MEC ENG 193B/292B: Feedback Control of Legged Robots

Homework 3

Professor Koushil Sreenath
UC Berkeley, Department of Mechanical Engineering
September 26, 2025

Larry Hui[1]
SID: 3037729658

[1]University of California at Berkeley, College of Engineering, Department of Mechanical Engineering. Author to whom any correspondence should be addressed. email: larryhui7@berkeley.edu

1 Problems

1.1 Vertical Spring-Mass Hopper

In this problem we will analyze a simple model of a vertical hopper and design our first controller for vertical hopping. Consider a spring-mass-damper system as shown in Figure 1. The free length of the spring is l_0. The mass enters into flight mode when its vertical position is greater than the free length and enters stance mode otherwise, i.e. the dynamics can be written as

$$\ddot{y} = \begin{cases} \text{Flight Dynamics}, & \text{if } y > l_0, \\ \text{Stance Dynamics}, & \text{if } y \leq l_0. \end{cases} \tag{1}$$

During stance, a linear spring (Spring Force, $F_s = -k(y - l_0)$) and a linear damper (Damping Force, $F_d = -c\dot{y}$) along with a control input force $u(t)$ acts on the mass.

Upon impact, assume an identity impact map (i.e. assume the post-impact position and velocity of the mass to be equal to the pre-impact positions and velocities).

During flight, assume a constant gravity force acting on the mass. Furthermore, unless otherwise mentioned, use the parameters given in Table 1.

(a) Draw the free body diagrams in flight and in stance and write down the Stance and Flight Dynamics using the state $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y \\ \dot{y} \end{bmatrix}$

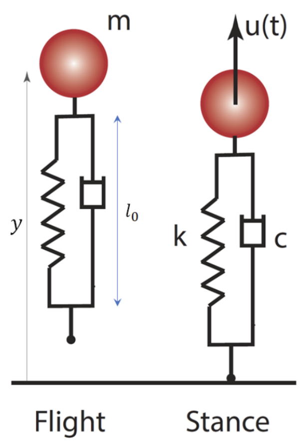
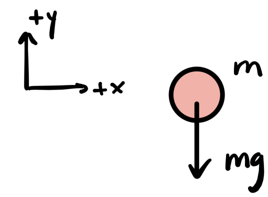


Figure 1: The spring mass hopper

Parameter	Symbol	Value
Spring Constant	k	20 kN/m
Mass	m	80 kg
Free length of spring	l_0	1 m
Damping Coefficient	c	5 kN·s/m
Initial height	x_0	5 m

Table 1: Parameters for the spring mass hopper.

Solution: For the in flight mode, the only force acting on the system is gravity, so our FBD is



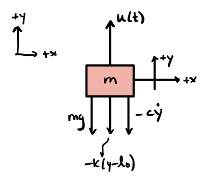
Hence, we can write the dynamics using Newton's 2nd law as

$$m\ddot{y} = -mg \implies \ddot{y} = -g$$

Then putting it into the state-evolution equation with states $\dot{x}_1 = x_2$ and $\dot{x}_2 = -g$, yields

$$\boxed{\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} \dot{y} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} x_2 \\ -g \end{bmatrix}}$$

For the stance configuration, we have the following FBD



Again, like before, we can write the dynamics using Newton's 2nd law as

$$\begin{aligned} m\ddot{y} &= \sum_i F_i \\ m\ddot{y} &= -mg - k(y - l_0) - c\dot{y} + u(t) \\ \ddot{y} &= -g - \frac{k}{m}(y - l_0) - \frac{c}{m}\dot{y} + \frac{1}{m}u(t) \end{aligned}$$

Then putting it into the state-evolution equation with $\dot{x}_1 = x_2$ and $\dot{x}_2 = \ddot{y}$

$$\boxed{\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} \dot{y} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} x_2 \\ -g - \frac{k}{m}(x_1 - l_0) - \frac{c}{m}x_2 + \frac{1}{m}u(t) \end{bmatrix}}$$

(b) We will define the Poincaré section $\mathcal{S}$ at the apex of the flight phase, i.e.

$$\mathcal{S} := \left\{ \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y \\ \dot{y} \end{bmatrix} \,\middle|\, y > l_0,\ \dot{y} = 0 \right\}. \tag{2}$$

Let $P : \mathcal{S} \to \mathcal{S}$ represent the Poincaré map. Consider a state $\mathbf{x} \in \mathcal{S}$ and analytically compute $P(x)$ assuming zero control input force, i.e. $u(t) \equiv 0$, and zero damping, i.e. $c = 0$.

