ESE 5582 Homework 3 / Itterative Nonlinear Optimal Control Approaches

Seif Elkhashab / 04-10-2024

1. Consider the discrete-time nonlinear control system

$$x(k+1) = x(k) + \Delta T \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -x_2(k) & x_1(k) \end{pmatrix} \begin{pmatrix} u_1(k) \\ u_2(k) \end{pmatrix},$$

with ΔT sufficiently small. Recall that one obtains this system from a naive first-order discretization (Euler forward) of the continuous-time nonholonomic integrator system

$$\begin{aligned} \dot{x}_1 &= u_1 \\ \dot{x}_2 &= u_2 \\ \dot{x}_3 &= x_1 u_2 - x_2 u_1. \end{aligned}$$

Implement the two parts, Part 1 (achieving satisfaction of end point constraint) and Part 2 (achieving minimum energy steering while preserving end point constraints) of the iterative control design scheme on the discretized nonholonomic integrator.

Choose ΔT = 0.01, N = 200, x(0) = (0, 0, 0), and x(N) = (0, 0, 2).

-- Visualize the intermediate steps and in each intermediate step, plot the following:

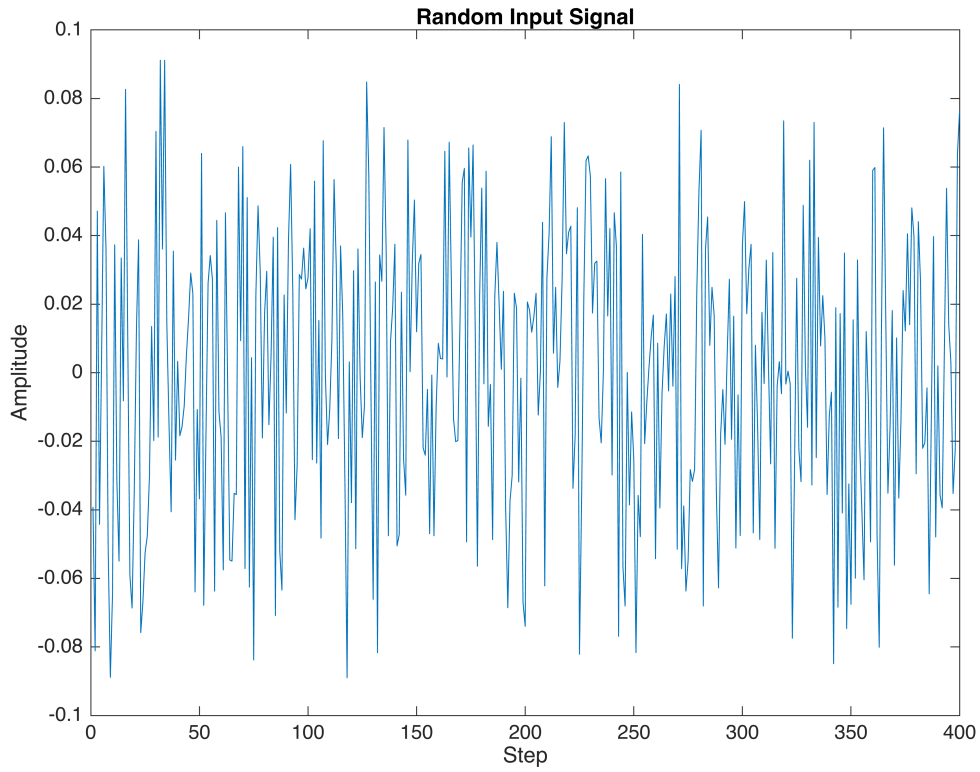
- Nominal state trajectory of the previous step (blue)

- The resulting updated state trajectory obtained from steering the linearized model using the input generated from the linearized model and the quadratic optimization problem (green)

- The resulting updated new state trajectory obtained from steering the actual nonlinear model using the input generated from the linearized model and the quadratic optimization problem (black)

- In a different plot, the input signal of the current iteration.

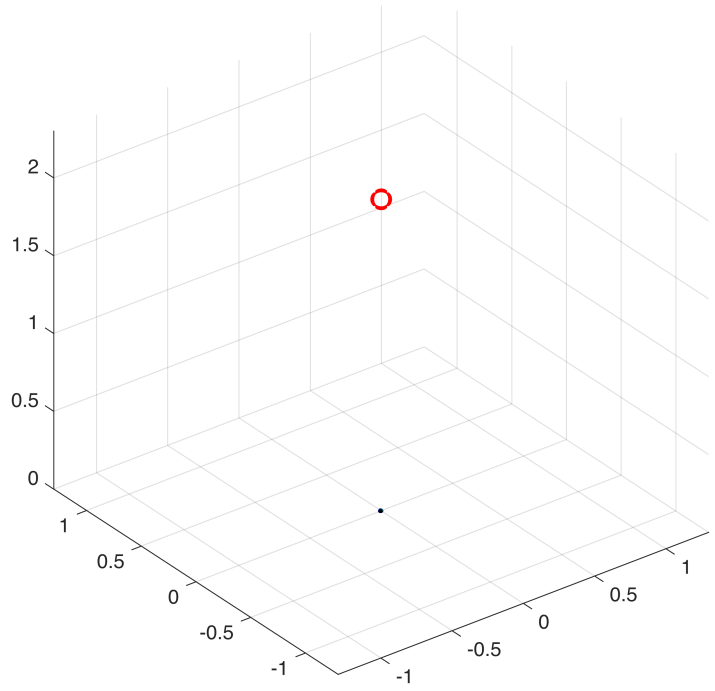
```
% Define simulation parameters
numSteps = 200;
timeStep = 0.01;

% Generate a random input signal
inputSignal = -0.1 * (rand(2 * numSteps, 1) - rand(2 * numSteps, 1));
% x(N) = x(200)
targetPosition = [0; 0; 2];

% Display the input signal
figure;
plot(inputSignal);
title('Random Input Signal');
xlabel('Step');
ylabel('Amplitude');
```

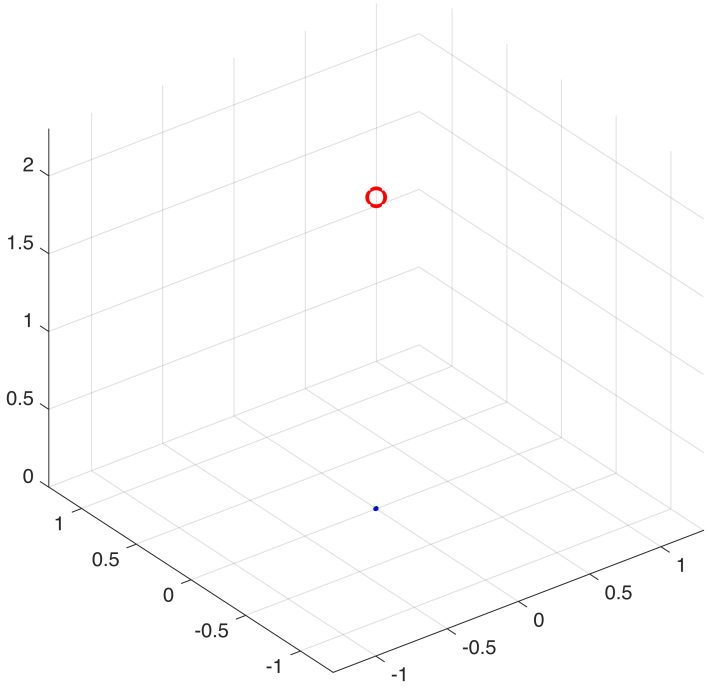


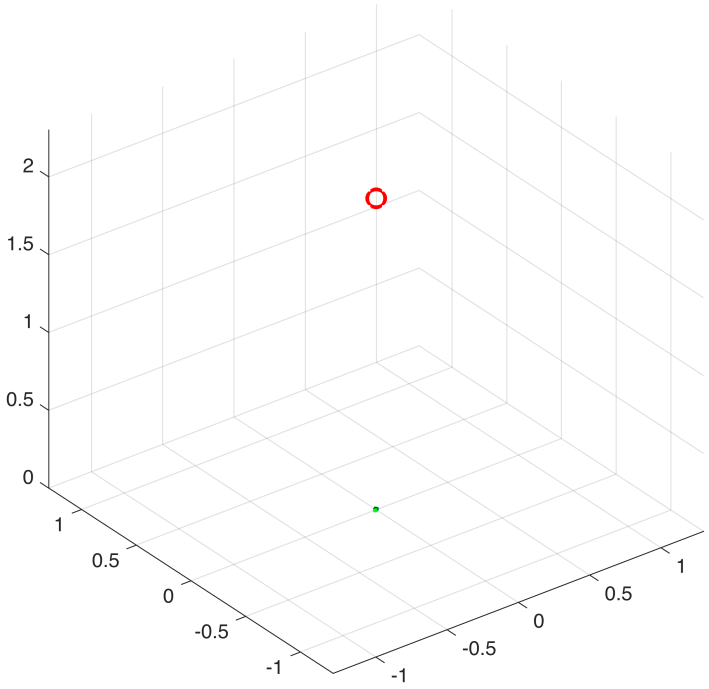

```
% Main iterative process
for iteration = 1:10
    % Initialize system position
    currentPosition = [0; 0; 0];

    % Track the trajectory starting from the initial position
    trajectory1 = currentPosition;
```

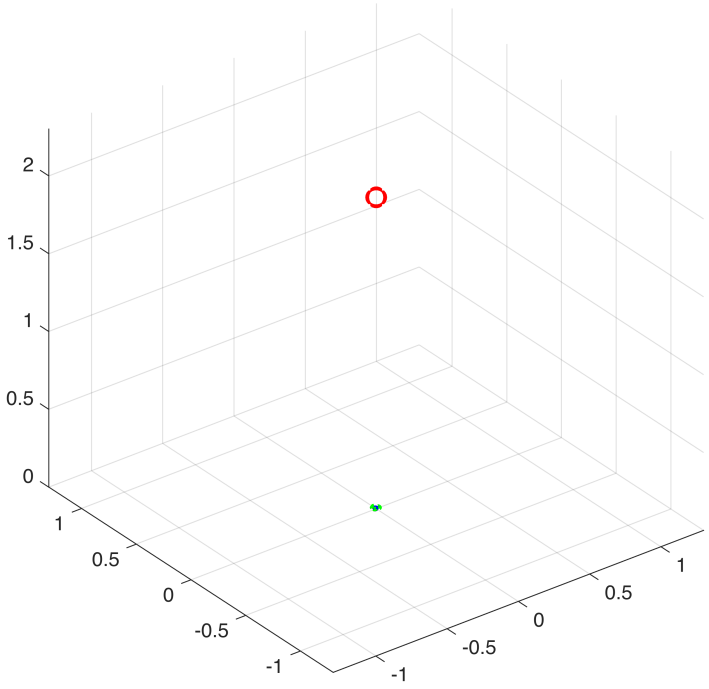
```
    % Update trajectory at each time step
    for step = 1:numSteps
        updateMatrix = [1 0; 0 1; -currentPosition(2) currentPosition(1)];
        currentPosition = currentPosition + timeStep * updateMatrix *
[inputSignal(2 * (step - 1) + 1); inputSignal(2 * (step - 1) + 2)];
        trajectory1 = [trajectory1, currentPosition];
    end

    % Plot the trajectory and target point
    figure;
    plot3(targetPosition(1), targetPosition(2), targetPosition(3), 'ro',
'MarkerSize', 9, 'LineWidth', 2);

    hold on;
    plot3(trajectory1(1, :), trajectory1(2, :), trajectory1(3, :), 'b:',
'LineWidth', 2);

    % Compute correction for the input signal
    controlMatrix = [];
    for step = 1:numSteps
        controlMatrix = [controlMatrix, timeStep * [1 0; 0 1;
-trajectory1(2, step) trajectory1(1, step)]];
    end

    ins(:,iteration) = inputSignal;
    inputSignal = inputSignal - (controlMatrix' * controlMatrix + 0.005 *
eye(2 * numSteps)) \ (controlMatrix' * (currentPosition - targetPosition));

    % Update trajectories with corrected signal and plot
    for color = ['g','k']
        currentPosition = [0; 0; 0];
        trajectory = currentPosition;

        for step = 1:numSteps
            if color == 'k'
                updateMatrix = [1 0; 0 1; -trajectory1(2, step)
trajectory(1, step)];
                currentPosition = currentPosition + timeStep * updateMatrix
* [inputSignal(2 * (step - 1) + 1); inputSignal(2 * (step - 1) + 2)];
            elseif color == 'g'
                currentPosition = currentPosition + timeStep*[inputSignal(2
* (step - 1) + 1); inputSignal(2 * (step - 1) + 2); -inputSignal(2
* (step - 1) + 1)*currentPosition(2) + inputSignal(2 * (step - 1) +
2)*currentPosition(1)];
            end

            trajectory = [trajectory, currentPosition];
        end
```

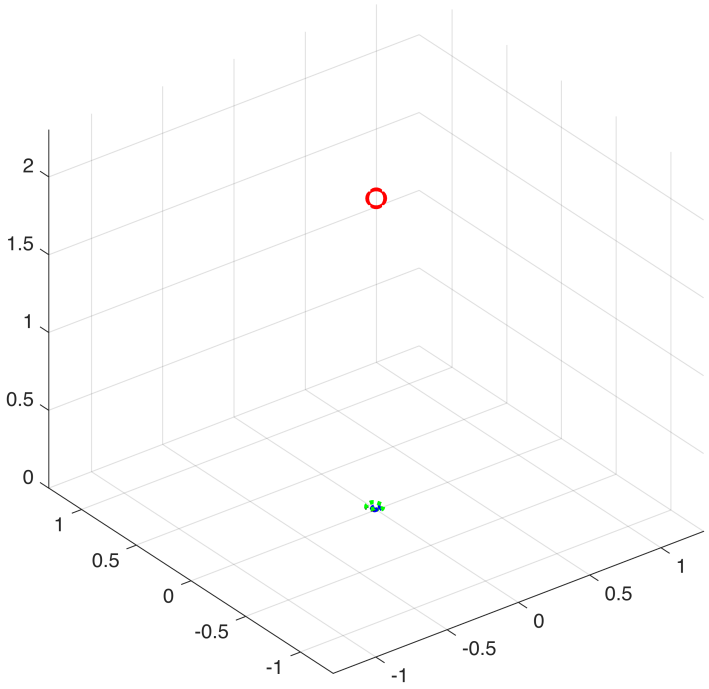
```
        plot3(trajectory(1, :), trajectory(2, :), trajectory(3, :),
'Color', color, 'LineStyle', ':', 'LineWidth', 2);
    end

    axis equal;
    grid on;
    axis([-1.3 1.3 -1.3 1.3 0 2.3]);
end
```

