Table of Contents

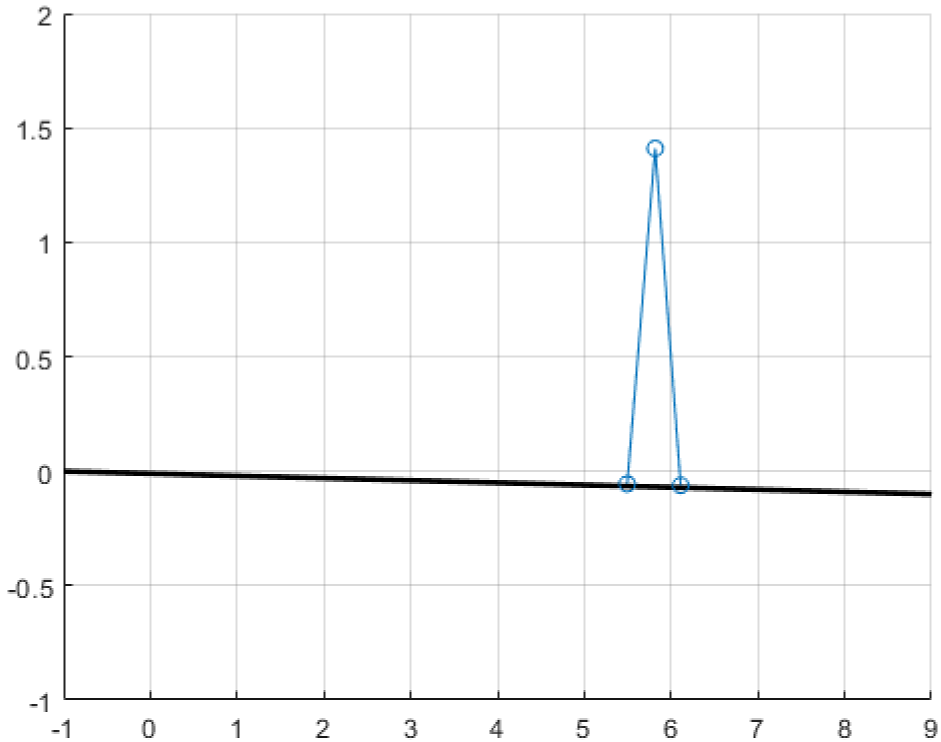

```
clc
close all
clear all
```

ME292B HW04

Simulate Two Link Walker - Problem 1(d)

```
x0 = [0.2065;
    0.4130;
    -0.2052;
    -0.0172];

% init data vectors
t_vec = [];
x_vec = [];

% Loop for 10 steps
for i = 1:10
% Define time range to simulate the system
Tspan = [0 15] ;
t0 = 0 ; % Initial Time

% Initialize vectors
t_ode = []; x_ode = [];

% Define the event functions (stop integration when impact happens)
options = odeset('Events', @two_link_event) ;

% Simulate the system for each step
[t_ode, x_ode] = ode45(@two_link_dynamics, t0+Tspan, x0, options) ;

% Save simulation data
t_vec = [t_vec; t_ode] ;
x_vec = [x_vec; x_ode] ;

% Store row numbers of new steps for animation
t_I(i) = size(t_vec,1);
```

```
% Initialize xo and t for next step
x0(1:4) = two_link_impactdynamics(x_ode(end,1:4)); t0 = t_vec(end);
end

% Animate
animate_two_link_walker(t_vec, x_vec, 0.01, t_I)
```