Solution: Assuming $u(t) = 0$ and $c = 0$, the system is considered conservative so we can use conservation of mechanical energy. We can consider the initial state at an apex $\mathbf{x}_i = \begin{bmatrix} y & 0 \end{bmatrix}^\top$, $y \in \mathcal{S}$. We just need to find the state at the next apex $P(\mathbf{x}_i) = \begin{bmatrix} y' & 0 \end{bmatrix}^\top$. Start with balance of energy for each phase of the hybrid model.

$$E_i = E^-_{ground} \implies mgy = \frac{1}{2}m(\dot{y}^-)^2 - mgl_0 \implies \dot{y}^- = -\sqrt{2g(y - l_0)}$$

Then after impact, we take $\dot{y}^- = -\dot{y}^+$ so

$$\dot{y}^+ = \sqrt{2g(y - l_0)}$$

Then by conservation of energy again, we can calculate the energy at the next state using $E^+_{ground} = E_f$

$$\frac{1}{2}m(\dot{y}^+)^2 + mgl_0 = mgy' \implies \frac{1}{2}m(2g(y - l_0)) = mg(y' - l_0) \implies y = y'$$

Therefore, the height of the next apex is the same as the initial apex. Thus, for any state $x \in \mathcal{S}$, the map returns the same state implying that the Poincaré map is an identity map and all points are fixed points

$$\boxed{P\left(\begin{bmatrix} y \\ 0 \end{bmatrix}\right) = \begin{bmatrix} y \\ 0 \end{bmatrix} \implies P(\mathbf{x}) = \mathbf{x},\ \forall \mathbf{x} \in \mathcal{S}}$$

(c) What is a fixed point for this Poincaré map? (Find x^* satisfying $P(x^*) = x^*$.) Next, compute the linear approximation of the Poincaré map about the fixed point. Determine if the fixed point is stable (to do this, compute the eigenvalues of the Jacobian of the Poincaré map. Note that you will get one eigenvalue equal to zero.)

Solution: Assuming damping is sill zero and the control input $u(t) = 0$, every point in the Poincaré section $\mathcal{S}$ would be a fixed point for this Poincaré map. Explicitly, this means any apex state

$$\mathbf{x}^* = \{ \begin{bmatrix} y^* \\ 0 \end{bmatrix} | y^* > l_0 \} \in \mathcal{S}$$