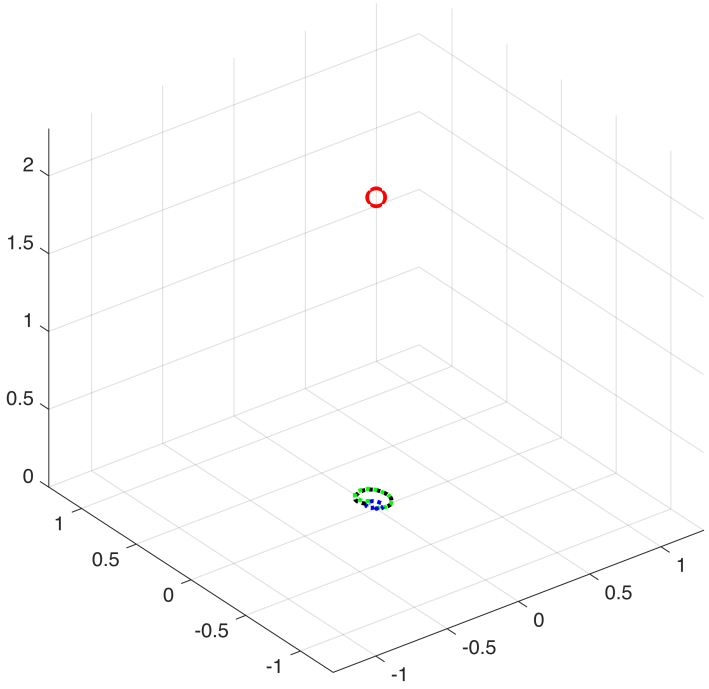


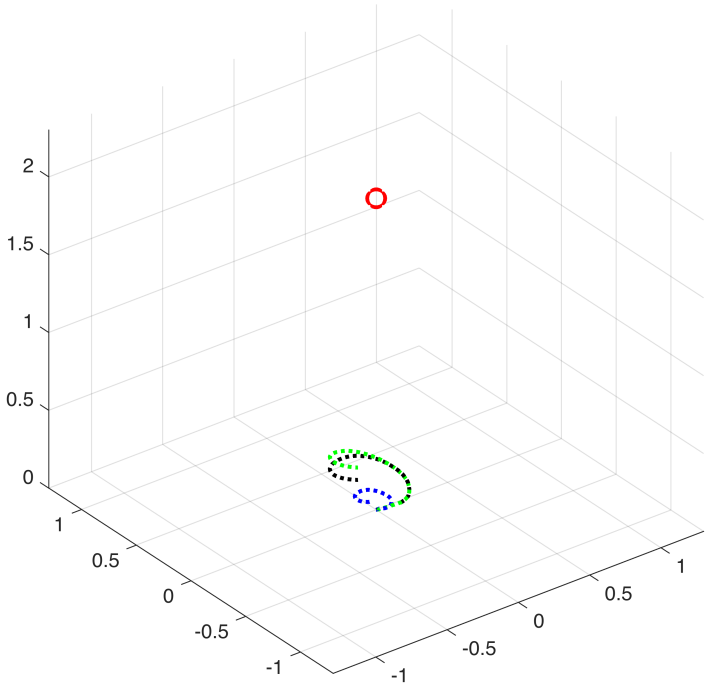

2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

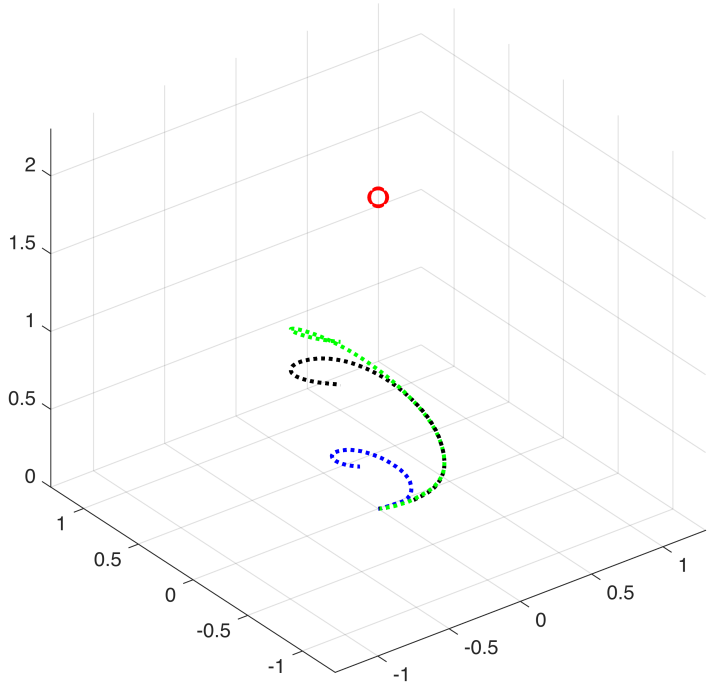

2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

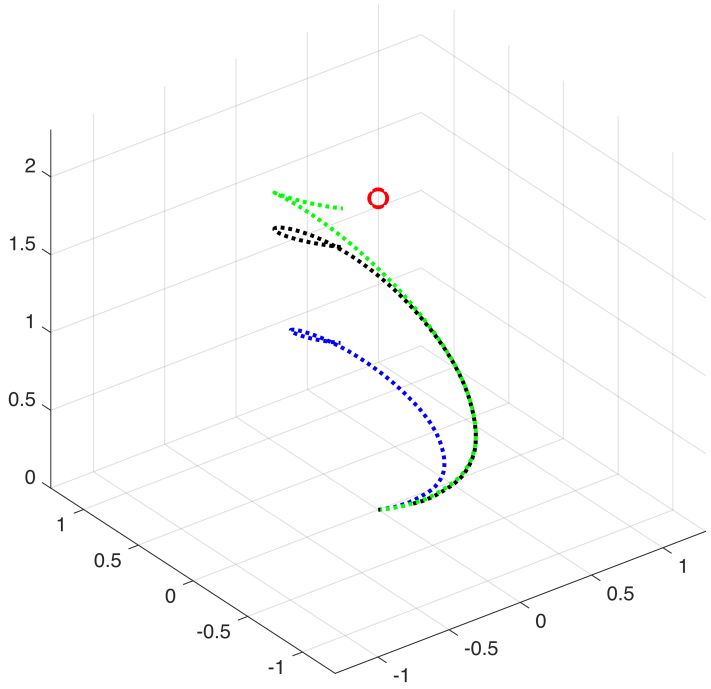

2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

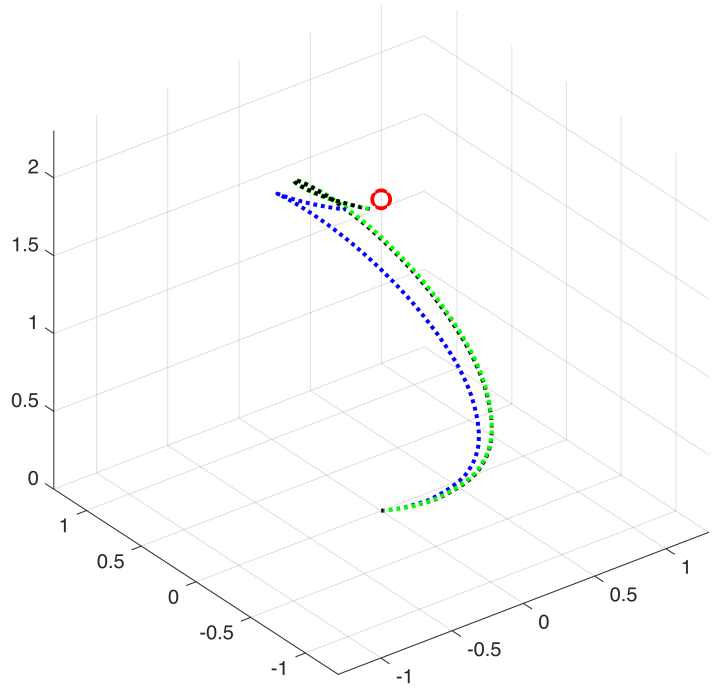

2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1


2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

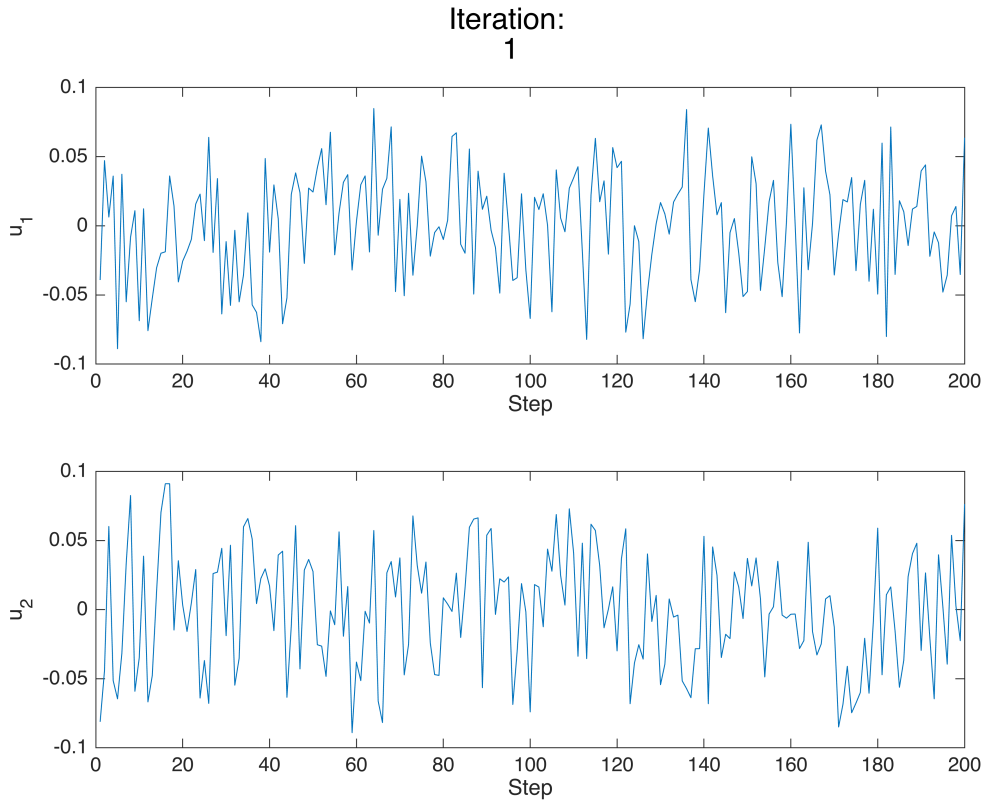

2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

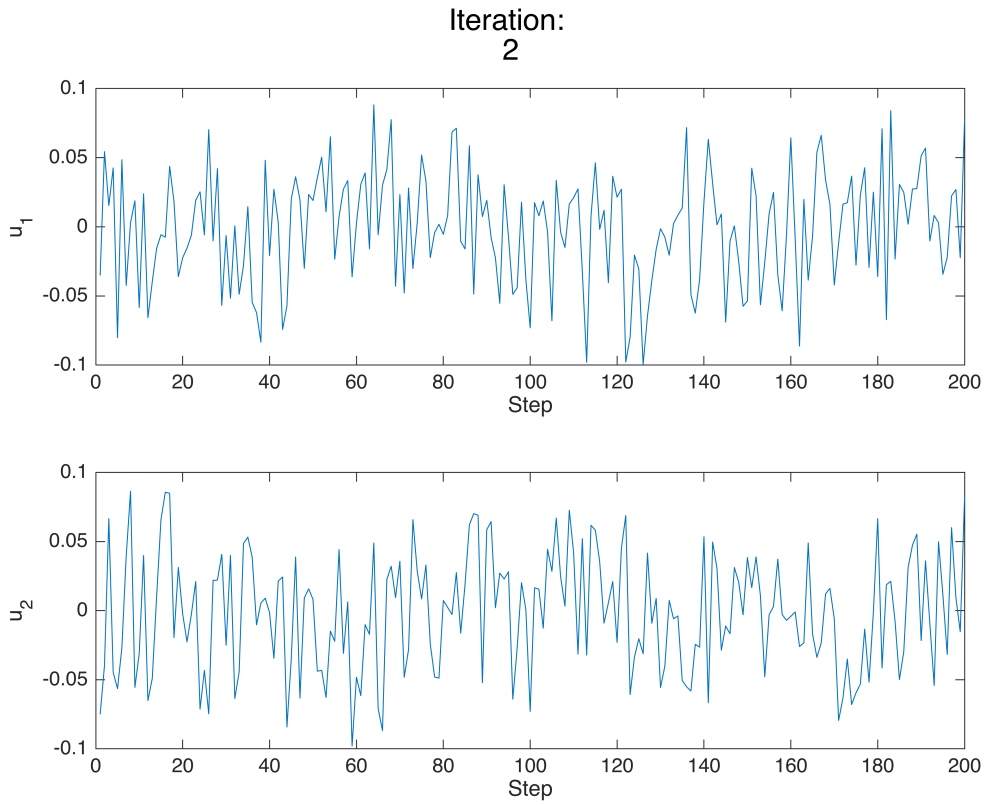

2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
1
0.5
0
-0.5
-1


2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
1
0.5
0
-0.5
-1

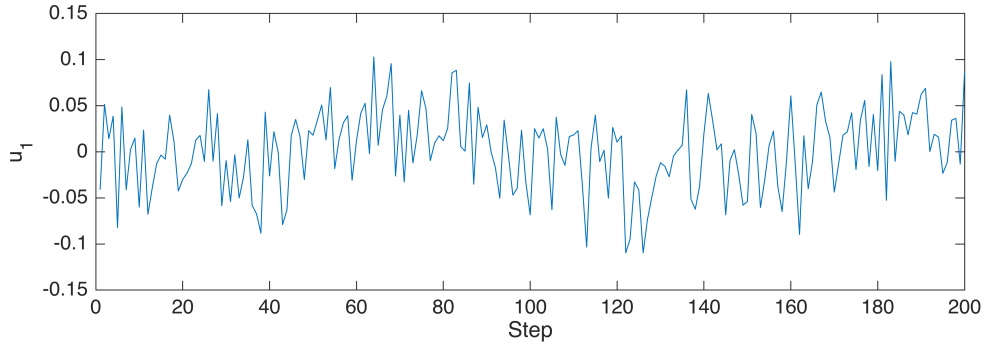


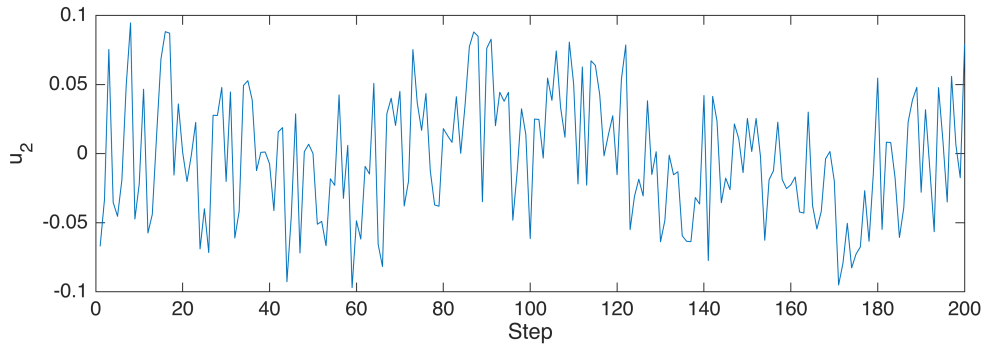

```
step = linspace(1,200,200);
for i = 1:10
    figure;
    sgtitle(['Iteration: ',{i}])
    subplot(2,1,1)
    plot(ins(2 * (step - 1) + 1,i))
    xlabel('Step'), ylabel('u_1')
    subplot(2,1,2)
    plot(ins(2 * (step - 1) + 2,i))
    xlabel('Step'), ylabel('u_2')
end
```


Iteration:
1
0.1
0.05
0
-0.05
-0.1
u_1
0
20
40
60
80
100
120
140
160
180
200
Step
0.1
0.05
0
-0.05
-0.1
u_2
0
20
40
60
80
100
120
140
160
180
200
Step

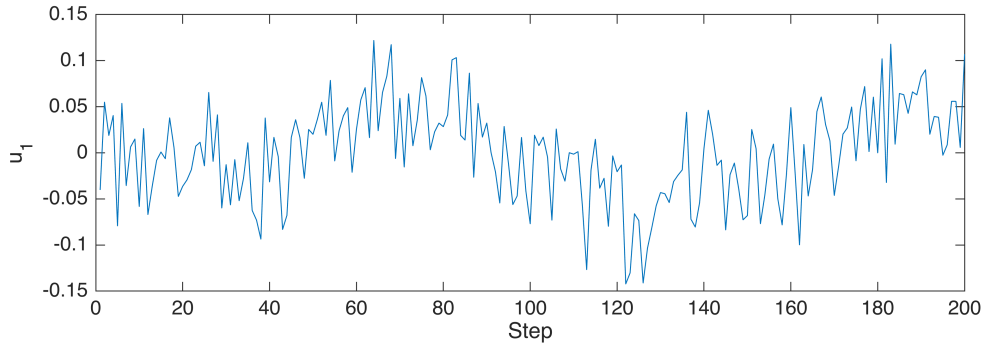

Iteration:
2
0.1
0.05
0
-0.05
-0.1
u_1
0
20
40
60
80
100
120
140
160
180
200
Step
0.1
0.05
0
-0.05
-0.1
u_2
0
20
40
60
80
100
120
140
160
180
200
Step

