lower limit
- b be the upper limit

- $x_1 = a + GR(b-a)$
- $x_2 = b - GR(b-a)$

If $\mathrm{J(x_1) < J(x_2)}$:

- Eliminate all $x < x_2$
- x_2 becomes the new a
- x_1 becomes the new x_2
- (no change in b)

If $\mathrm{J(x_1) > J(x_2)}$:

- Eliminate all $x > x_1$
- x_1 becomes the new b
- x_2 becomes the new x_1
- (no change in a)

The loop terminates when the interval $b-a$ is smaller than the specified tolerance.

MATLAB Code:main generate trajectort data.m) I

```
clc
clear
close all
%% Problem
% The large RMS in the k = 0.548 case is due to the
    function becoming too nonlinear,
% causing our approximate analytical solution using the
    trapezoidal method to no longer fit the true
    trajectory.
%% Solution
% Change the time step to 0.001 radians, and the result
    will improve.


%%
% Define parameters
global g k
v0 = 50; % Initial velocity
x0 = 0;
z0 = 0;
t0 = 0; % Initial time
```

MATLAB Code:main generate trajectort data.m) II

```
theta0 = (80)*(pi/180); % Initial angle (rad)
initial_conditions = [t0;v0;x0;z0];
g = 9.81; % Acceleration due to gravity (m/s^2)
k = 0.00548; % Damping coefficient
step = (-0.001)*(pi/180); % (rad)

% Define the angle range, slightly above -90 degrees
theta_span = theta0 : step : deg2rad(-90+1e-10);

% Preallocate arrays for better performance
t = zeros(1, length(theta_span));
x = zeros(1, length(t));
z = zeros(1, length(t));
v = zeros(1, length(t));
y = zeros(4, length(t));
k_x = zeros(1, length(t)-1);
k_z = zeros(1, length(t)-1);
k_t = zeros(1, length(t)-1);

% Initial conditions
t(1) = t0;
```

MATLAB Code:main generate trajectort data.m) III

```
v(1) = v0;
x(1) = x0;
z(1) = z0;
y(:,1) = initial_conditions;


% loop
for i = 2:length(theta_span)

    % v(theta)
    v(i) = v_theta( theta_span(i), theta_span(1), v(1), k)
   ;
    % parameters
    a = v(i-1)^2*sin(theta_span(i-1));
    b = v(i).^2*sin(theta_span(i));
    beta = k*(a+b);
    % t(theta)
    t(i) = t_theta(t(i-1), v(i), v(i-1), theta_span(i),
   theta_span(i-1), g, beta);
```

MATLAB Code:main generate trajectort data.m) IV

```
    % x(theta)
    x(i) = x_theta(x(i-1), v(i), v(i-1), theta_span(i),
   theta_span(i-1), g, beta);
    % z(theta)
    z(i) = z_theta(z(i-1), v(i), v(i-1), theta_span(i),
   theta_span(i-1), g, beta);
    % numerical
    y(:,i) = RK4(@f_theta,theta_span(i-1),y(:,i-1),step);

    % break if hit the ground
    if y(4,i) < 0
        t = t(1:i);
        v = v(1:i);
        x = x(1:i);
        z = z(1:i);
        theta_span = theta_span(1:i);
        y = y(:, 1:i);
        break;
    end
```

MATLAB Code:main generate trajectort data.m) V

```
end

% Store x and z into x_data and y_data after the loop
x_data = x;
y_data = z;
save('trajectory_data.mat', 'x_data', 'y_data', '
    theta_span');
```

MATLAB Code:RK4.m MATLAB Code:t theta.m I

```matlab
function y_next = RK4(f, t, y, h)
    k1 = f(t, y);
    k2 = f(t + 0.5 * h, y + 0.5 * h * k1);
    k3 = f(t + 0.5 * h, y + 0.5 * h * k2);
    k4 = f(t + h, y + h * k3);
    k = (k1 + 2 * k2 + 2 * k3 + k4) /6;
    y_next = y + k*h;
end
```

```matlab
function t = t_theta(t0, v, v0, theta, theta0, g, beta)
    t = t0 + 2 * (v0 * sin(theta0) - v * sin(theta)) / (g
   * (2 + beta));
end
```

MATLAB Code:v theta.m MATLAB Code:x theta.m I

```
function v = v_theta( theta, theta0, v0, k)
    f = (sin(theta) ./ cos(theta).^2) + log(tan(theta / 2
   + pi / 4));
    f0 = (sin(theta0) ./ cos(theta0).^2) + log(tan(theta0
   / 2 + pi / 4));
    v = (v0 * cos(theta0)) ./ (cos(theta) .* sqrt(1 + k *
   v0^2 * cos(theta0)^2 .* (f0 - f)));
end
```

```
function x = x_theta(x0, v, v0, theta, theta0, g, beta)
    x = x0 + (v0^2 * sin(2 * theta0) - v^2 * sin(2 * theta
   )) / (2 * g * (1 + beta));
end
```