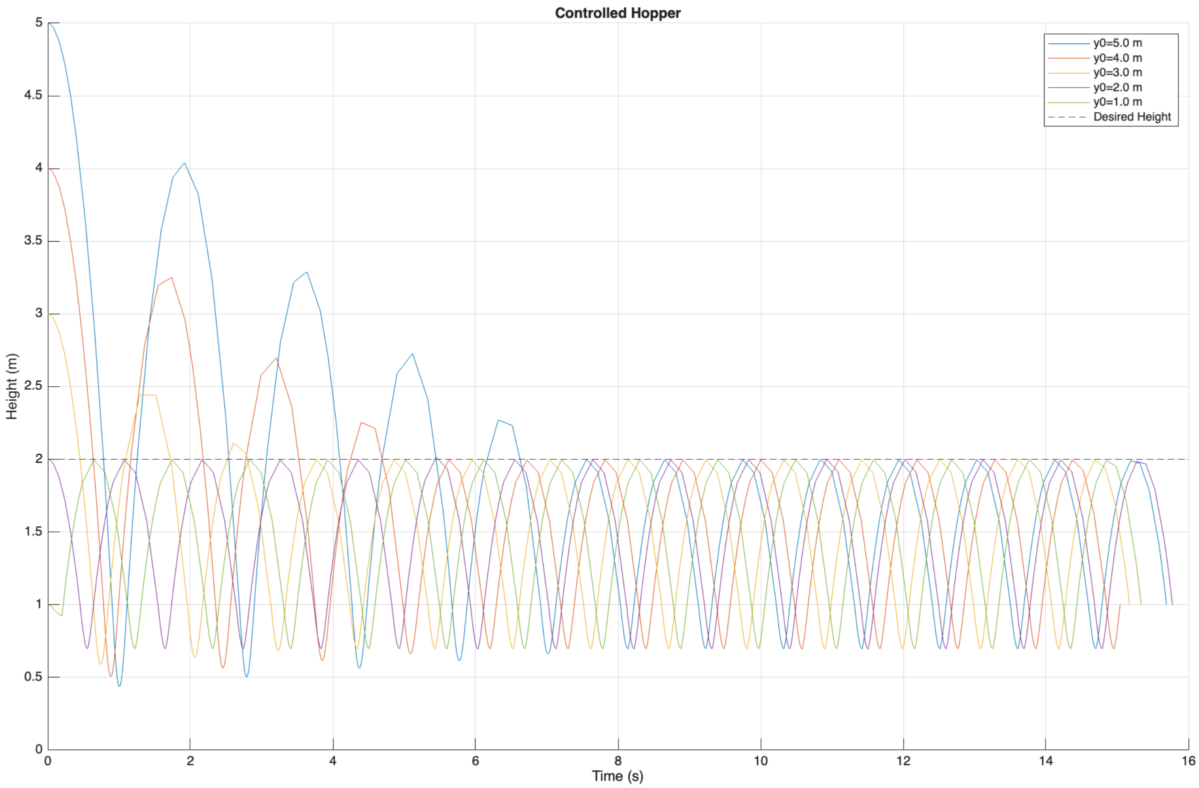
Next, to compute the linear approximation of the Poincaré map, we first compute the Jacobian J

$$J = \left.\frac{\partial P}{\partial \mathbf{x}}\right|_{\mathbf{x}^*} = \begin{bmatrix} \frac{\partial P_1}{\partial y} & \frac{\partial P_1}{\partial \dot{y}} \\ \frac{\partial P_2}{\partial y} & \frac{\partial P_2}{\partial \dot{y}} \end{bmatrix}, \quad P(\mathbf{x}) = [y \quad 0]^\top$$
$$= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

By observation for $\mathbf{x} = \mathbf{x}^*$, the eigenvalues are $\lambda_1 = 1$ and $\lambda_2 = 0$ for it to be exponentially stable, we need $|\lambda_i \left(\left.\frac{\partial P}{\partial x}\right|_{\mathbf{x}^*} \right)| < 1$. Since $\lambda_i = 1 \leq 1 \implies$ the fixed point is not locally exponentially stable or is **marginally stable**.

(d) Now, use your *intuition* to design a controller $u(t)$ such that the height at the apex converges to $y_d = 2\,\text{m}$. Choose **5** different initial conditions and provide plots of the vertical height $y(t)$ of the mass as a function of time t to illustrate convergence of the apex height of the hopping to y_d.
Note: For the above plot, you can either solve for the position $y(t)$ analytically, or by using a numerical solver such as `ode45` in MATLAB.