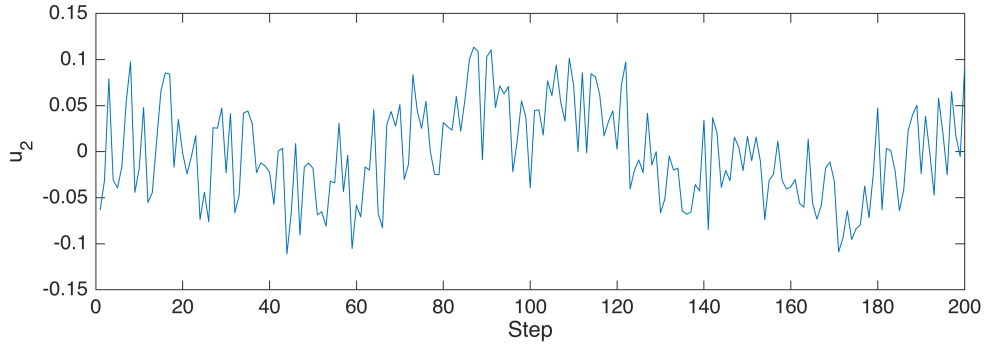
Iteration:
3



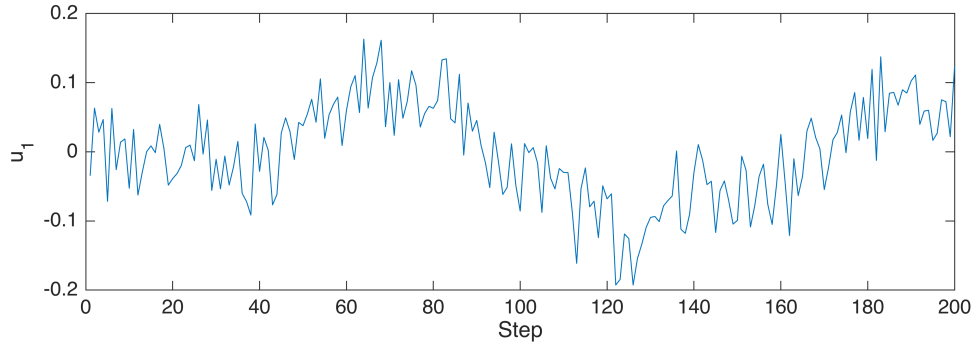


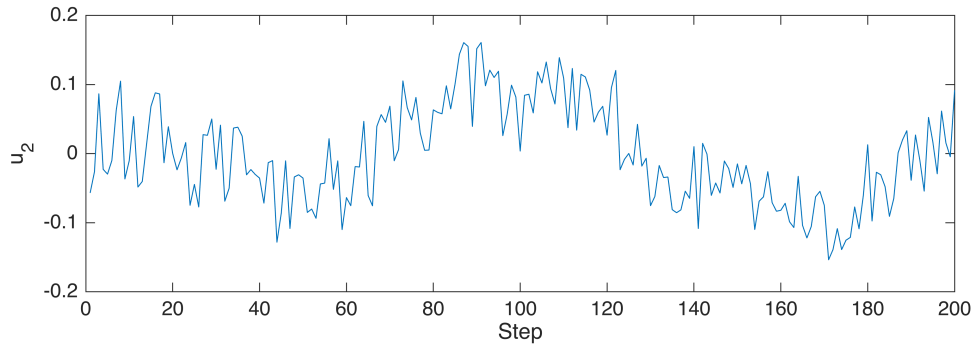
Iteration:
4



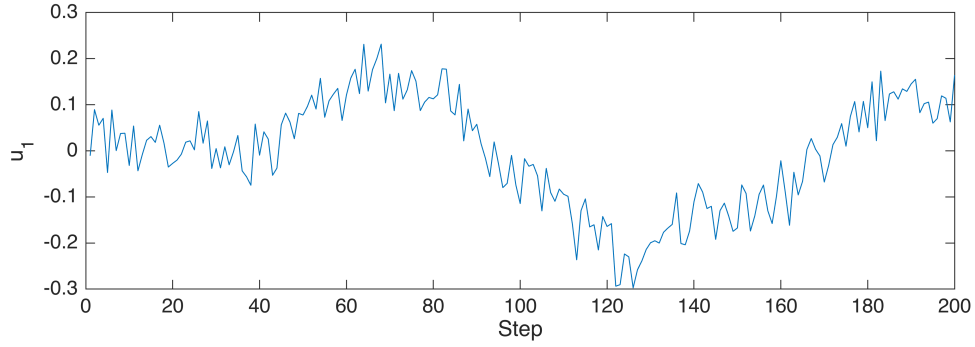


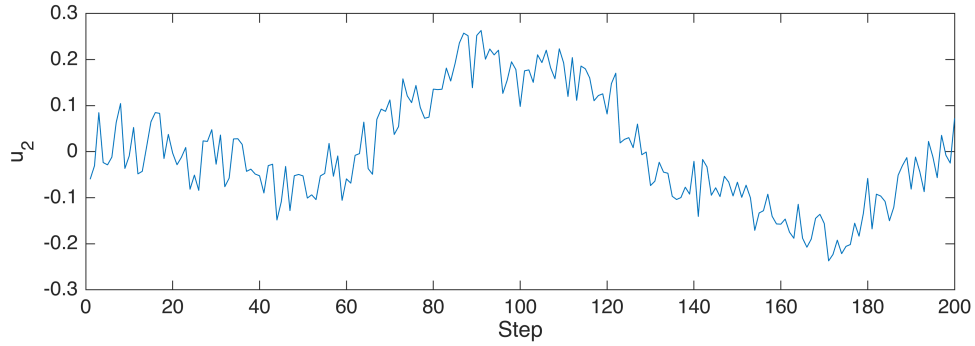
Iteration:
5



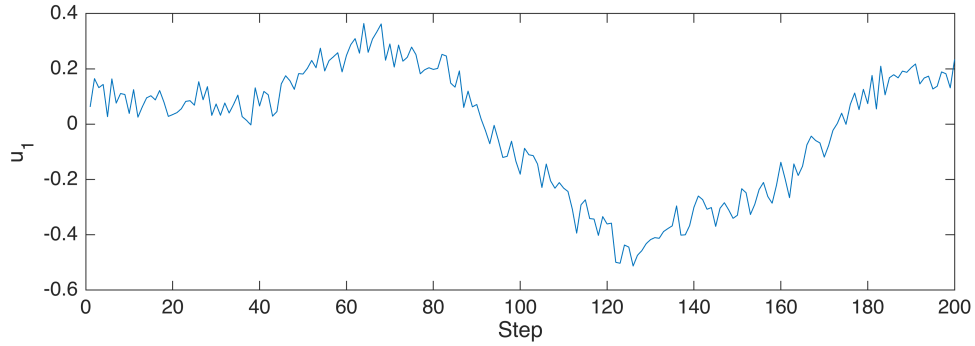


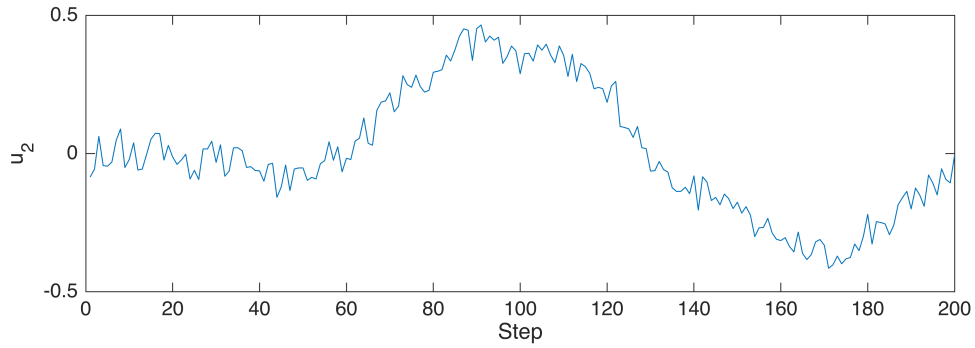
Iteration:
6



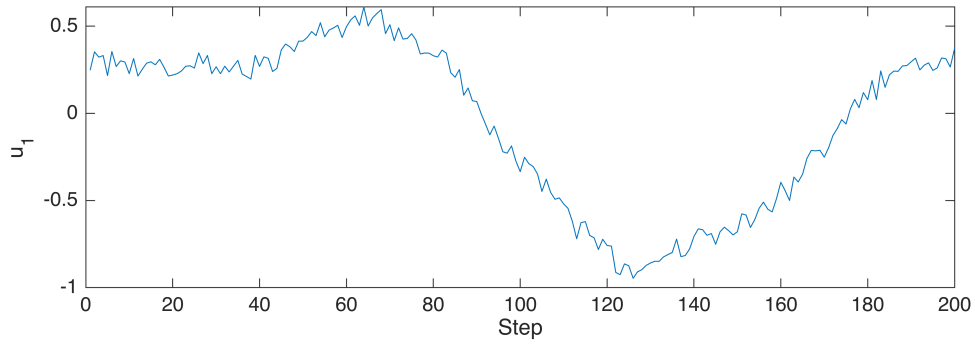


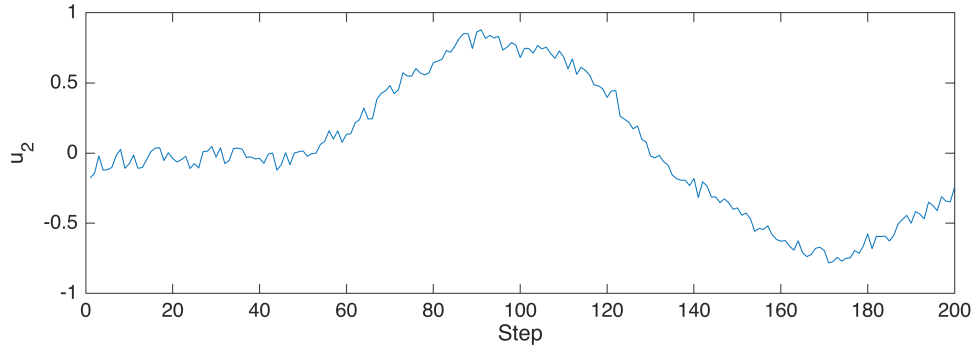
Iteration:
7



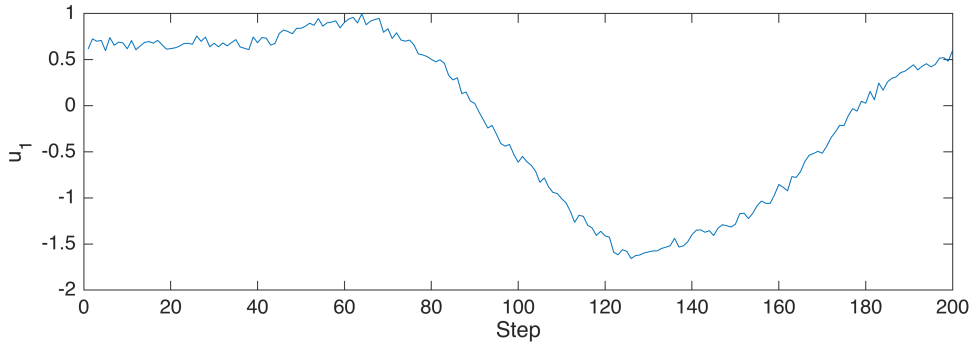


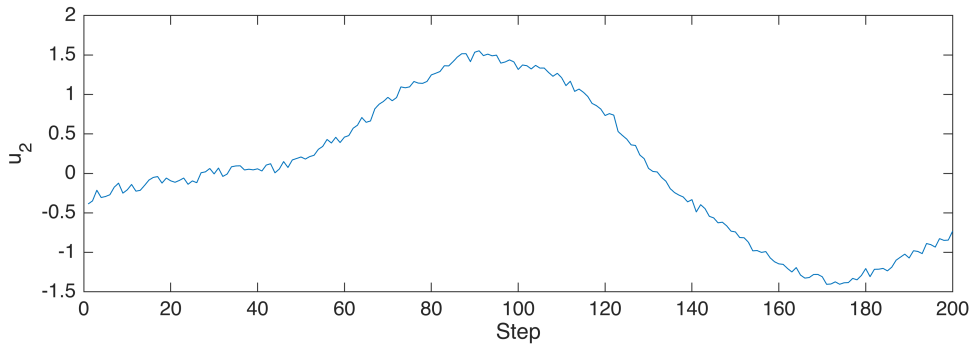
Iteration:
8



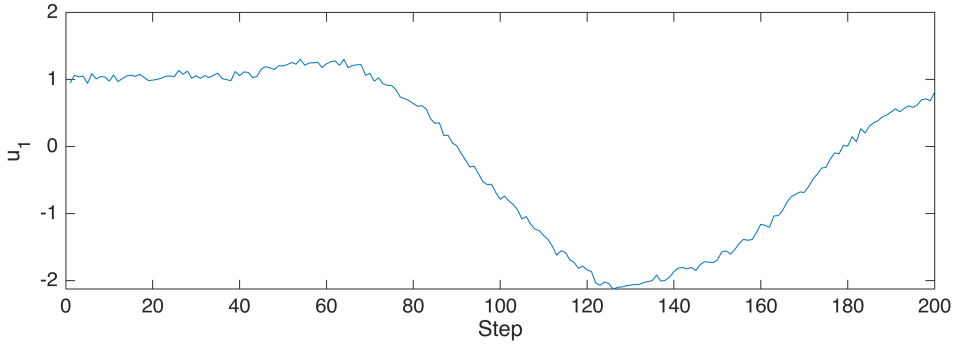


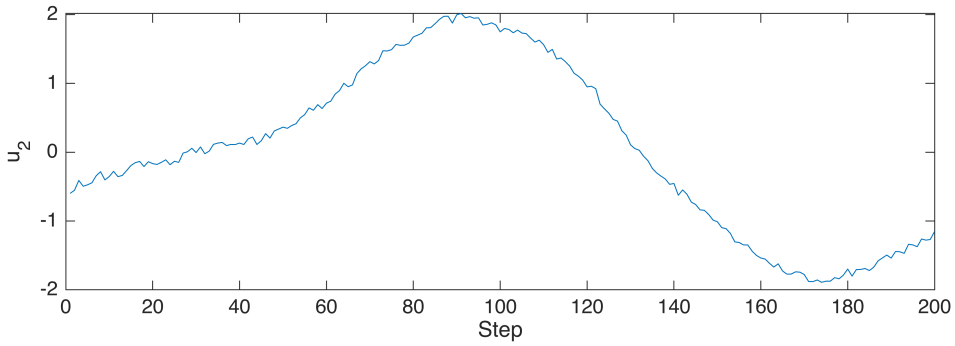
Iteration:
9





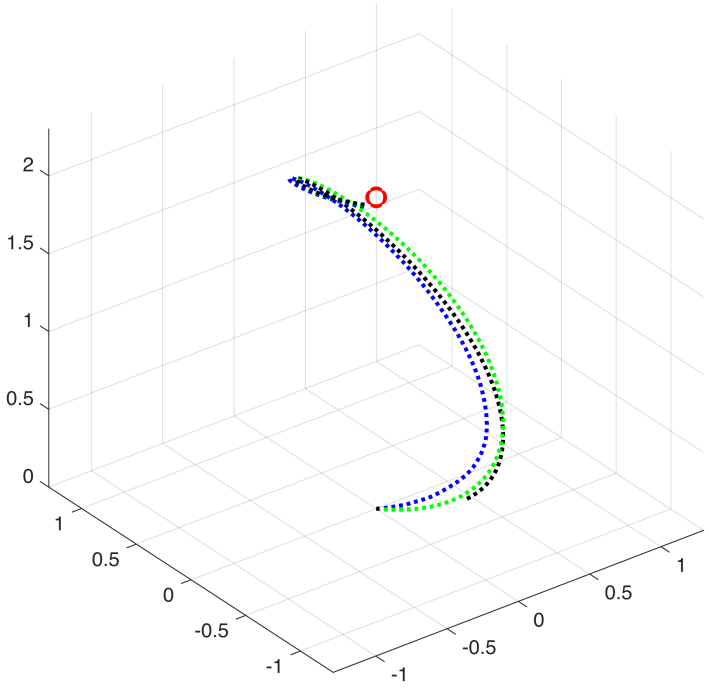
Iteration:
10

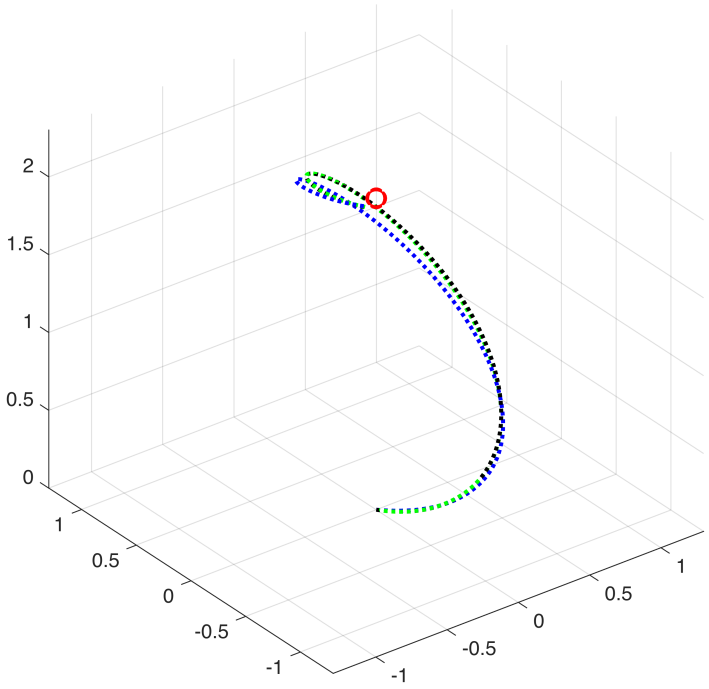


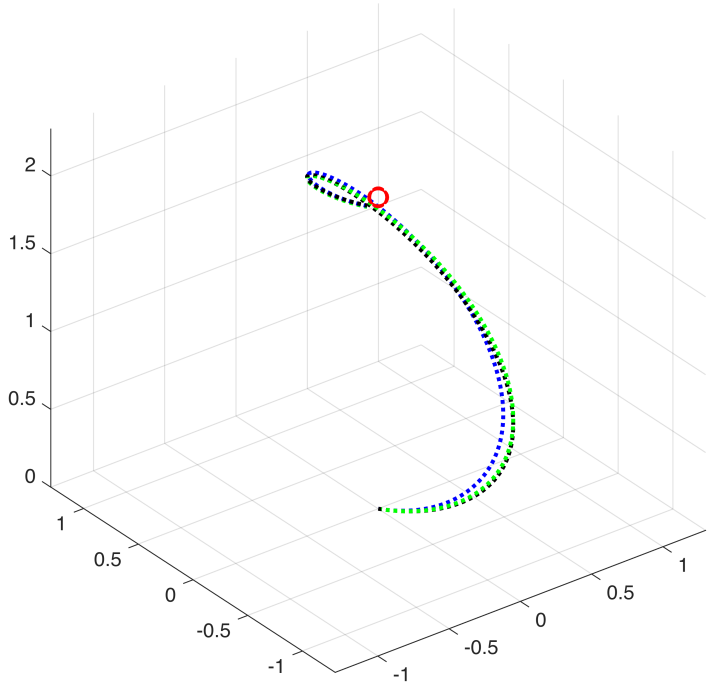


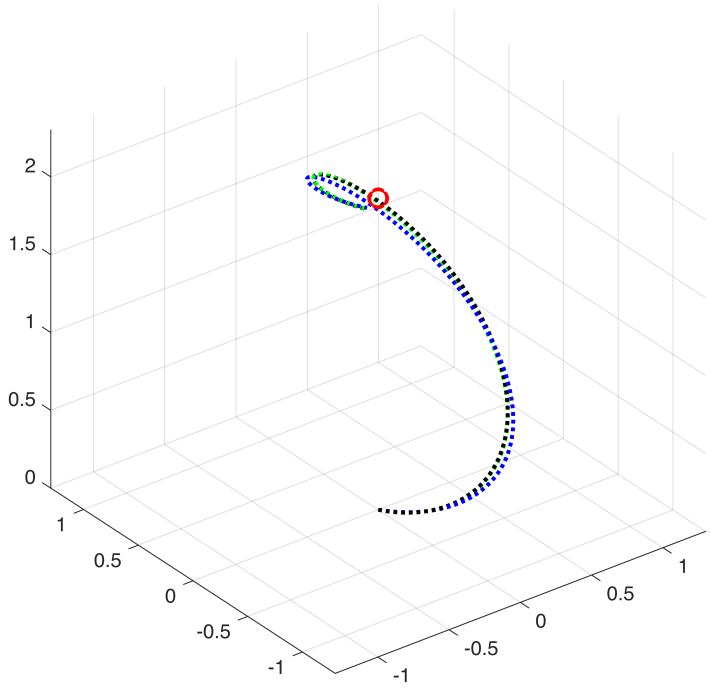
```
options = optimset('Display', 'off');
% Main iterative process
for iteration = 1:10
    % Initialize system position
    currentPosition = [0; 0; 0];

    % Track the trajectory starting from the initial position
    trajectory1 = currentPosition;

    % Update trajectory at each time step
    for step = 1:numSteps
        updateMatrix = [1 0; 0 1; -currentPosition(2) currentPosition(1)];
        currentPosition = currentPosition + timeStep * updateMatrix *
[inputSignal(2 * (step - 1) + 1); inputSignal(2 * (step - 1) + 2)];
        trajectory1 = [trajectory1, currentPosition];
    end

    % Plot the trajectory and target point
    figure;
    plot3(targetPosition(1), targetPosition(2), targetPosition(3), 'ro',
'MarkerSize', 9, 'LineWidth', 2);

    hold on;
    plot3(trajectory1(1, :), trajectory1(2, :), trajectory1(3, :), 'b:',
'LineWidth', 2);

    % Compute correction for the input signal
    controlMatrix = [];
    for step = 1:numSteps
        controlMatrix = [controlMatrix, timeStep * [1 0; 0 1;
-trajectory1(2, step) trajectory1(1, step)]];
    end

    % Optimization
    U = inputSignal;
    gamma = 0.005;
    H_constraint = controlMatrix;

    % The number of variables (size of ΔU)
    n = size(U, 1);

    % Define the H matrix for the quadratic term
    H = (1 + gamma) * eye(n);

    % Define the linear term
    f = U;

    % No inequality constraints
    A = [];
    b = [];
```

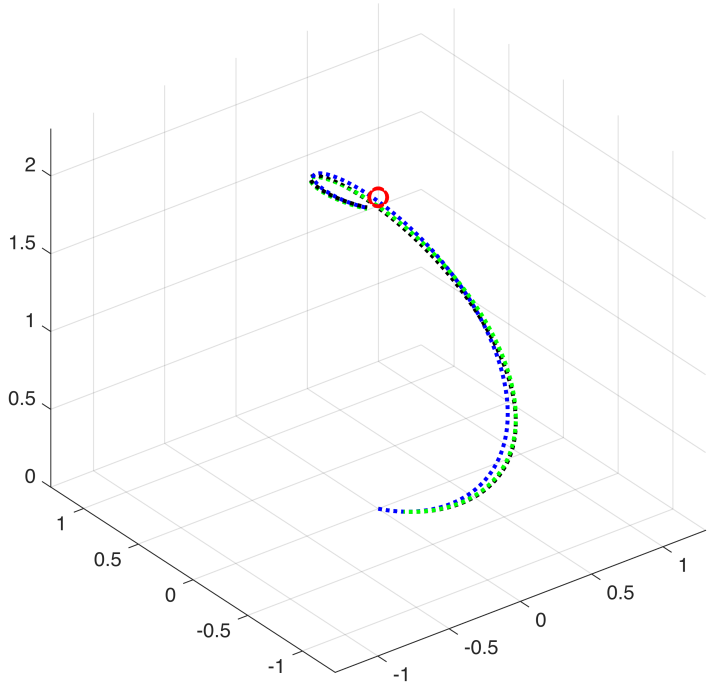
```
    % Equality constraints
    Aeq = H_constraint;
    beq = zeros(size(H_constraint, 1), 1);

    % Lower and upper bounds (assuming none for this problem)
    lb = [];
    ub = [];

    % Solve the quadratic program
    delta_U = quadprog(H, f, A, b, Aeq, beq, lb, ub,(targetPosition -
currentPosition), options);

    % Now delta_U contains the optimized ΔU
    ins(:,iteration) = inputSignal;
    inputSignal = inputSignal + delta_U;

    % Update trajectories with corrected signal and plot
    for color = ['g','k']
        currentPosition = [0; 0; 0];
        trajectory = currentPosition;

        for step = 1:numSteps
            if color == 'k'
                updateMatrix = [1 0; 0 1; -trajectory1(2, step)
trajectory(1, step)];
                currentPosition = currentPosition + timeStep * updateMatrix
* [inputSignal(2 * (step - 1) + 1); inputSignal(2 * (step - 1) + 2)];
            elseif color == 'g'
                currentPosition = currentPosition + timeStep*[inputSignal(2
* (step - 1) + 1); inputSignal(2 * (step - 1) + 2); -inputSignal(2
* (step - 1) + 1)*currentPosition(2) + inputSignal(2 * (step - 1) +
2)*currentPosition(1)];
            end

            trajectory = [trajectory, currentPosition];
        end

        plot3(trajectory(1, :), trajectory(2, :), trajectory(3, :),
'Color', color, 'LineStyle', ':', 'LineWidth', 2);
    end

    axis equal;
    grid on;
    axis([-1.3 1.3 -1.3 1.3 0 2.3]);
end
```