Problem 2

```
% Define initial condition
x0 = [0.2065;
    0.4130;
    -0.2052;
    -0.0172];

% Compute more accurate fixed point from x0 (this loop is similar to
% 'TwoLinkPoincare, but without the constraint of x0(1) = 1/2*x0(2);

x_old = x0;
tol = 0.01;
err = 1;
while err > 0.01

% Define time range to simulate the system
Tspan = [0 15] ;
t0 = 0 ; % Initial Time
```

```
% Define the event function that stops integration at next poincare
% intersection
options = odeset('Events', @two_link_event) ;

% Simulate the system till next poincare intersection
[t_ode, x_ode] = ode45(@two_link_dynamics, t0+Tspan, x0, options) ;

% Compute point on Poincare map after one cycle (with impact dynamics
% applied)
x_new = two_link_impactdynamics(x_ode(end,:));

% Compute err
err = norm(x_old - x_new);

% Reinitialize for next loop
x_old = x_new;
end

% x_fixed is final x_new value
x_fixed = x_new

% Compute A of linearized Poincare map numerically using finite
 difference
% Change delta iteratively within the loop
A = zeros(length(x_fixed):length(x_fixed));
tol = 0.1;
delta = 0.1*ones(length(x_fixed),1);

for j = 1:size(x_fixed,1)

    [a_j, diff(j)] = compute_aj(x_fixed, delta(j), j);

    while diff(j) > tol
        delta(j) = delta(j) /10;
        [a_j, diff(j)] = compute_aj(x_fixed, delta(j), j);
    end

A(:,j) = a_j';
end

% Display A
A

% Find eigenvalues of A to determine stability
eigA = eig(A)
if abs(eigA) < 1
    disp('The periodic downhill walking gait is exponentially stable')
elseif abs(eigA) == 1
    disp('The periodic downhill walking gait is stable in the sense of
 Lyapunov')
else
    disp('The periodic downhill walking gait is not stable')
end
```

```
x_fixed =

    0.1848
    0.3697
   -0.1901
   -0.0128


A =

         0    0.4595   -0.0987    0.2106
         0    0.9189   -0.1974    0.4212
         0    0.0106    0.7582   -0.1411
         0   -0.0639    0.0764   -0.0454


eigA =

         0
   -0.0043
    0.8598
    0.7762

The periodic downhill walking gait is exponentially stable
```

Placeholder to get MATLAB to publish correctly

```
z = 1;
```

Various functions created for problems 1(a) to 1(c), and problem 2

```
function [dx] = two_link_dynamics(t, x)

th = x(1);
phi = x(2);
dth = x(3);
dphi = x(4);

B = 0.01;
gdivl = 1;
gam = 0.01;

D = [1+2*B*(1 - cos(phi)), -B*(1 - cos(phi));
    B*(1 - cos(phi)), -B];

C = [-B*sin(phi)*(dphi^2-2*dth*dphi);
    B*dth^2*sin(phi)];
```

```
G = [B*gdivl*(sin(th-phi-gam)-sin(th-gam))-gdivl*sin(th-gam);
    B*gdivl*sin(th-phi-gam)];

ddq = inv(D)*(-G-C);

dx = [x(3:4);
      ddq];

end

function [xplus] = two_link_impactdynamics(xminus)

xplus = [-1 0 0 0;
          -2 0 0 0;
          0 0 cos(2*xminus(1)) 0;
          0 0 cos(2*xminus(1))*(1-cos(2*xminus(1))) 0] * xminus';

end

function [value,isterminal,direction] = two_link_event(t,x)

value = x(2) - 2*x(1); % detect when phi - 2*theta == 0 (approx)
isterminal = 1 ; % stop integration when value == 0
direction = 1 ; % detect zero when function is increasing

end

function [a_j, diff] = compute_aj(x_fixed, delta, j)
    ej = zeros(size(x_fixed,1),1);
    ej(j,1) = 1;
    a_j = ( TwoLinkPoincare(x_fixed + delta*ej)...
        - TwoLinkPoincare(x_fixed - delta*ej) ) / (2*delta) ;

    delta_alt = delta/10;
    a_hat_j = ( TwoLinkPoincare(x_fixed + delta_alt*ej)...
        - TwoLinkPoincare(x_fixed - delta_alt*ej) ) / (2*delta_alt) ;

    diff = abs(norm(a_j) - norm(a_hat_j));
end

function [x1] = TwoLinkPoincare(x0)

% Constrain ICs to be on the poincare section (since delta perturbs it
 off
% the section
x0(1) = 1/2*x0(2);

% Define time range to simulate the system
Tspan = [0 15] ;
t0 = 0 ; % Initial Time

% Define the event function that stops integration at next poincare
% intersection
```

```
options = odeset('Events', @two_link_event) ;

% Simulate the system till next poincare intersection
[t_ode, x_ode] = ode45(@two_link_dynamics, t0+Tspan, x0, options) ;

% Compute point on Poincare map after one cycle (with impact dynamics
% applied)
x1 = two_link_impactdynamics(x_ode(end,:));

end
```

Published with MATLAB® R2019a