Solution: Below is the plots for the 5 different IC's with the code below.




```
k   = 20000;
m   = 80;
l0 = 1;
c   = 100;
g   = 9.81;
yd = 2;

reqE = m*g*yd;
init = [5, 4, 3, 2, 1];
figure; hold on;
for i = 1:length(init)
    x0 = [init(i); 0]; t0 = 0;
    t_all = []; x_all = [];
    curr = 'flight';

    while t0 < tf
        if curr ==  flight
            options = odeset('Events', @(t,x) landing(t,x,m,k,c,g,l0,yd,
                reqE));
            [t,x,te,xe,ie] = ode45(@(t,x) flight_dynamics(t,x,m,k,c,g,l0,yd
```

```
                ,reqE), [t0 20], x0, options);
            curr = 'stance';
        else
            options = odeset('Events', @(t,x) takeoff(t,x,m,k,c,g,l0,yd,
                reqE));
            [t,x,te,xe,ie] = ode45(@(t,x) stance_dynamics(t,x,m,k,c,g,l0,yd
                ,reqE), [t0 20], x0, options);
            curr = 'flight';
        end

        t_all = [t_all; t]; x_all = [x_all; x];
        x0 = xe.';  t0 = te;
    end

    plot(t_all, x_all(:,1),'DisplayName', sprintf('y0=%.1f m',init(i)));
end

yline(yd,'k--','DisplayName','Desired Height');
xlabel('Time (s)'); ylabel('Height (m)');
legend show; grid on;
title('Controlled Hopper');


function dx = flight_dynamics(~,x,~,~,~,g,~,~,~)
    y = x(1); v = x(2);
    dx = [v; -g];
end

function dx = stance_dynamics(~,x,m,k,c,g,l0,~,Ed)
    y = x(1); v = x(2);
    Fs = -k*(y-l0);
    Fc = c*v;

    E = 0.5*m*v^2 + 0.5*k*(y-l0)^2 + m*g*y;

    K = 1500;
    if v > 0
        u = max(0, K*(Ed - E));
    else
        u = 0;
    end
    F = Fs - Fc + u - m*g;
    dx = [v; F/m];
end

function [value, isterminal, direction] = landing(~,x,~,~,~,~,l0,~,~)
    value = x(1) - l0;
    isterminal = 1;
    direction = -1;
end

function [value, isterminal, direction] = takeoff(~,x,~,~,~,~,l0,~,~)
    value = x(1) - l0;
    isterminal = 1;
```

```
    direction = 1;
end
```

1.2 Van der Pol Oscillator

In the previous problem, we looked at deriving the Poincaré map analytically. This relied on computing the solution of the hybrid system in (1). However, for legged systems with nonlinear dynamics, it is almost impossible to analytically compute the solution and hence the Poincaré map. For such systems, the Poincaré map can be computed *numerically*. In this problem, we will look at a 2-dimensional nonlinear system and obtain a Poincaré map numerically.

Consider the Van der Pol Oscillator with state

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

with dynamics given by

$$\dot{x}_1 = x_2, \tag{3}$$

$$\dot{x}_2 = \mu \left(1 - x_1^2\right) x_2 - x_1. \tag{4}$$

For this problem, assume $\mu = 1$. We will define the Poincaré section to be

$$\mathcal{S} := \{\mathbf{x} \mid x_1 = 0,\ x_2 > 0\}, \tag{5}$$

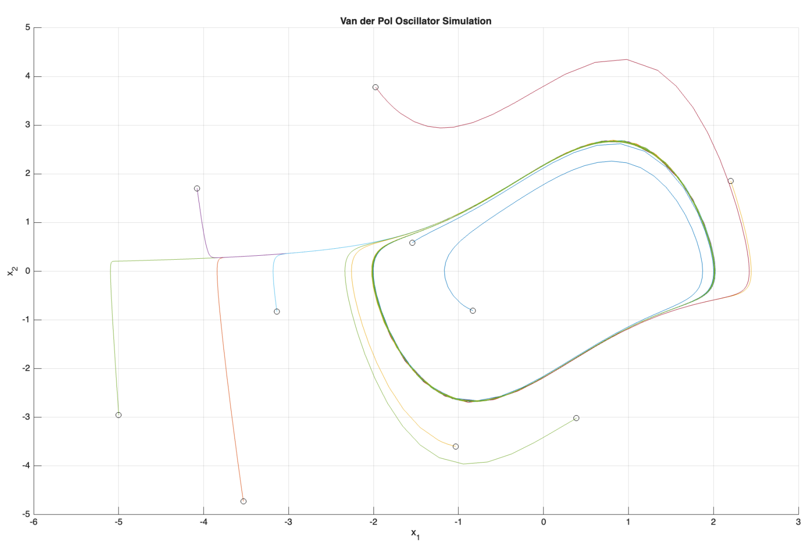
and the Poincaré map $P : \mathcal{S} \to \mathcal{S}$ as $P(\mathbf{x})$.