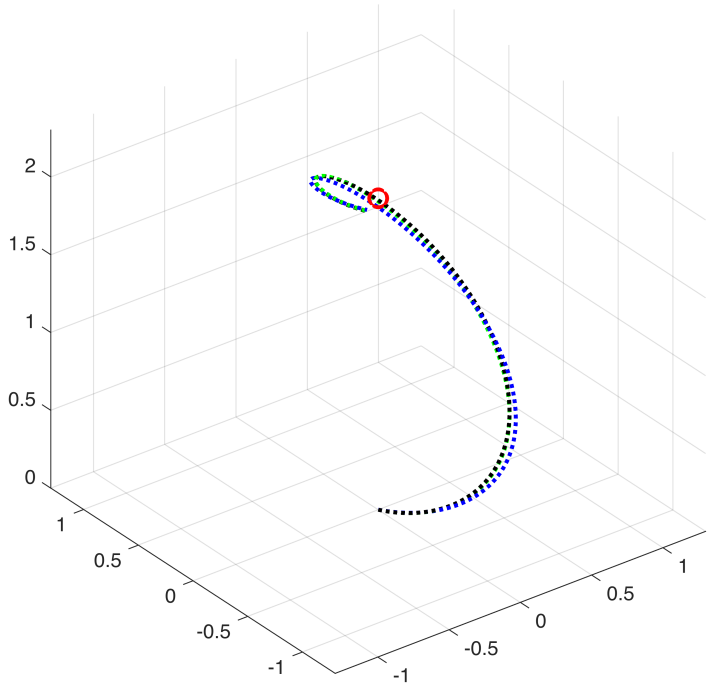

2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

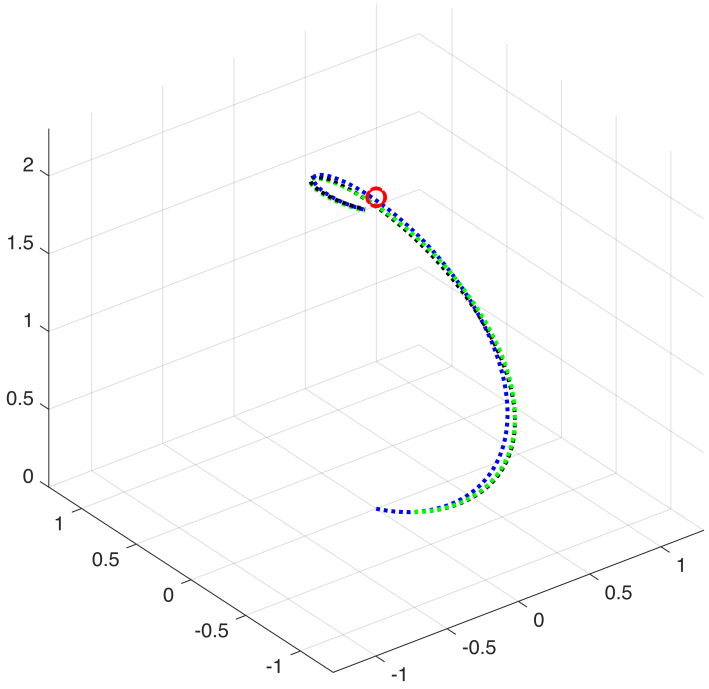

2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

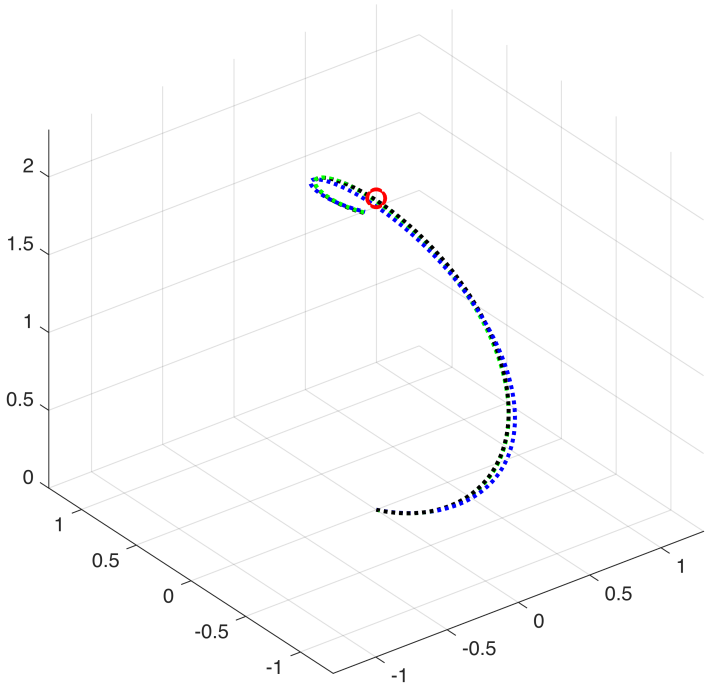

2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

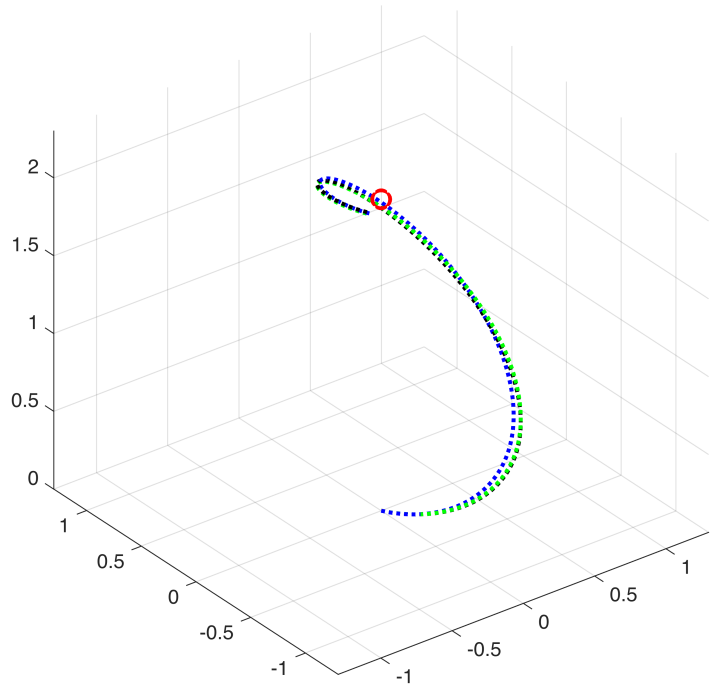

2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

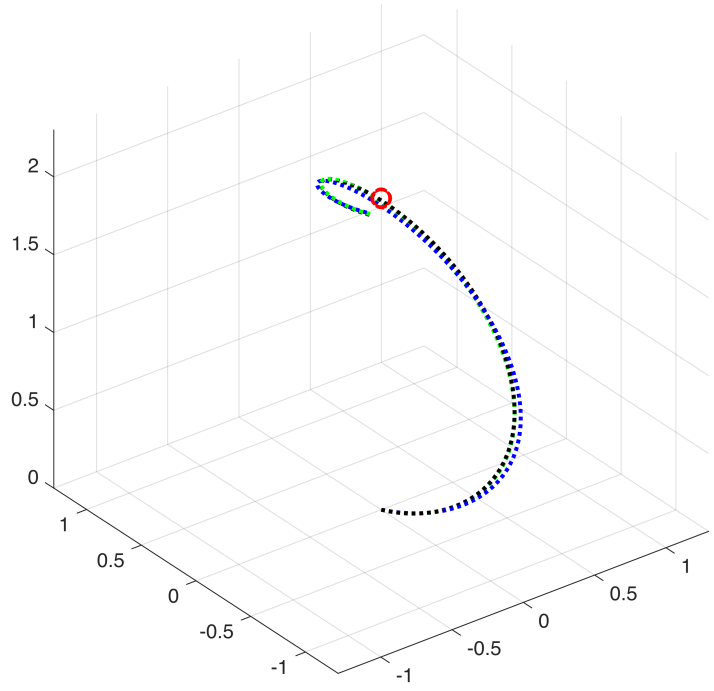

2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1


2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1


2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

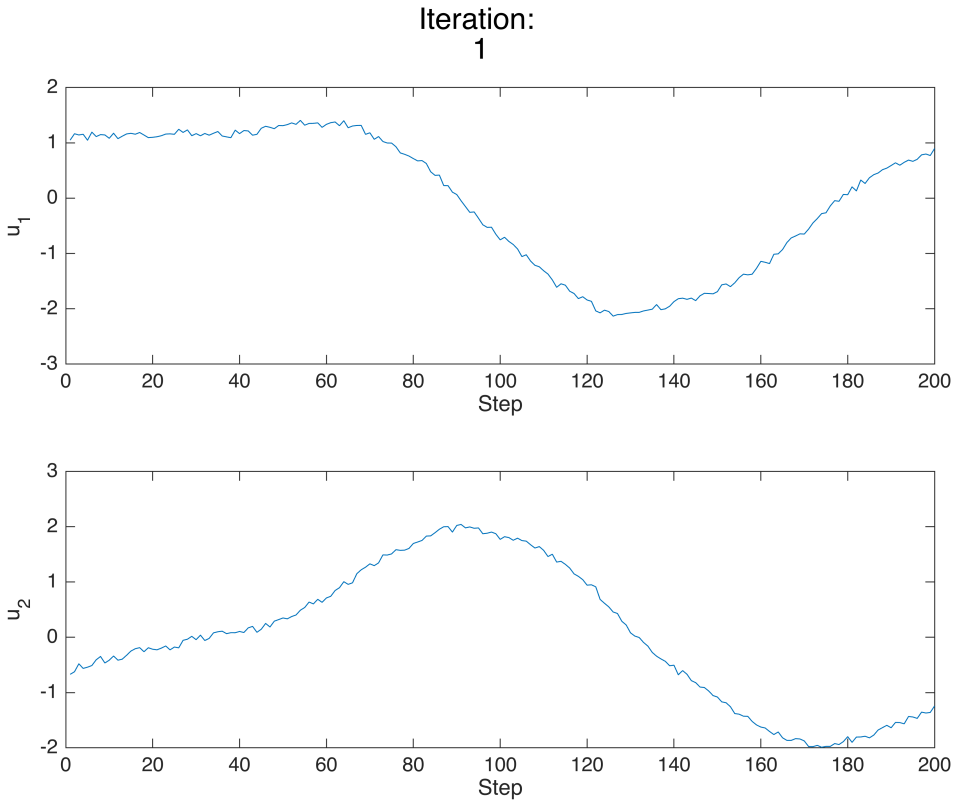

2
1.5
1
0.5
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1



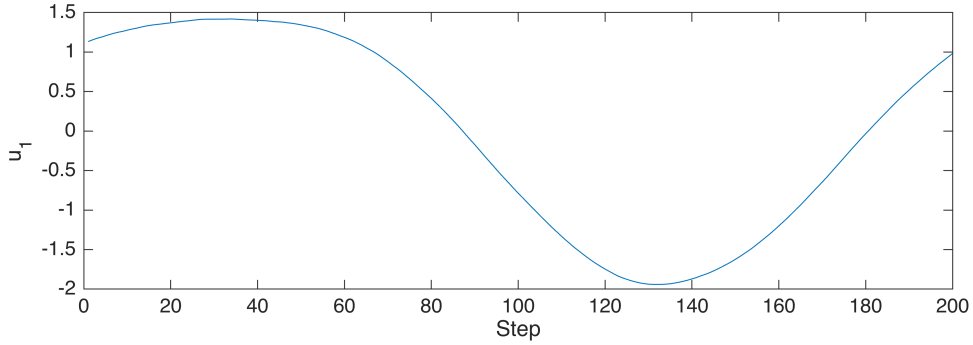


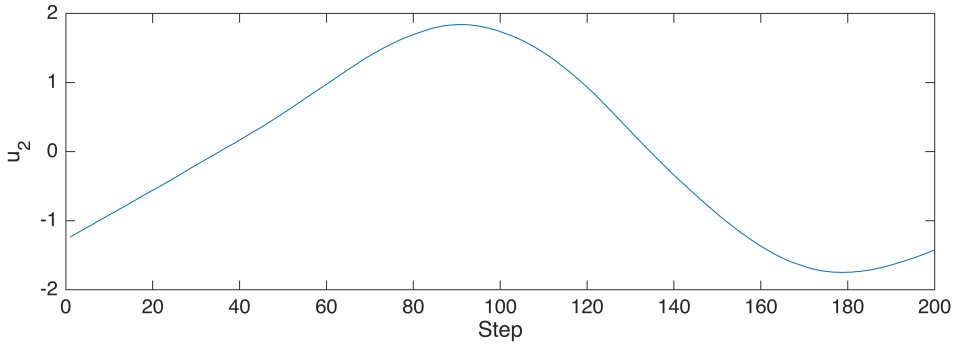

```
step = linspace(1,200,200);
for j = 1:10
    figure;
```

```
    sgtitle(['Iteration: ',{j}])
    subplot(2,1,1)
    plot(ins(2 * (step - 1) + 1,j))
    xlabel('Step'), ylabel('u_1')
    subplot(2,1,2)
    plot(ins(2 * (step - 1) + 2,j))
    xlabel('Step'), ylabel('u_2')
end
```