MATLAB Code:z theta.m I

```
function z = z_theta(z0, v, v0, theta, theta0, g, beta)
    z = z0 + (v0^2 * sin(theta0)^2 - v^2 * sin(theta)^2) /
     (g * (2 + beta));
end
```

MATLAB Code:main estimate k.m I

```
clc;
clear;
close all;

%% main estimate k
% The bisection method cannot be used because the least squares error is always greater than
% or equal to zero (  0  ) and typically not equal to zero (  0  ). Thus, the function does not have
% a root at zero, which is a requirement for the bisection method.
% Load Data
load('trajectory_data.mat');
v0 = 50; % Initial velocity
x0 = 0;
y0 = 0;
g = 9.81;


x_data = x_data + 0.01*randn;
y_data = y_data + 0.01*randn;
```

MATLAB Code:main estimate k.m II

```

k_left = 0;
k_right = 1;
k_span = k_left:0.001:k_right;

J_squre = @(k) costXYsquareerror(x_data, y_data, theta_span
    , v0, x0, y0, g, k);

% goldenSectionSearch find minimum
[k_goldenSectionSearch,iteration_time] =
    goldenSectionSearch(k_left, k_right, J_squre, 1e-6, 1
    e4);

% Display the result of the first optimization
disp(['goldenSectionSearch k: ', num2str(
    k_goldenSectionSearch)])

% plot Cost Function
plotCostFunction(k_span, J_squre);
```

MATLAB Code:costXYsqureerror.m I

```matlab
function J_squre = costXYsqureerror(x_data, y_data,
    theta_span, v0, x0, y0, g, k)
     % Preallocate arrays for the estimated x and y
    positions
     x_est = zeros(1, length(theta_span));
     y_est = zeros(1, length(theta_span));

     % Initial conditions
     x_est(1) = x0;
     y_est(1) = y0;
     v(1) = v0; % Initial velocity

     for i = 2:length(theta_span)
         % v(theta)
         v(i) = v_theta(theta_span(i), theta_span(1), v(1),
      k);

         % Parameters for beta
         a = v(i-1)^2 * sin(theta_span(i-1));
         b = v(i)^2 * sin(theta_span(i));
         beta = k * (a + b);
```

MATLAB Code:costXYsqureerror.m II

```

        % Estimate x and y positions
        x_est(i) = x_theta(x_est(i-1), v(i), v(i-1), theta_span(i), theta_span(i-1), g, beta);
        y_est(i) = z_theta(y_est(i-1), v(i), v(i-1), theta_span(i), theta_span(i-1), g, beta);
    end

    % Compute the cost function based on the difference between estimated and true data
    costX = x_est - x_data;
    costY = y_est - y_data;

    % Compute the total cost as the sum of squared errors
    J_squre = costX * costX' + costY * costY';
end
```

MATLAB Code:goldenSectionSearch.m I

```
function [k,i] = goldenSectionSearch(a, b, J, tol, maxiter
    )
i=0;
    % Golden ratio
    gr = (sqrt(5) - 1) / 2;

    % Initialize internal points
    x1 = a+gr*(b-a);
    x2 = b-gr*(b-a);

    % Calculate objective function values at internal
   points
    J_x1 = J(x1);
    J_x2 = J(x2);

    % Start iteration
    while b-a > tol || i>maxiter
        i = i+1;
        if J_x1 < J_x2
            a = x2;  % Shrink the left boundary
            x2 = x1;
```

MATLAB Code:goldenSectionSearch.m II

```
            J_x2 = J_x1;
            x1 = a+gr*(b-a);
            J_x1 = J(x1);
        else
            b = x1;   % Shrink the right boundary
            x1 = x2;
            J_x1 = J_x2;
            x2 = b-gr*(b-a);
            J_x2 = J(x2);
        end
    end

    % Return the midpoint of the interval as the optimal
   solution
    k = (a + b) / 2;
end
```

MATLAB Code:plotCostFunction.m I

```
function plotCostFunction(k_span, J_squre)

    %                                     k
   J_squre
    J_squre_values = zeros(size(k_span));

    %              k            J_squre
    for i = 1:length(k_span)
        k = k_span(i);
        J_squre_values(i) = J_squre(k);
    end

    %         J_squre(k)     k
    figure;
    plot(k_span, J_squre_values, 'b-', 'LineWidth', 2);

    %         x     y
    xlabel('k');
    ylabel('Cost Function Value');

    %         LaTeX
```

MATLAB Code:plotCostFunction.m II

```
    title('Cost Function $J_{\mathrm{square}}(k)$', 'Interpreter', 'latex');

    %                                LaTeX
    legend('$J_{\mathrm{square}}(k)$', 'Interpreter', 'latex');

    %
    grid on;

end
```