(a) Simulate the system (using `ode45` in MATLAB or any numerical solver of your choice) for 10 different initial conditions with $x_1, x_2 \in [-5, 5]$ for 50 seconds and provide a plot of x_1 vs. x_2 (also known as the *phase portrait*). Observe the periodic orbit that the solutions converge to.

Solution: Our code is as follows with the periodic orbits

```
mu = 1; time = [0 50];
dynamics = @(t,x) [x(2); mu*(1 - x(1)^2)*x(2) - x(1)];
x0s = 10*rand(10,2) - 5;
figure; hold on;
for i = 1:10
    [t, x] = ode45(dynamics, time, x0s(i,:));
    plot(x(:,1), x(:,2));
    plot(x(1,1), x(1,2), 'ko');
end

xlabel('x_1'); ylabel('x_2');  title('Van der Pol Oscillator'); grid on;
```



(b) We will now numerically determine a fixed point x^* of the Poincaré map such that $P(x^*) = x^*$. We will do this by following the steps below:

(i) Write a MATLAB function `x1 = VanderPolPoincare(x0)` that takes in a point x_0 on the Poincaré section and returns the point on the Poincaré section after one complete cycle (i.e. at the next intersection of the solution with the Poincaré section). Inside the function, you should change one of the coordinates of $\mathbf{x}_0$ to ensure $\mathbf{x}_0 \in \mathcal{S}$.

Solution: We can write the function using `ode45` and creating an event for each crossing

```
function x1 = VanderPolPoincare(x0)
    mu = 1;

    % change one of the coords of x0 to ensure x0 in S (Poincare section)
    x0(1) = 0;
    if x0(2) <= 0
        x0(2) = abs(x0(2))+ eps;
    end

    dynamics = @(t,x) [x(2); mu*(1 - x(1)^2)*x(2) - x(1)];
    options = odeset('Events', @(t,x) crossings(t,x));

    [~,~,~,xe,~] = ode45(dynamics, [0 100], x0, options);
    x1 = xe(end,:)';
end

function [value, isterminal, direction] = crossings(t,x)
    value = x(1); %detect event when x(1) crosses 0
    isterminal = 1; % stop integration when event occurs
    direction = 1; %detect events whne going from -ve to +ve
end
```

(ii) Pick any initial condition $\mathbf{x}_0 \in \mathcal{S}$ (i.e. pick $x_1 = 0$ and any $x_2 > 0$). Apply the above function to obtain $\mathbf{x}_1 = P(\mathbf{x}_0)$.

Solution: Selecting $\mathbf{x}_0 = [0 \quad 2]^\top$, and plugging it into `VanderPolPoincare(x0)` gets us $\mathbf{x}_1 = P(\mathbf{x}_0) = [0.0000 \quad 2.1731]^\top$

(iii) Repeat the above step to obtain a sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_n$, (for a large n) where $\mathbf{x}_{k+1} = P(\mathbf{x}_k)$, i.e. $\mathbf{x}_k = \mathbf{x}(t_k)$ where t_k is the time of the k^{th} intersection of the system solution starting at initial condition x_0.

Solution: Code follows. The sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_n$ for $n = 20$ is for $\mathbf{x}_0 = [0, 2]^\top$, $\mathbf{x}_1 = [0, 2.17311]^\top$, $\mathbf{x}_2 = [0, 2.1703]^\top$, $\mathbf{x}_3, \mathbf{x}_4, \ldots, \mathbf{x}_n = [0, 2.17303]^\top$.