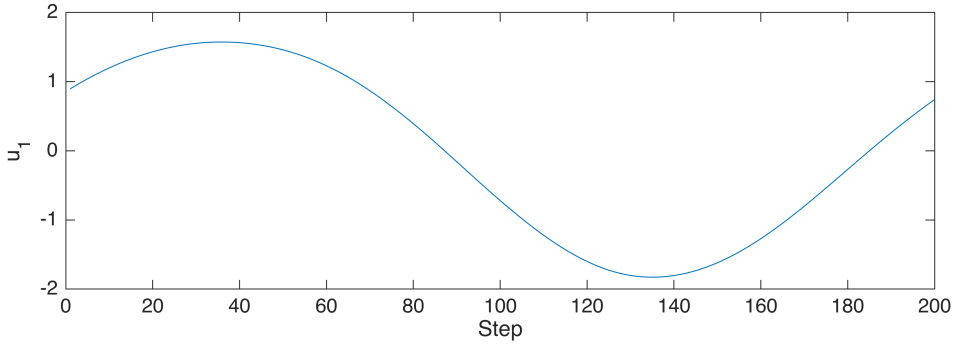


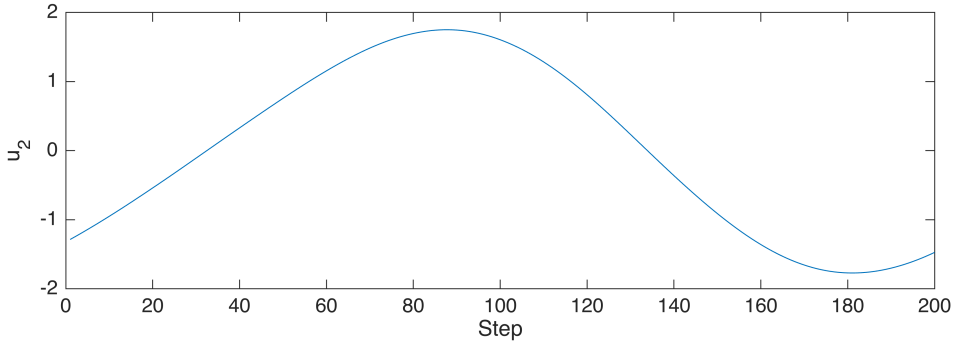
Iteration:
2



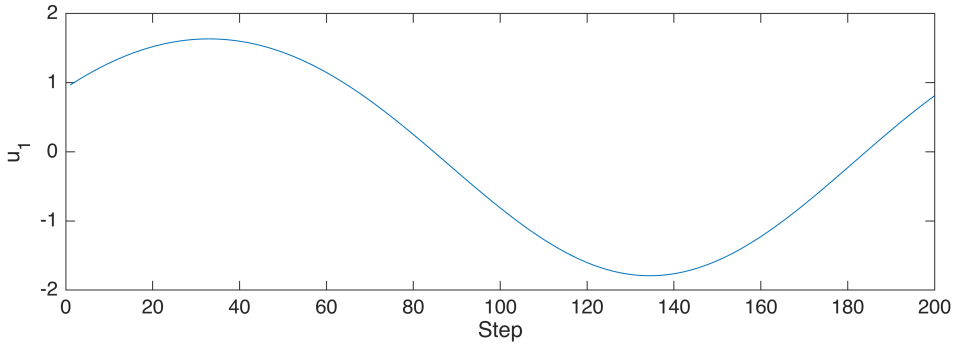


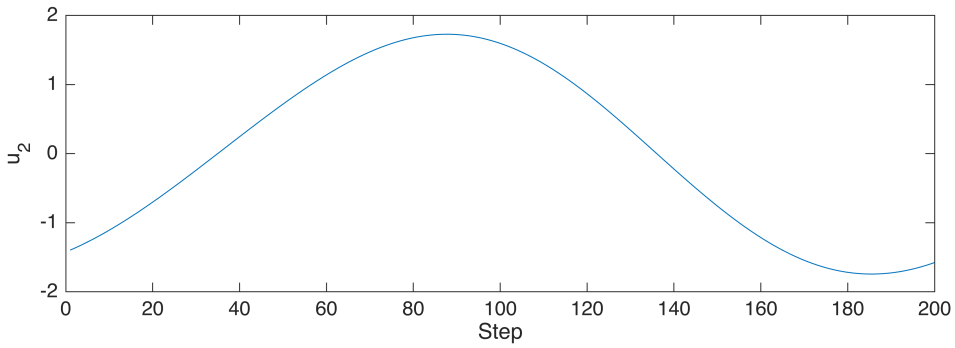
Iteration:
3



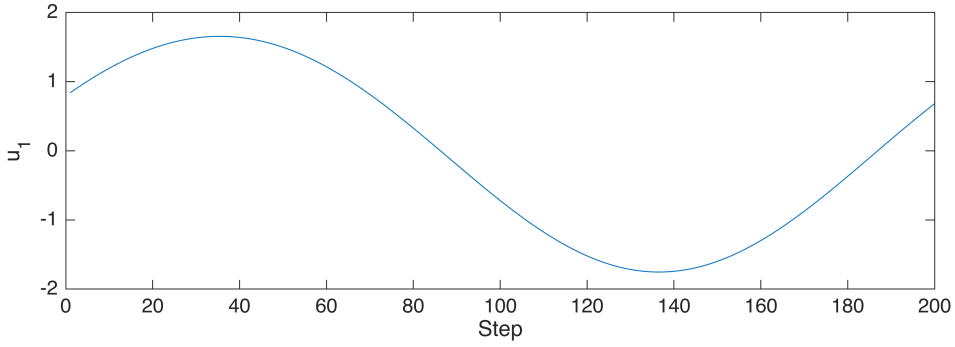


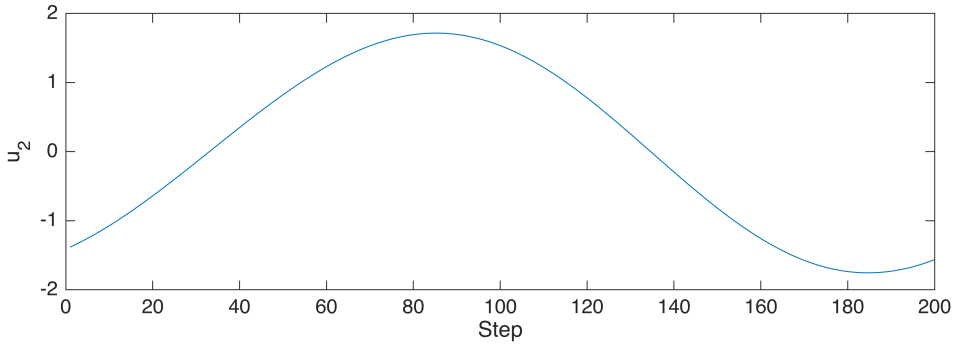
Iteration:
4



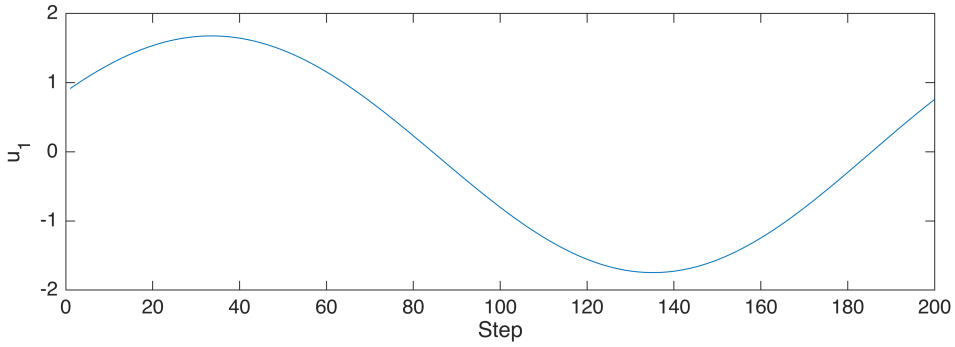


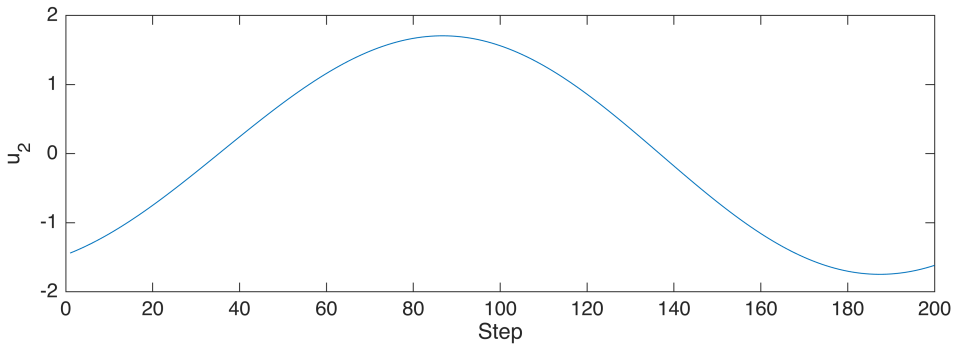
Iteration:
5



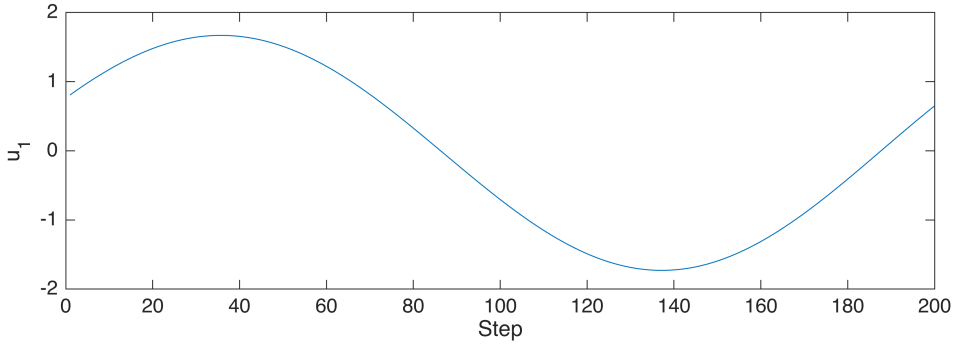


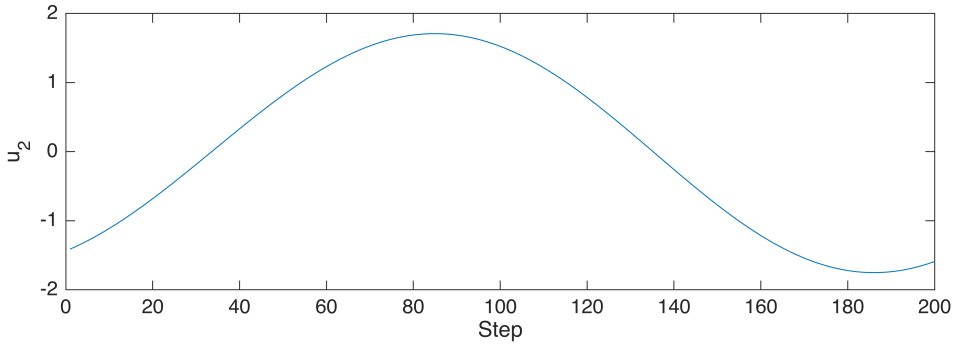
Iteration:
6



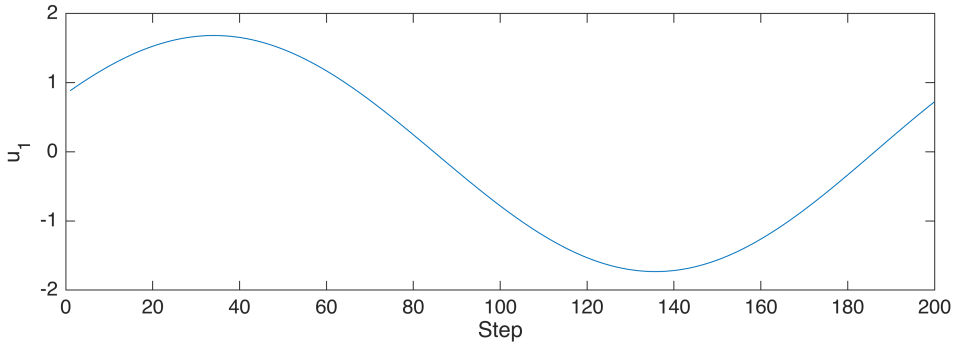


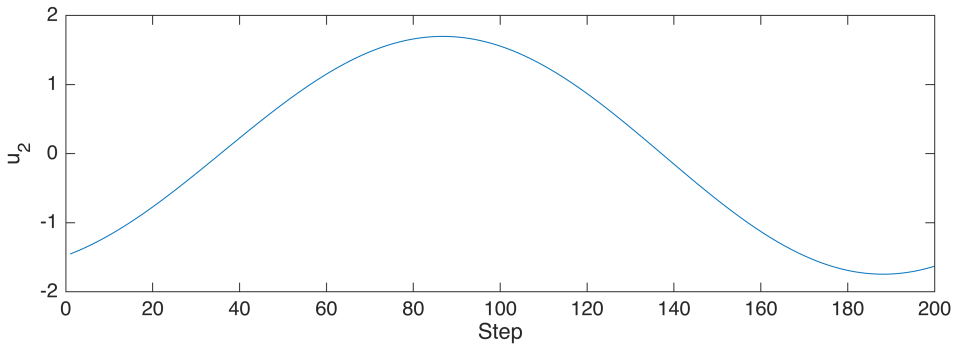
Iteration:
7



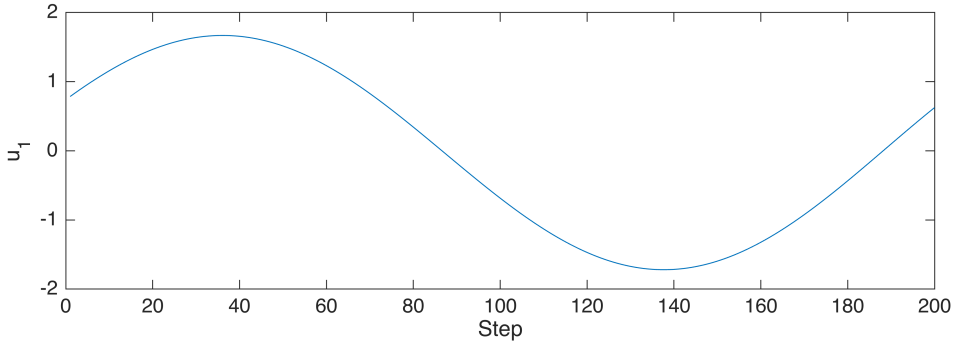


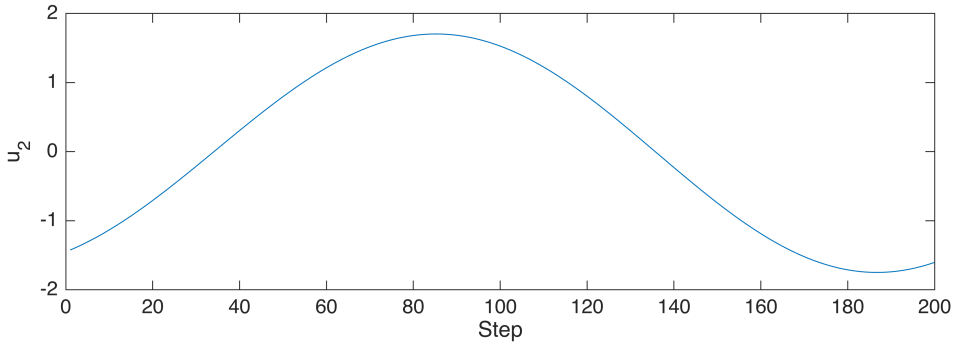
Iteration:
8

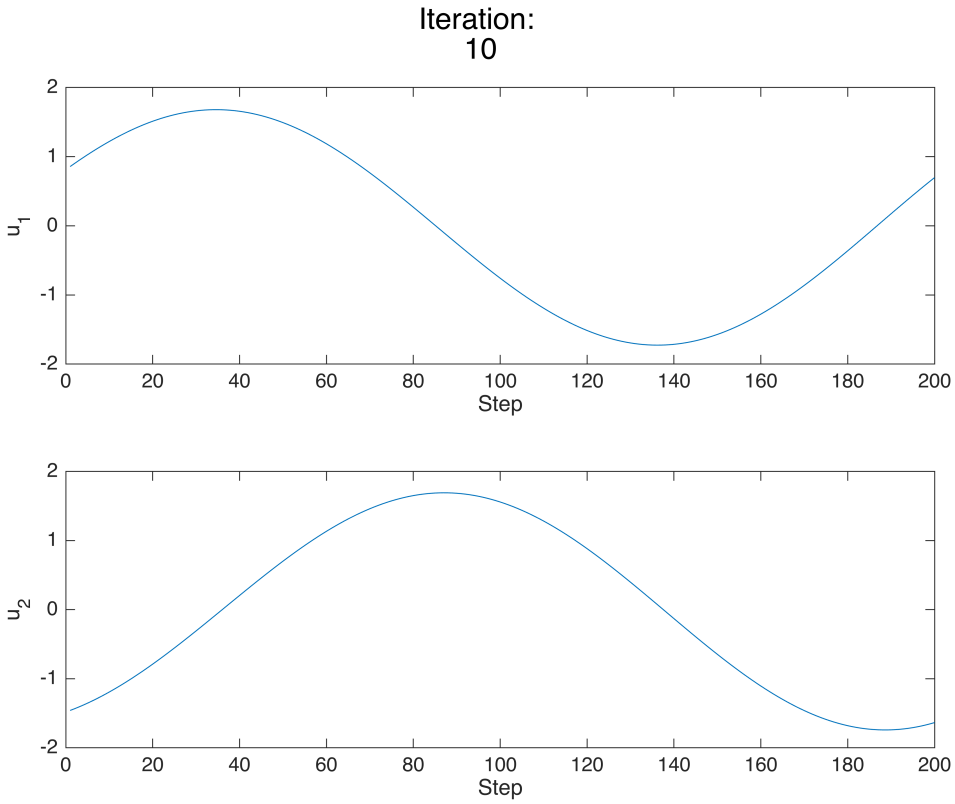




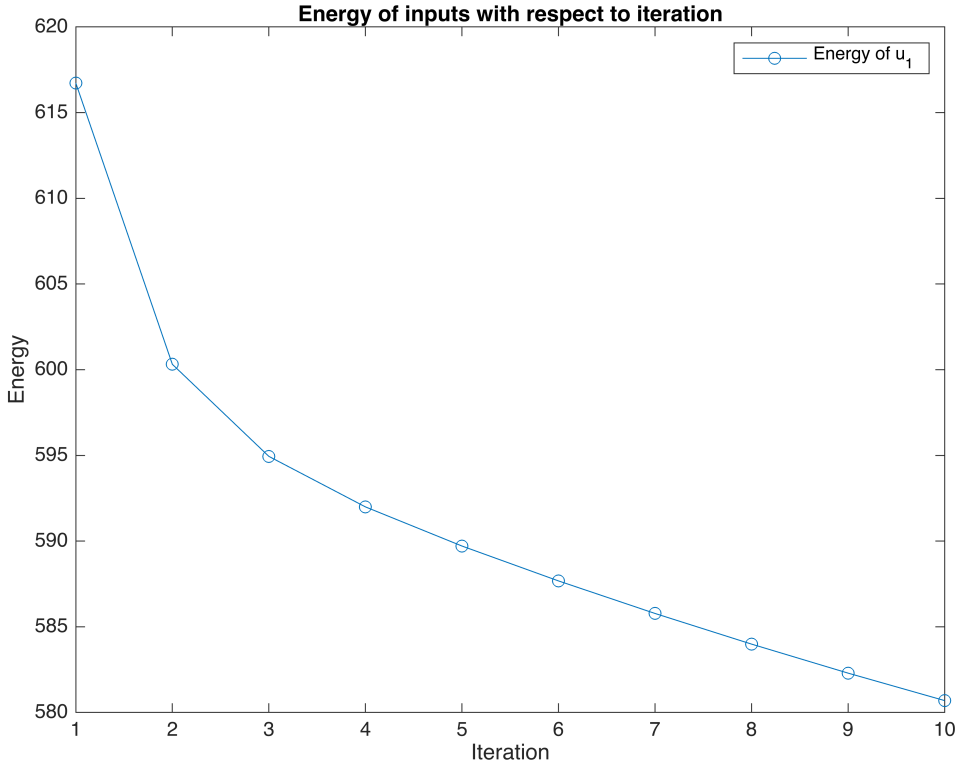
Iteration:
9








```
% Preallocate arrays to hold energy values for each iteration
energy_u = zeros(1, 10);
% Calculate energy for each iteration
for j = 1:10
    energy_u(j) = sum(ins(:,j).^2);
end

% Now plot the energy for each input with respect to iteration
figure;
plot(1:10, energy_u, '-o', 'DisplayName', 'Energy of u_1');
xlabel('Iteration');
ylabel('Energy');
title('Energy of inputs with respect to iteration');
legend show;
```



2. Consider the Van der Pol system with control inputs

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

$$\dot{x}_2 = \left(1 - x_1^2\right)x_2 - x_1 + u$$

(a) Implement the methodology involving the Taylor series expansion and symbolic operations described in the tutorial notes to simulatenously discretize the system in time and obtain linearizations of the controlled Van der Pol system.

```
clear
syms psi_1 psi_2 psi_3
psi = [psi_1; psi_2; psi_3]
```

```
psi =
```

$$\begin{pmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \end{pmatrix}$$

```
F = [psi(2); (1-psi(1)^2)*psi(2)-psi(1)+psi(3); psi(3)]
```

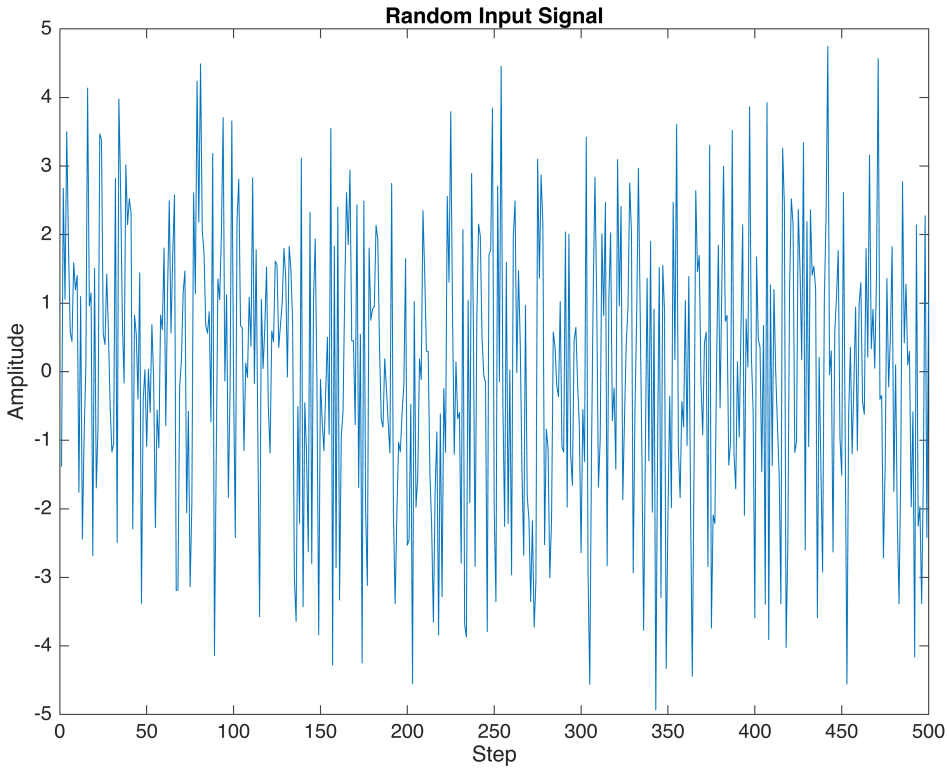
```
F =
```

$$\begin{pmatrix} \psi_2 \\ \psi_3 - \psi_1 - \psi_2 \left(\psi_1^2 - 1\right) \\ \psi_3 \end{pmatrix}$$

```
[Phi,Psi_p,JPhi] = compute_Phi_and_JPhi(5,F,psi,0.01);
```

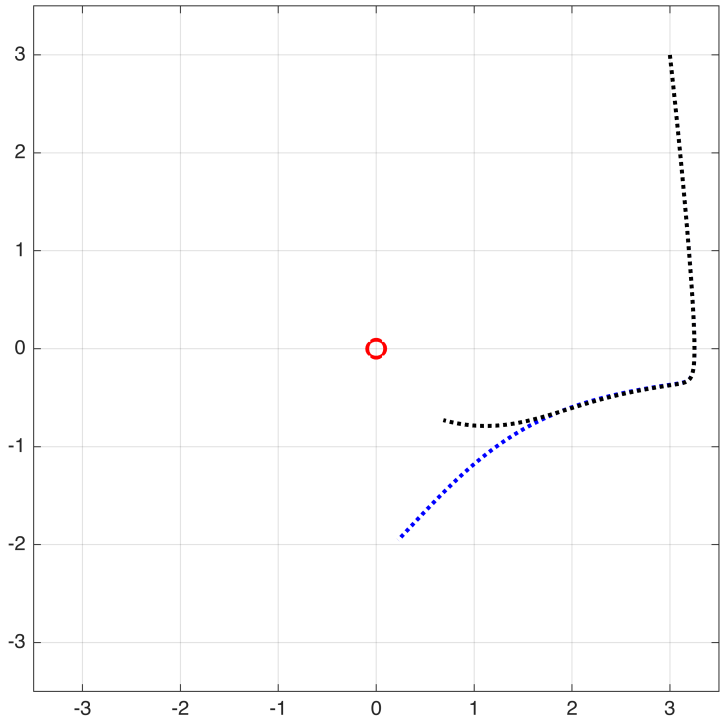
(b) Using the results of part (a) of this problem, implement Part 1 and Part 2 of the iterative control design scheme to steer the system from various (randomly chosen) initial states to the origin, i.e., to solve a nonlinear stabilization task. Visualize the steps similar to how it is shown in the lecture slides.

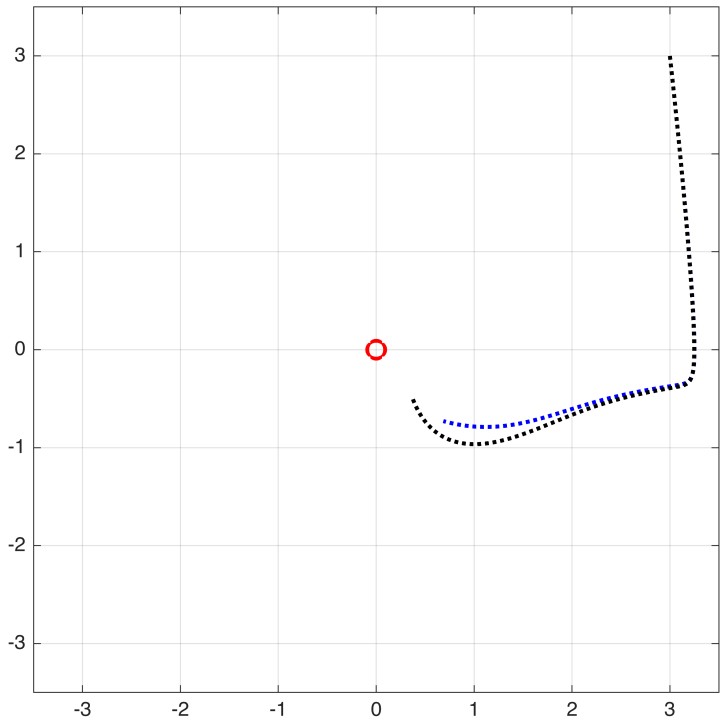
```
% Define simulation parameters
numSteps = 500;
timeStep = 0.01;

% Generate a random input signal
inputSignal = -0.005 * (rand(numSteps, 1) - rand(numSteps, 1));

targetPosition = [0; 0];

% Display the input signal
figure;
plot(inputSignal);
title('Random Input Signal');
xlabel('Step');
ylabel('Amplitude');
```

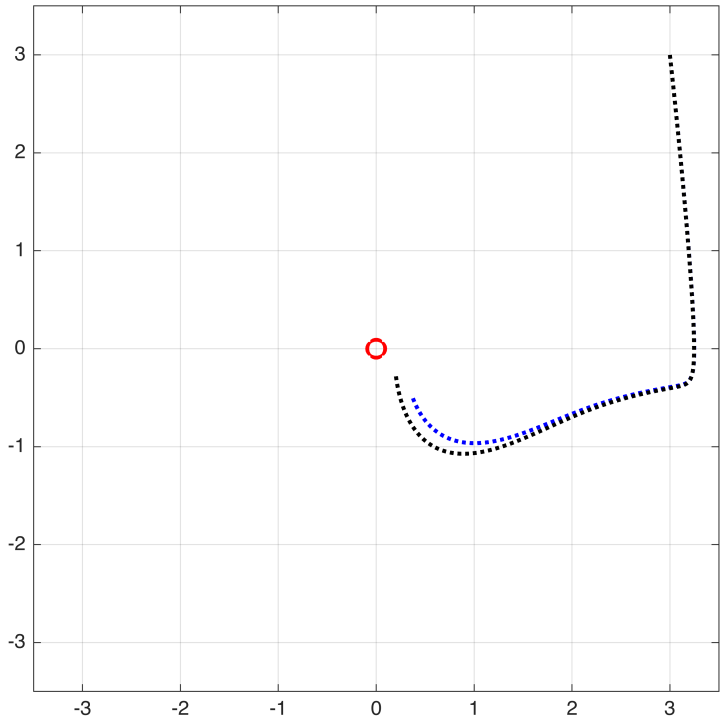


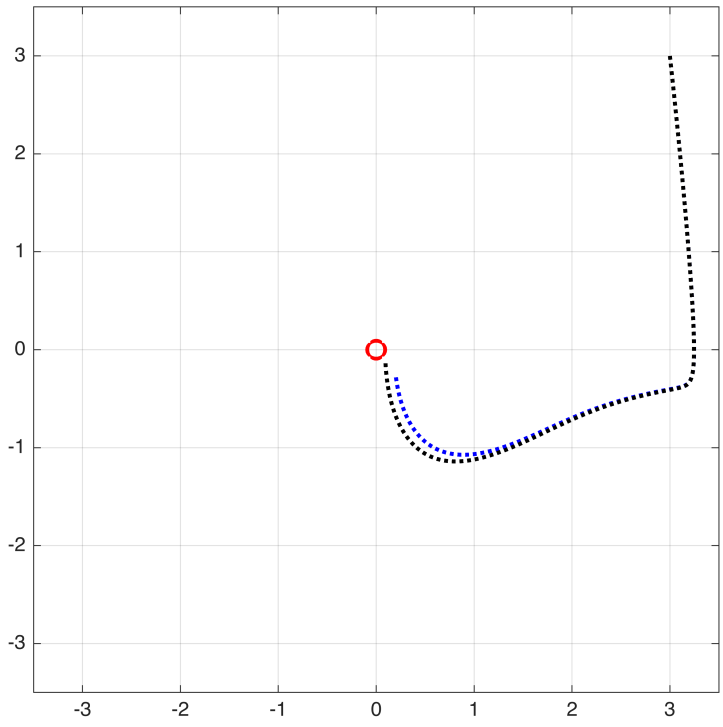

```
% Main iterative process
for iteration = 1:10
    % Initialize system position
    currentPosition = [3; 3];

    % Track the trajectory starting from the initial position
    trajectory1 = [currentPosition; inputSignal(1)];

    % Update trajectory at each time step
    for step = 1:numSteps
        currentPosition =
Phi(currentPosition(1),currentPosition(2),inputSignal(step));
        trajectory1 = [trajectory1, currentPosition];
    end

    % Plot the trajectory and target point
    figure;
    plot(targetPosition(1), targetPosition(2), 'ro', 'MarkerSize', 9,
'LineWidth', 2);

    hold on;
    plot(trajectory1(1, :), trajectory1(2, :), 'b:', 'LineWidth', 2);


    % Compute correction for the input signal
    controlMatrix = [];
```

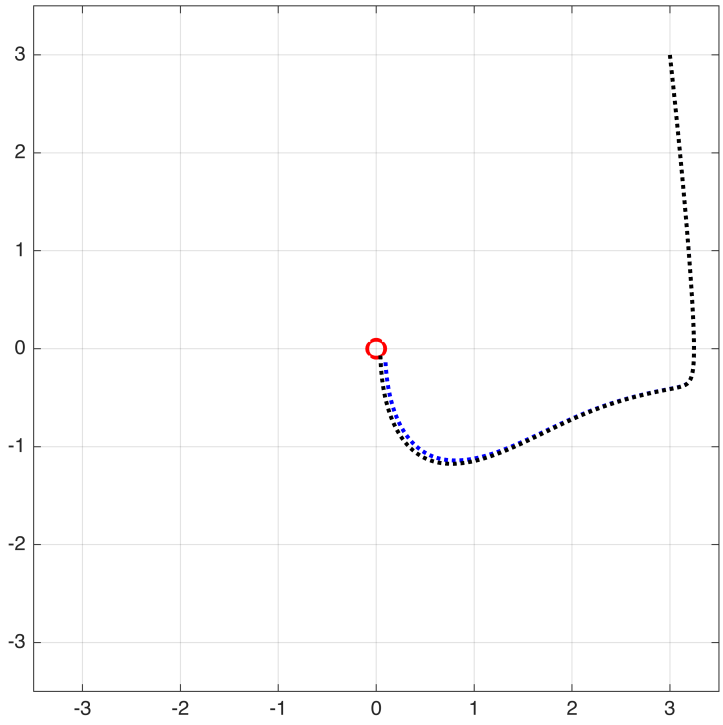
```
    for step = 1:numSteps
        a = JPhi(trajectory1(1, step),trajectory1(2,
step),inputSignal(step));
        if step ==1
            controlMatrix = [controlMatrix, a(1:2,3)];
        else
            controlMatrix = [a(1:2,1:2)*controlMatrix a(1:2,3)];
        end

    end

    ins(:,iteration) = inputSignal;
    inputSignal = inputSignal - (controlMatrix' * controlMatrix + .005 *
eye(numSteps)) \ (controlMatrix' * (currentPosition(1:2) - targetPosition));

    % Update trajectories with corrected signal and plot
    for color = ['g','k']
        currentPosition = [3; 3; 0];
        trajectory = [currentPosition];

        for step = 1:numSteps
            if color == 'g'
                currentPosition = Phi(trajectory(1, step),trajectory(2,
step),inputSignal(step));
            elseif color == 'k'
                currentPosition =
Phi(currentPosition(1),currentPosition(2),inputSignal(step));
            end

            trajectory = [trajectory, currentPosition];
        end

        plot(trajectory(1, :), trajectory(2, :), 'Color', color,
'LineStyle', ':', 'LineWidth', 2);
    end

    axis equal;
    grid on;
    axis([-3.5 3.5 -3.5 3.5]);
end
```