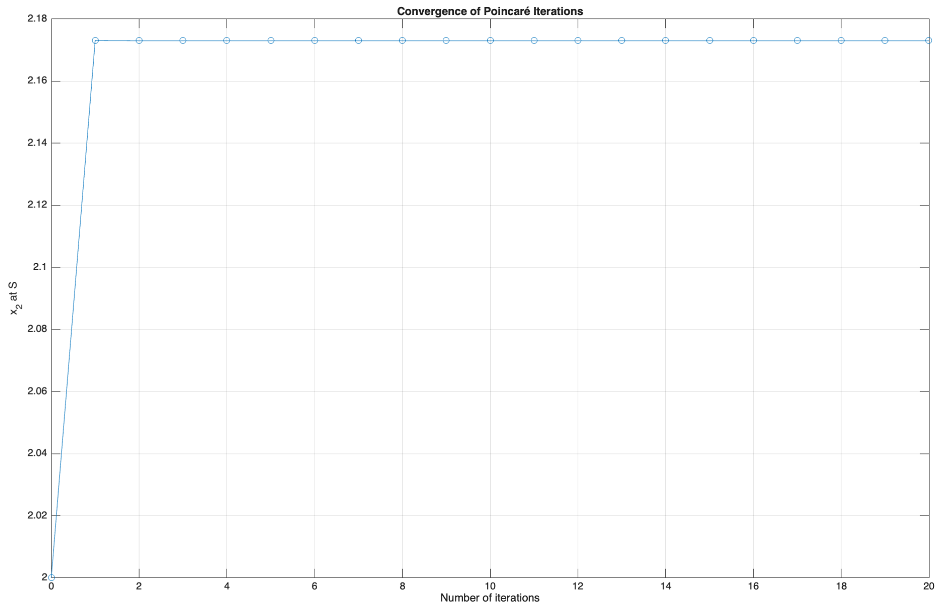
```
% initial condition
x0 = [0; 2]; N = 20;

X = zeros(2,N+1); X(:,1) = x0;

for k = 1:N
    X(:,k+1) = VanderPolPoincare(X(:,k));
end
for k = 1:N+1
    fprintf('%3d %8.5f %8.5f\n', k-1, X(1,k), X(2,k));
end
```

(iv) Plot $x_2(t_k)$ vs. k (Note that $x_1(t_k) \equiv 0$ since $\mathbf{x}_k \in \mathcal{S}$). Does this value of $x_2(t_k)$ settle to any particular value? If it does, then this value of x_2 along with $x_1 = 0$ will be a fixed point of the Poincaré map.

Solution: Visually from the plot, the value of $\mathbf{x}_2(t_k) \to 2.17303$ for an initial condition $x_0 = [0, 2]^\top$. Code below.




```
% initial condition
x0 = [0; 2];

N = 20;
X = zeros(2,N+1);
X(:,1) = x0;

for k = 1:N
    X(:,k+1) = VanderPolPoincare(X(:,k));
end

for k = 1:N+1
    fprintf('%3d %8.5f %8.5f\n', k-1, X(1,k), X(2,k));
end

figure;
plot(0:N, X(2,:), 'o-');
xlabel('Number of iterations'); ylabel('x_2 at S');
title('Convergence of Poincar  Iterations');
grid on;
```

(c) We will next numerically determine the linear approximation of the Poincaré map about the fixed point you found. We have,

$$\mathbf{x}_{k+1} = P(\mathbf{x}_k). \tag{6}$$

Taking the Taylor series expansion around the fixed point $\mathbf{x}^*$ and ignoring the higher order terms, we get

$$\mathbf{x}_{k+1} \approx P(\mathbf{x}^*) + \frac{\partial P}{\partial \mathbf{x}}(\mathbf{x}^*)(\mathbf{x}_k - \mathbf{x}^*), \tag{7}$$