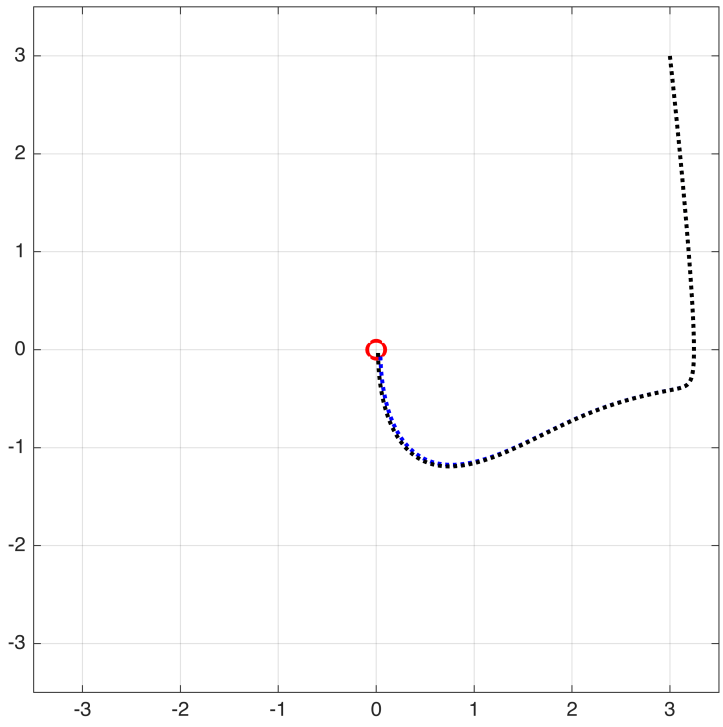

3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

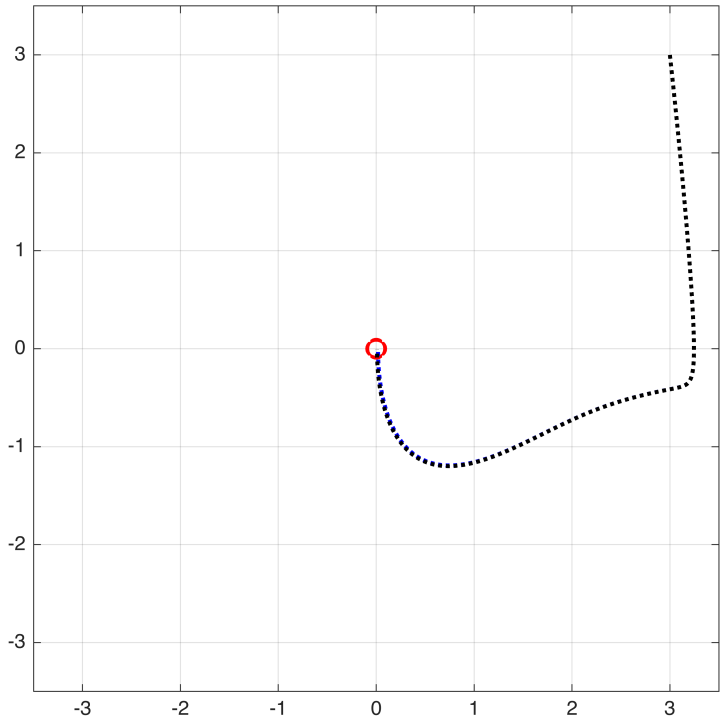

3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

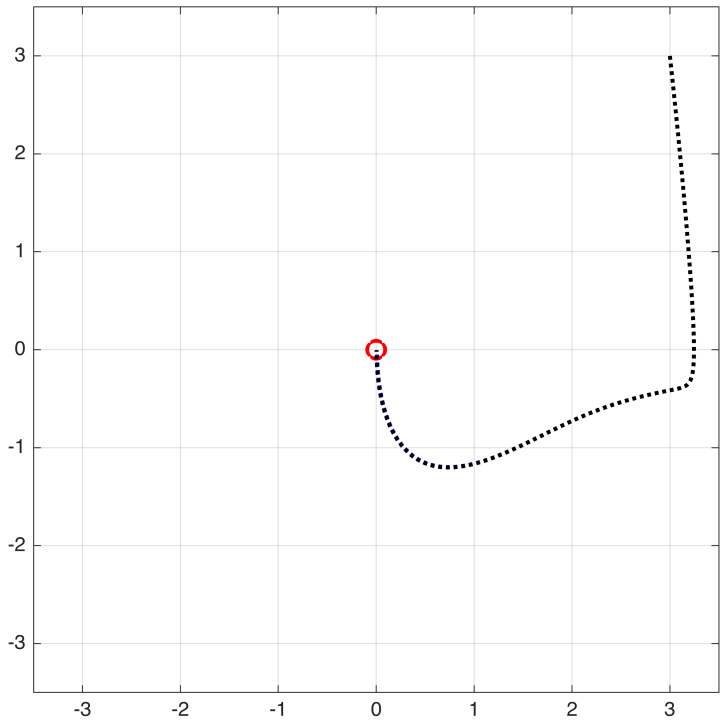

3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

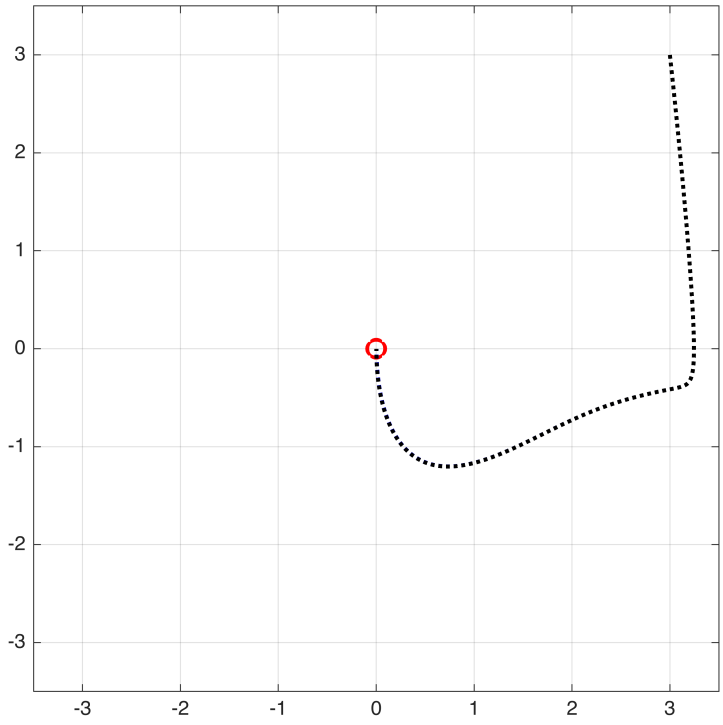

3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

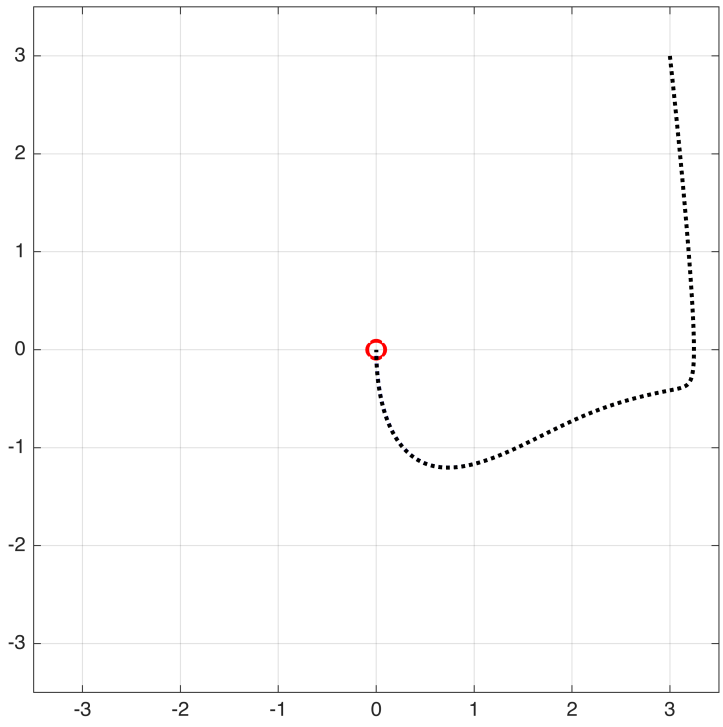

3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3


3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3


3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

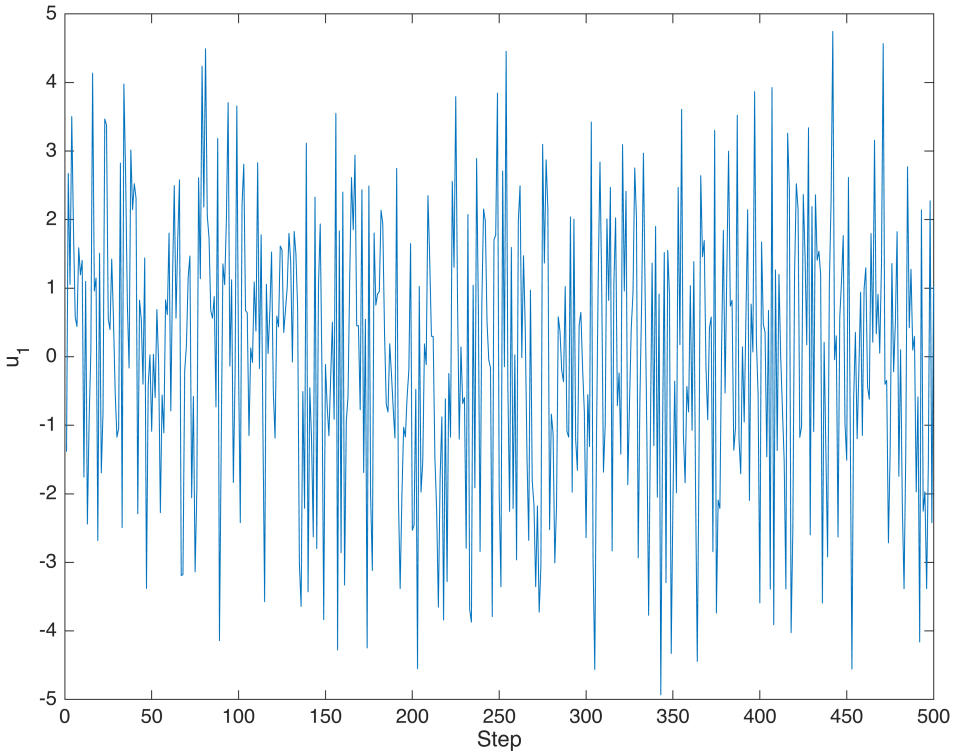

3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

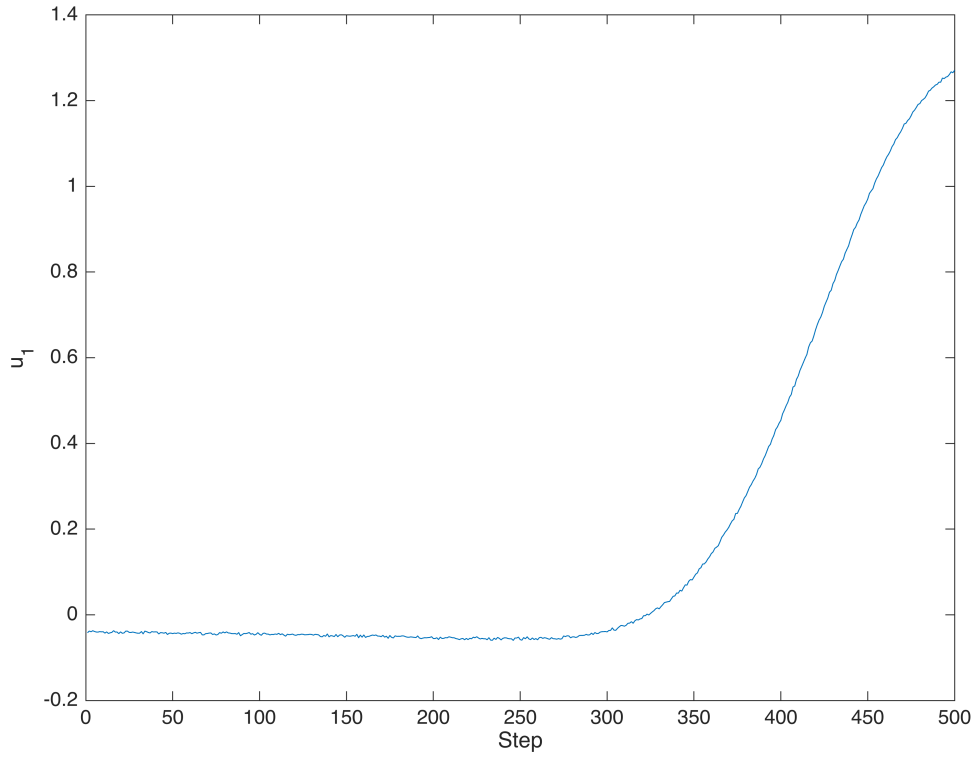


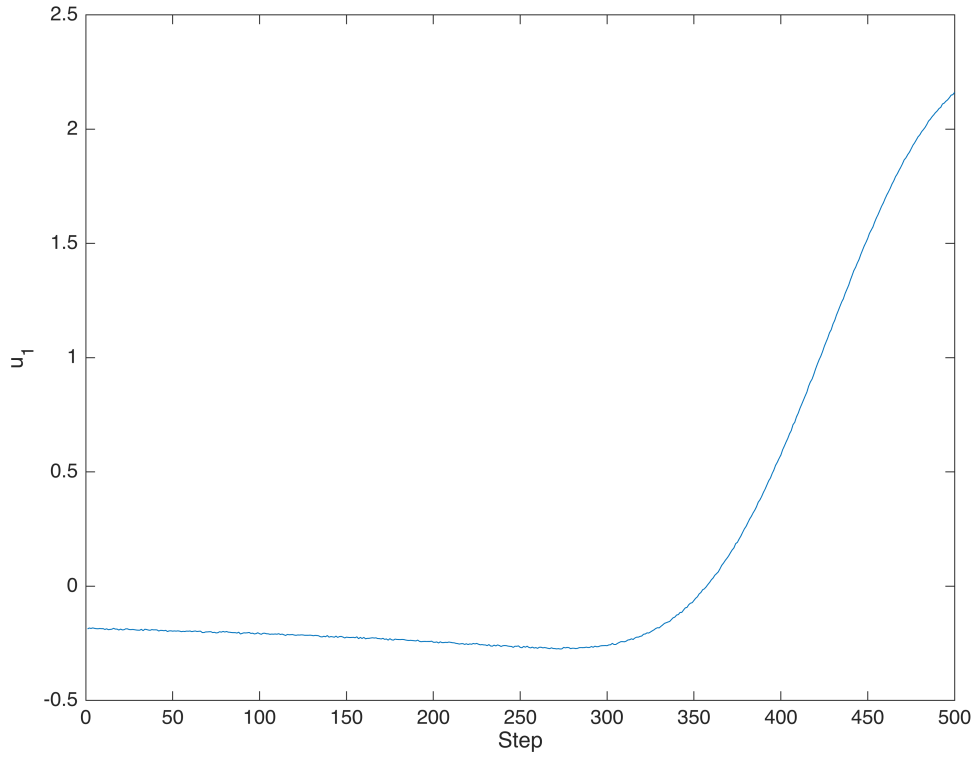


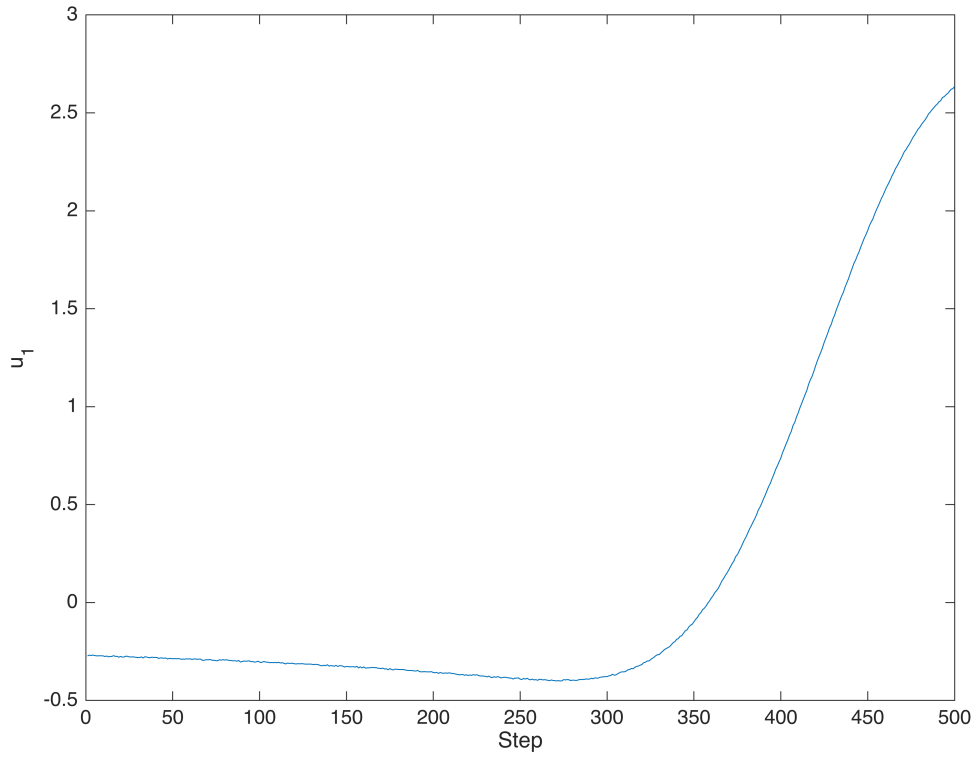

```
step = linspace(1,200,200);
for i = 1:10
```

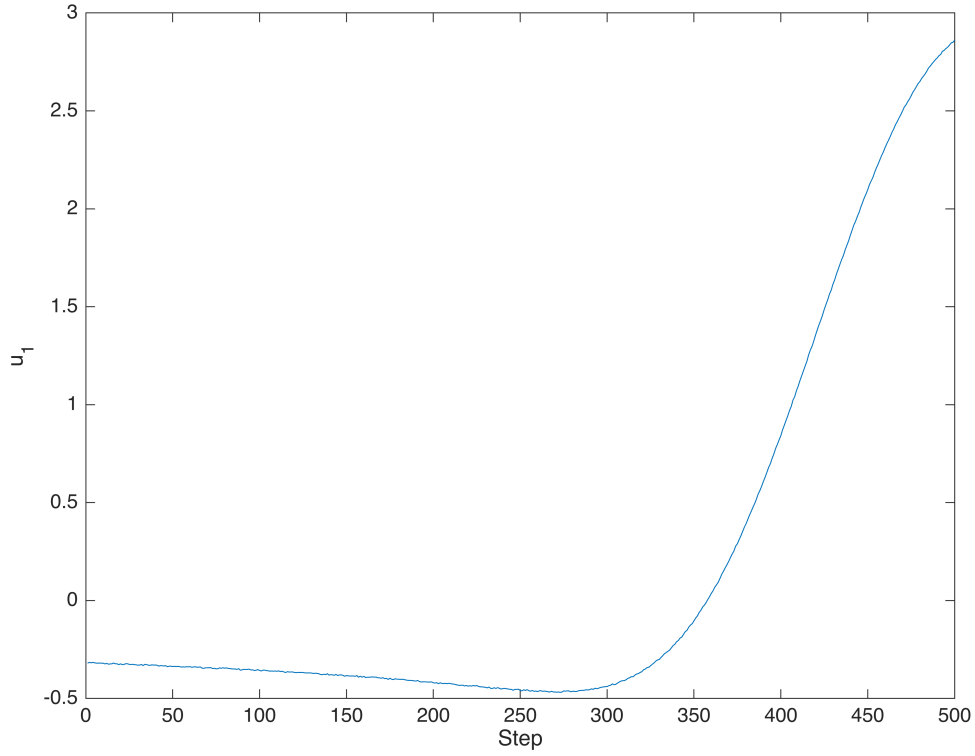
```
    figure;
    title(['Iteration: ',{i}])
    plot(ins(:,i))
    xlabel('Step'), ylabel('u_1')
end
```

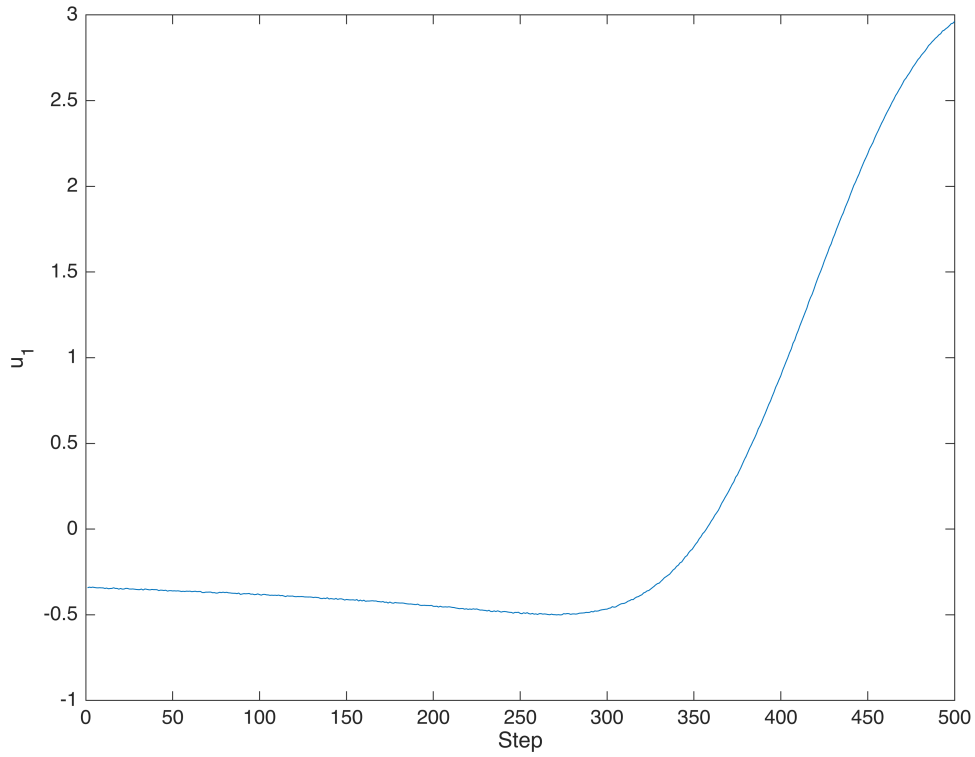


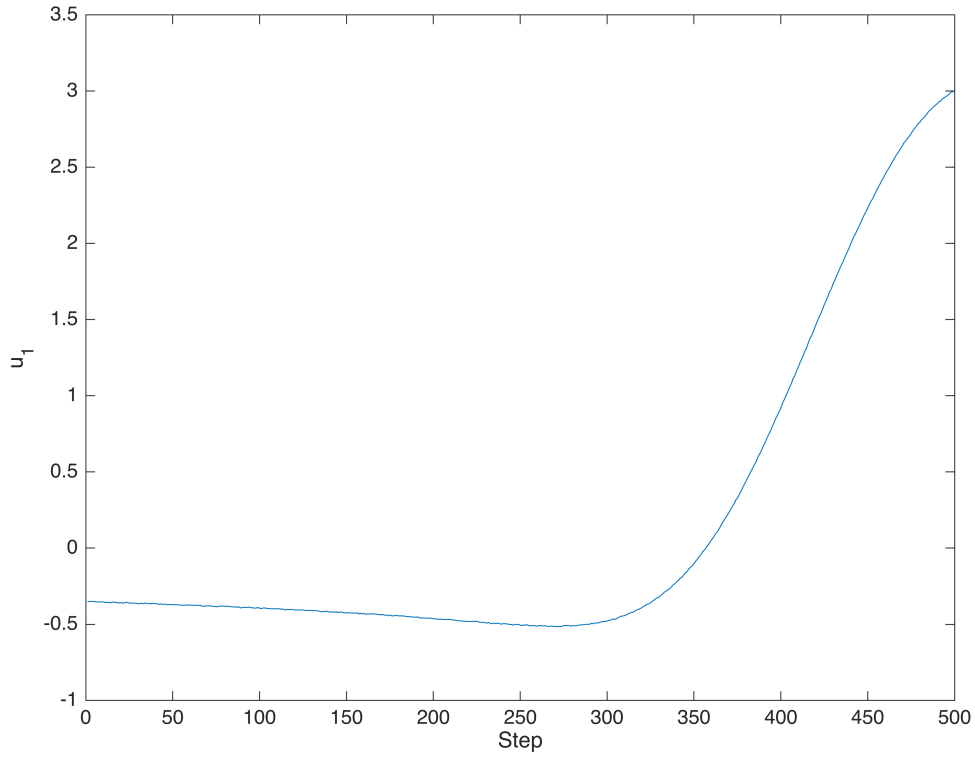

1.4
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
u_1
0
50
100
150
200
250
300
350
400
450
500
Step

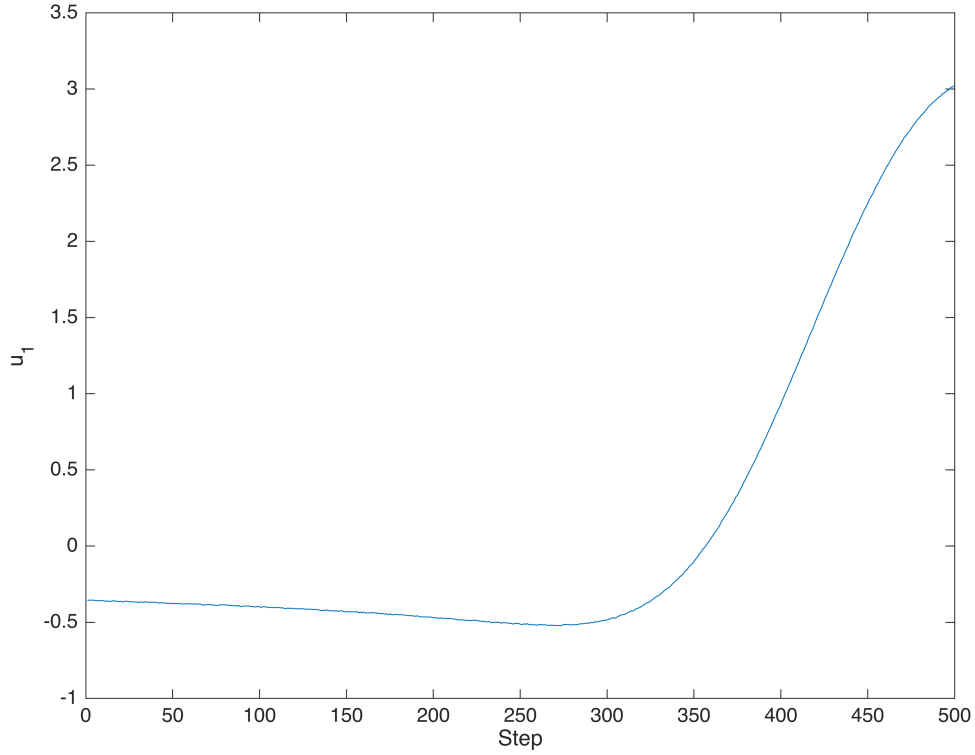

2.5
2
1.5
1
0.5
0
-0.5
u_1
0
50
100
150
200
250
300
350
400
450
500
Step

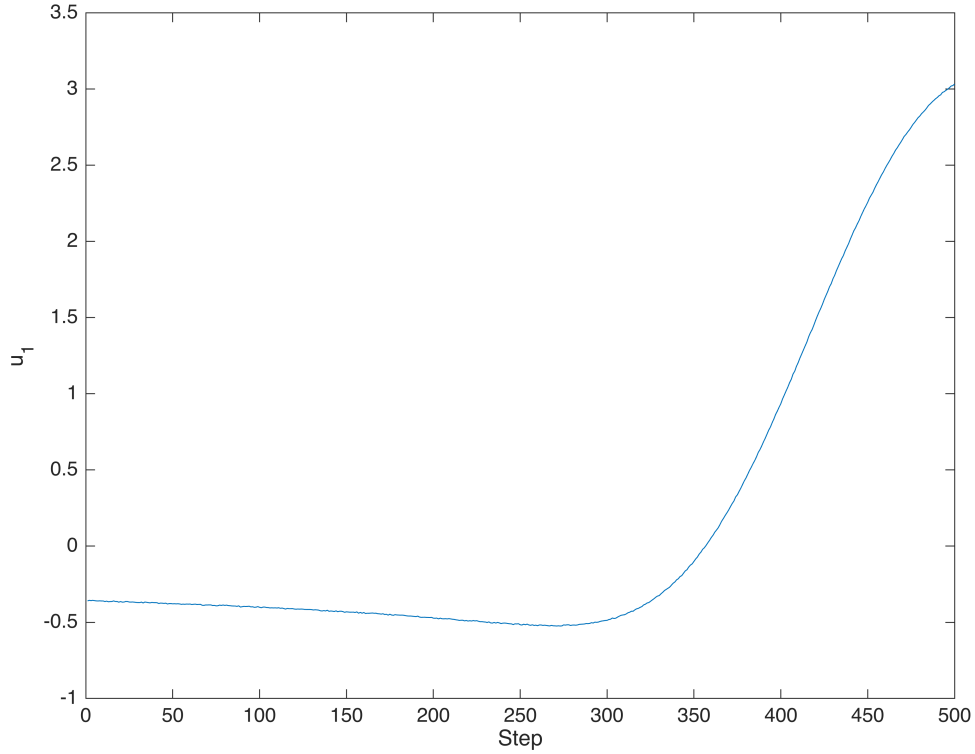

3
2.5
2
1.5
1
0.5
0
-0.5
u_1
0
50
100
150
200
250
300
350
400
450
500
Step

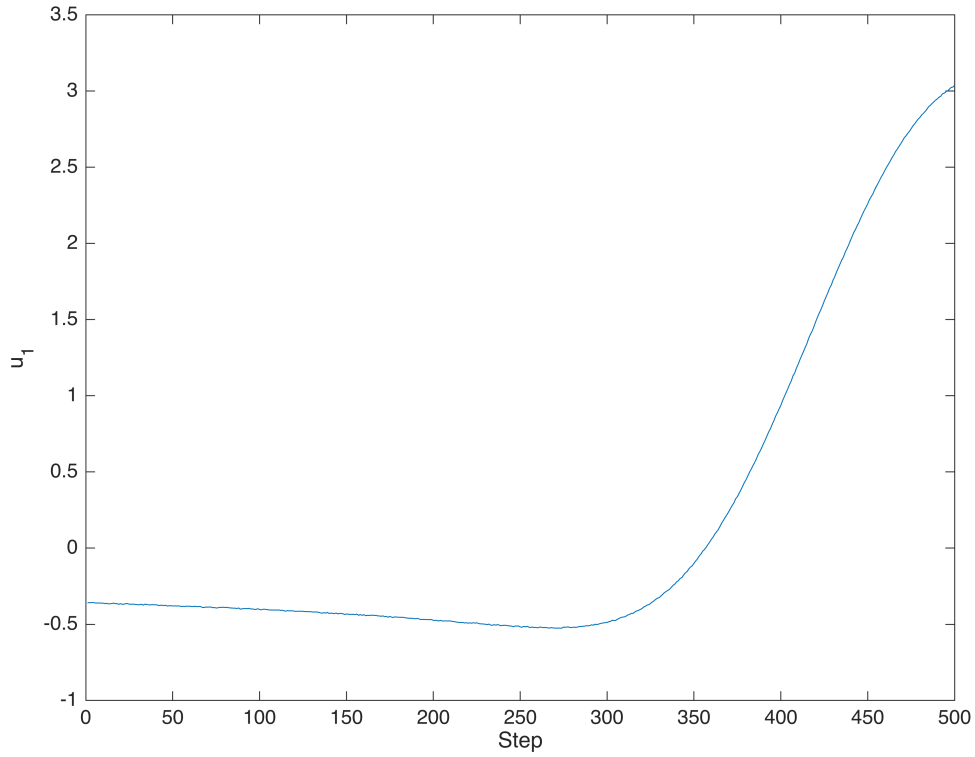

3
2.5
2
1.5
1
0.5
0
-0.5
u_1
0
50
100
150
200
250
300
350
400
450
500
Step


3
2.5
2
1.5
1
0.5
0
-0.5
-1
u_1
0
50
100
150
200
250
300
350
400
450
500
Step


3.5
3
2.5
2
1.5
1
0.5
0
-0.5
-1
u_1
0
50
100
150
200
250
300
350
400
450
500
Step