$$\mathbf{x}_{k+1} - \mathbf{x}^* \approx \frac{\partial P}{\partial \mathbf{x}}(\mathbf{x}^*)(\mathbf{x}_k - \mathbf{x}^*), \tag{8}$$

$$\Delta\mathbf{x}_{k+1} \approx A\Delta\mathbf{x}_k, \tag{9}$$

where $\Delta\mathbf{x}_k := (\mathbf{x}_k - \mathbf{x}^*)$ and $A := \frac{\partial P}{\partial \mathbf{x}}(\mathbf{x}^*)$. Thus, (9) is the linear approximation of the Poincaré map locally about the fixed point $\mathbf{x}^*$. This is a discrete-time system and we can assess the (local) stability of the fixed point $\mathbf{x}^*$ by analyzing the eigenvalues of A, i.e. $\mathbf{x}^*$ is locally stable if the magnitude of the eigenvalues of A are less than 1. (Remember that there will be one eigenvalue equal to zero that you will ignore — this arises due to the fact that $P(\mathbf{x}) \in \mathcal{S}$.)

Your task is to compute A numerically. The Euler approximation for computing the derivative of P around y^* is given by the following symmetric difference

$$A_j = \frac{\partial P}{\partial \mathbf{x}_j}(\mathbf{x}^*) \approx \frac{P(\mathbf{x}^* + \delta\mathbf{e}_j) - P(\mathbf{x}^* - \delta\mathbf{e}_j)}{2\delta}, \tag{10}$$

where A_j is the j^{th} column of A, δ is a small scalar denoting perturbation to x^* along the direction $\mathbf{e}_j$. Here, $\mathbf{e}_j$ is a vector with the j^{th} element being one and the rest zeros. Using the `vanderPolPoincare(x0)` function and taking $\delta = 0.01$, compute A and find its eigenvalues (Note that one of the eigenvalues will be zero). Is the limit cycle of the Van der Pol Oscillator stable?

Solution: The A matrix is the Jacobian and is found to be the following with eigenvalues

$$A = \begin{bmatrix} 0 & 0 \\ 0 & 0.0372 \end{bmatrix} \quad \text{with eigenvalues } \lambda_1 = 0.0372, \lambda_2 = 0$$

Since $\lambda_1 = 0.0372 < 1$, the limit cycle of the Van der Pol Oscillator is locally exponentially stable. Code is below.

```
delta = 0.01;
fixed = X(:, end);

e1 = [1; 0]; e2 = [0; 1];

pos_peturb1 = VanderPolPoincare(fixed + delta*e1);
neg_peturb1 = VanderPolPoincare(fixed - delta*e1);
pos_peturb2 = VanderPolPoincare(fixed + delta*e2);
neg_peturb2 = VanderPolPoincare(fixed - delta*e2);

A1 = (pos_peturb1 - neg_peturb1)/(2*delta);
A2 = (pos_peturb2 - neg_peturb2)/(2*delta);

J = [A1, A2]
evals = eig(J)
```

1.3 Code Appendix

```
k  = 20000;
m  = 80;
l0 = 1;
c  = 100;
g  = 9.81;
yd = 2;

reqE = m*g*yd;
init = [5, 4, 3, 2, 1];
figure; hold on;
for i = 1:length(init)
    x0 = [init(i); 0]; t0 = 0;
    t_all = []; x_all = [];
    curr = 'flight';

    while t0 < tf
        if curr ==  flight
            options = odeset('Events', @(t,x) landing(t,x,m,k,c,g,l0,yd,reqE));
            [t,x,te,xe,ie] = ode45(@(t,x) flight_dynamics(t,x,m,k,c,g,l0,yd,reqE), [t0 20], x0, options);
            curr = 'stance';
        else
            options = odeset('Events', @(t,x) takeoff(t,x,m,k,c,g,l0,yd,reqE));
            [t,x,te,xe,ie] = ode45(@(t,x) stance_dynamics(t,x,m,k,c,g,l0,yd,reqE), [t0 20], x0, options);
            curr = 'flight';
        end

        t_all = [t_all; t]; x_all = [x_all; x];
        x0 = xe.';  t0 = te;
    end

    plot(t_all, x_all(:,1),'DisplayName', sprintf('y0=%.1f m',init(i)));
end

yline(yd,'k--','DisplayName','Desired Height');
xlabel('Time (s)'); ylabel('Height (m)');
legend show; grid on;
title('Controlled Hopper');


function dx = flight_dynamics(~,x,~,~,~,g,~,~,~)
    y = x(1); v = x(2);
    dx = [v; -g];
end

function dx = stance_dynamics(~,x,m,k,c,g,l0,~,Ed)
    y = x(1); v = x(2);
    Fs = -k*(y-l0);
    Fc = c*v;

    E = 0.5*m*v^2 + 0.5*k*(y-l0)^2 + m*g*y;
```

```

    K = 1500;
    if v > 0
        u = max(0, K*(Ed - E));
    else
        u = 0;
    end
    F = Fs - Fc + u - m*g;
    dx = [v; F/m];
end

function [value, isterminal, direction] = landing(~,x,~,~,~,~,l0,~,~)
    value = x(1) - l0;
    isterminal = 1;
    direction = -1;
end

function [value, isterminal, direction] = takeoff(~,x,~,~,~,~,l0,~,~)
    value = x(1) - l0;
    isterminal = 1;
    direction = 1;
end
----------------------------------------
%% VAN DER POL OSCILLATOR

%% QUESTION 1

mu = 1;
time = [0 50];

dynamics = @(t,x) [x(2); mu*(1 - x(1)^2)*x(2) - x(1)];

rng(1);
x0s = 10*rand(10,2) - 5;

figure; hold on;
for i = 1:10
    [t, x] = ode45(dynamics, time, x0s(i,:));

    plot(x(:,1), x(:,2));
    plot(x(1,1), x(1,2), 'ko');
end

xlabel('x1'); ylabel('x2');
title('Van der Pol Oscillator');
grid on;

%% QUESTION 2

function x1 = VanderPolPoincare(x0)
    mu = 1;

    % change one of the coords of x0 to ensure x0 in S (Poincar  section)
    x0(1) = 0;
```

```
    if x0(2) <= 0
        x0(2) = abs(x0(2))+ eps;
    end

    dynamics = @(t,x) [x(2); mu*(1 - x(1)^2)*x(2) - x(1)];
    options = odeset('Events', @(t,x) crossings(t,x));

    [~,~,~,xe,~] = ode45(dynamics, [0 100], x0, options);

    x1 = xe(end,:)';
end

function [value, isterminal, direction] = crossings(t,x)
    value = x(1); %detect event when x(1) crosses 0
    isterminal = 1; % stop integration when event occurs
    direction = 1; %detect events whne going from -ve to +ve
end

%% QUESTION 3

% initial condition
x0 = [0; 2];

N = 20;
X = zeros(2,N+1);
X(:,1) = x0;

for k = 1:N
    X(:,k+1) = VanderPolPoincare(X(:,k));
end

for k = 1:N+1
    fprintf('%3d %8.5f %8.5f\n', k-1, X(1,k), X(2,k));
end

figure;
plot(0:N, X(2,:), 'o-');
xlabel('Number of iterations'); ylabel('x_2 at S');
title('Convergence of Poincar  Iterations');
grid on;

%% QUESTION 4
delta = 0.01;
fixed = X(:, end);

e1 = [1; 0]; e2 = [0; 1];

pos_peturb1 = VanderPolPoincare(fixed + delta*e1);
neg_peturb1 = VanderPolPoincare(fixed - delta*e1);
pos_peturb2 = VanderPolPoincare(fixed + delta*e2);
neg_peturb2 = VanderPolPoincare(fixed - delta*e2);

A1 = (pos_peturb1 - neg_peturb1)/(2*delta);
A2 = (pos_peturb2 - neg_peturb2)/(2*delta);
```

```

J = [A1, A2]
evals = eig(J)
```