3.5
3
2.5
2
1.5
1
0.5
0
-0.5
-1
u_1
0
50
100
150
200
250
300
350
400
450
500
Step

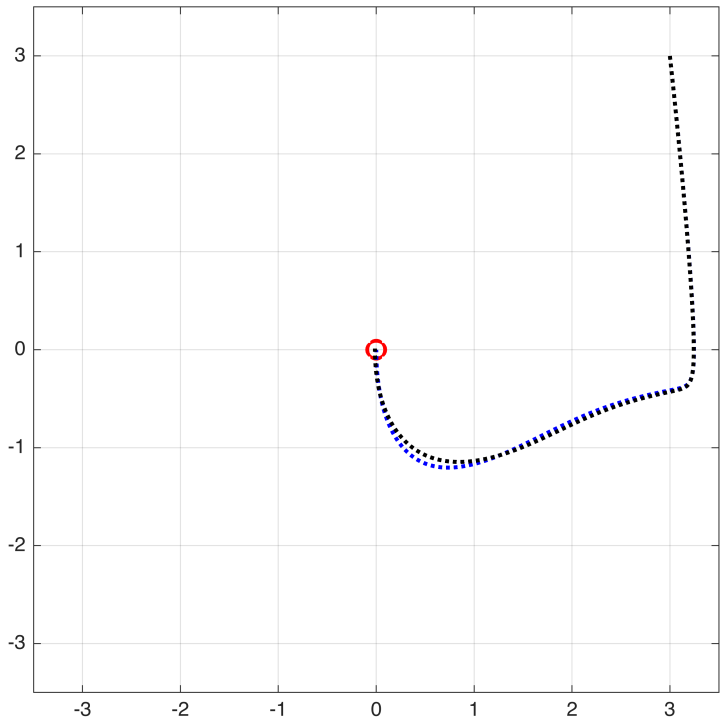

3.5
3
2.5
2
1.5
1
0.5
0
-0.5
-1
u_1
0
50
100
150
200
250
300
350
400
450
500
Step

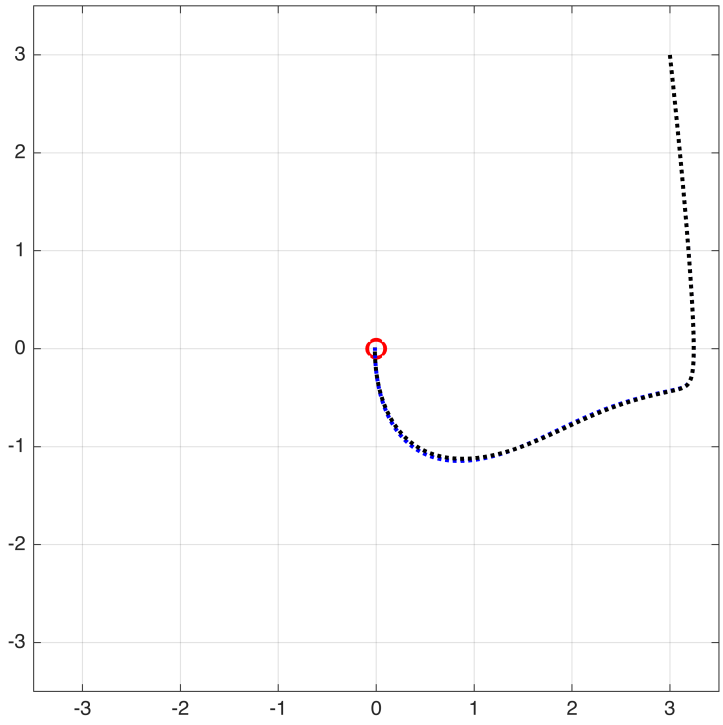


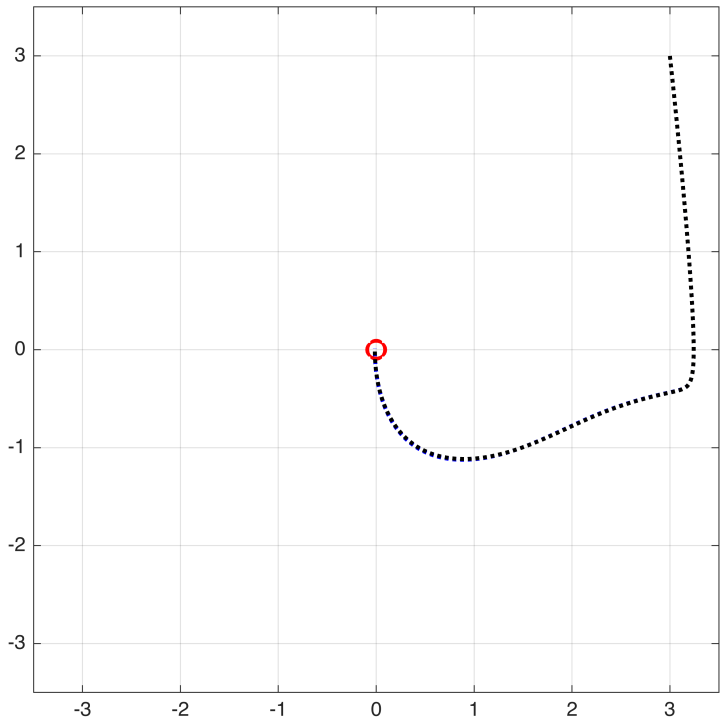

```
options = optimset('Display', 'off');
% Main iterative process
for iteration = 1:10
    % Initialize system position
    currentPosition = [3; 3];

    % Track the trajectory starting from the initial position
    trajectory1 = [currentPosition; inputSignal(1)];

    % Update trajectory at each time step
    for step = 1:numSteps
        currentPosition =
Phi(currentPosition(1),currentPosition(2),inputSignal(step));
        trajectory1 = [trajectory1, currentPosition];
    end

    % Plot the trajectory and target point
    figure;
    plot(targetPosition(1), targetPosition(2), 'ro', 'MarkerSize', 9,
'LineWidth', 2);

    hold on;
    plot(trajectory1(1, :), trajectory1(2, :), 'b:', 'LineWidth', 2);
```

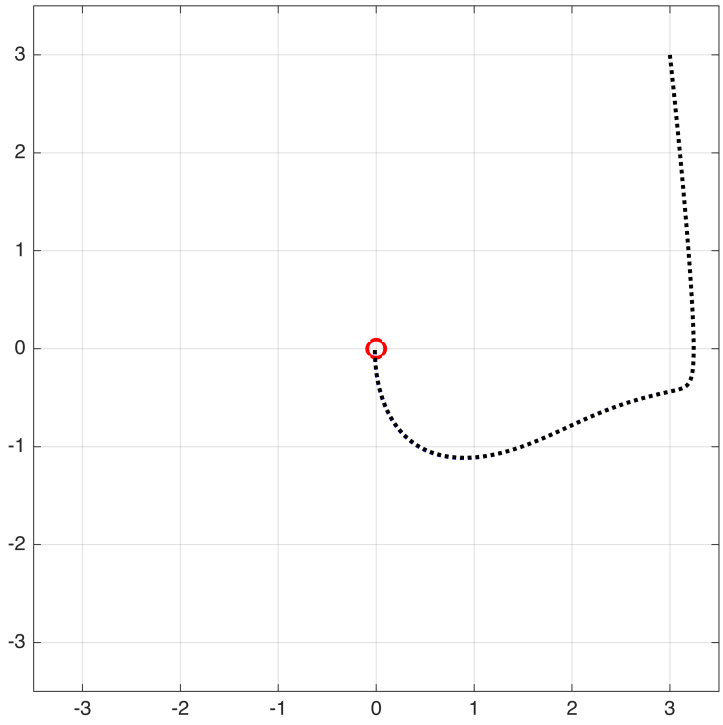
```
    % Compute correction for the input signal
    controlMatrix = [];
    for step = 1:numSteps
        a = JPhi(trajectory1(1, step),trajectory1(2,
step),inputSignal(step));
        if step ==1
            controlMatrix = [controlMatrix, a(1:2,3)];
        else
            controlMatrix = [a(1:2,1:2)*controlMatrix a(1:2,3)];
        end

    end

    % Optimization
    U = inputSignal;
    gamma = 0.005;
    H_constraint = controlMatrix;

    % The number of variables (size of ΔU)
    n = size(U, 1);

    % Define the H matrix for the quadratic term
    H = (1 + gamma) * eye(n);

    % Define the linear term
    f = U;

    % No inequality constraints
    A = [];
    b = [];

    % Equality constraints
    Aeq = H_constraint;
    beq = zeros(size(H_constraint, 1), 1);

    % Lower and upper bounds (assuming none for this problem)
    lb = [];
    ub = [];

    % Solve the quadratic program
    delta_U = quadprog(H, f, A, b, Aeq, beq, lb, ub,(targetPosition -
currentPosition(1:2)), options);

    % Now delta_U contains the optimized ΔU
    ins(:,iteration) = inputSignal;
    inputSignal = inputSignal + delta_U;

    % Update trajectories with corrected signal and plot
    for color = ['g','k']
        currentPosition = [3; 3; 0];
```

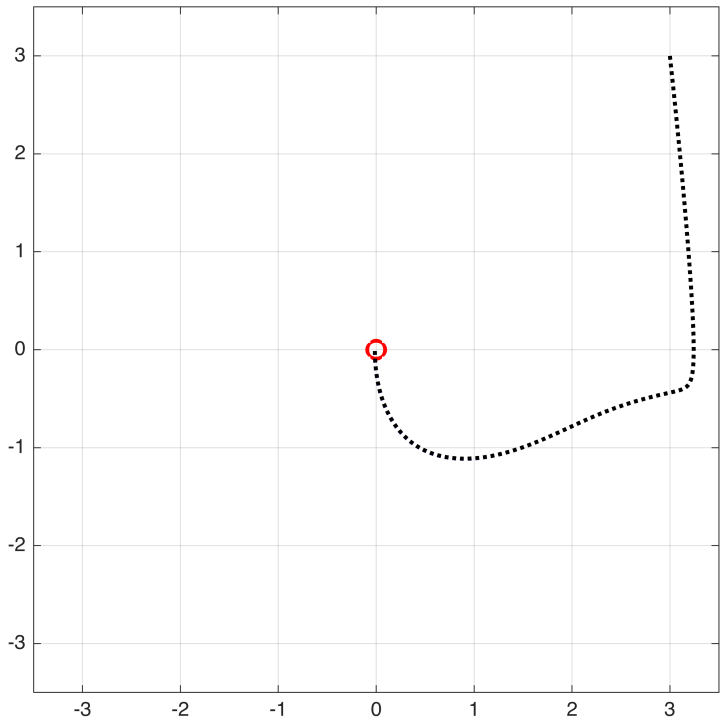
```
        trajectory = [currentPosition];

        for step = 1:numSteps
            if color == 'k'
                currentPosition = Phi(trajectory(1, step),trajectory(2,
step),inputSignal(step));
            elseif color == 'g'
                currentPosition =
Phi(currentPosition(1),currentPosition(2),inputSignal(step));
            end

            trajectory = [trajectory, currentPosition];
        end

        plot(trajectory(1, :), trajectory(2, :), 'Color', color,
'LineStyle', ':', 'LineWidth', 2);
    end

    axis equal;
    grid on;
    axis([-3.5 3.5 -3.5 3.5]);
end
```

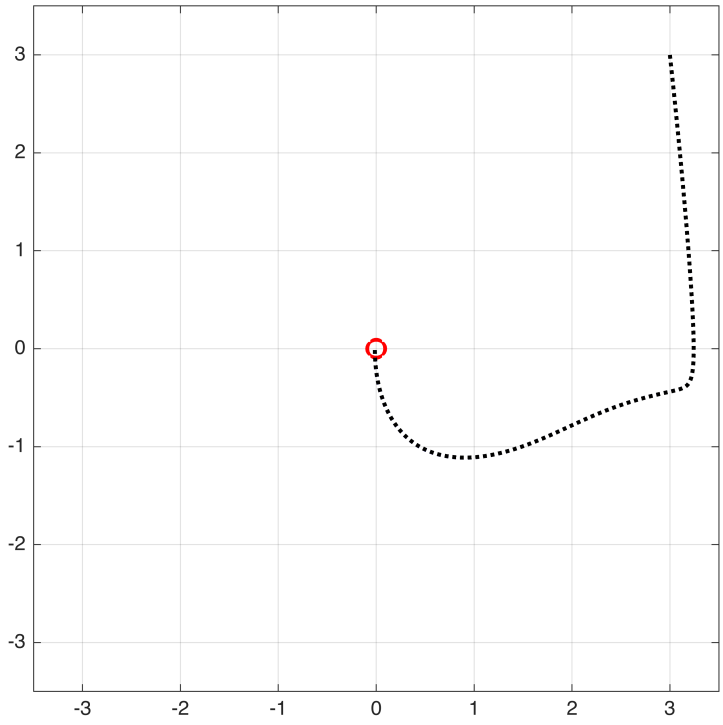


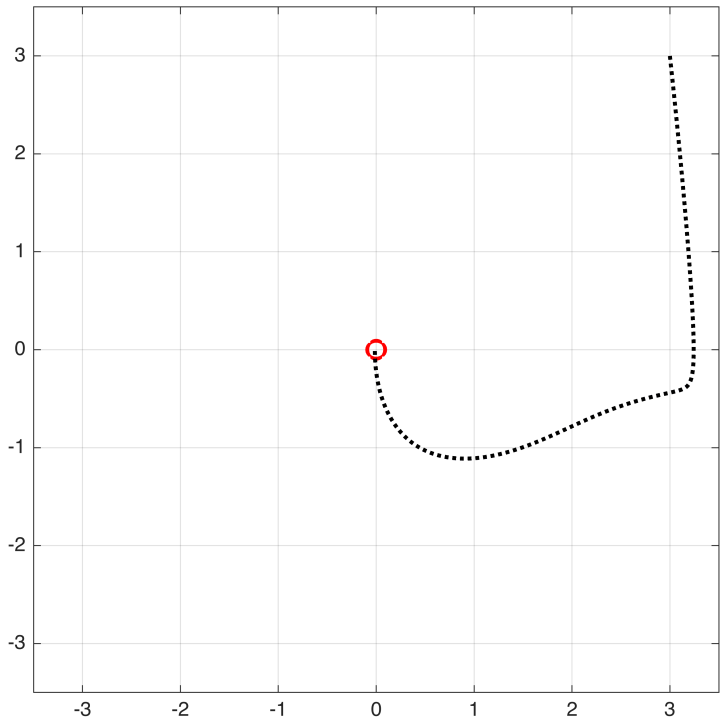

3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

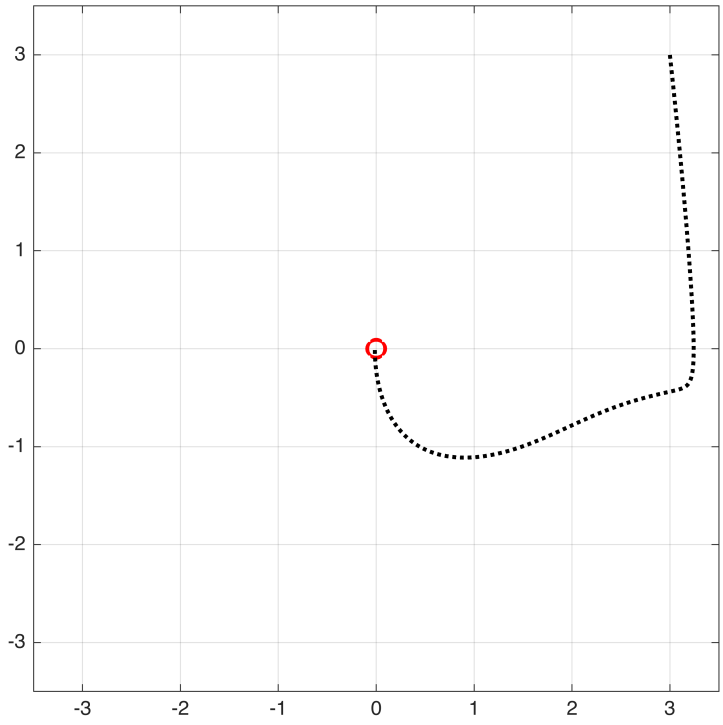

3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

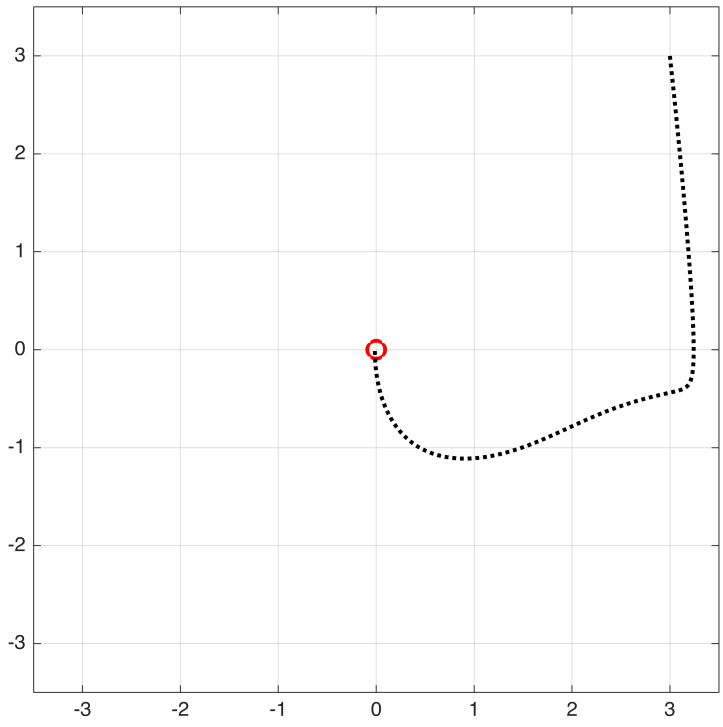

3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

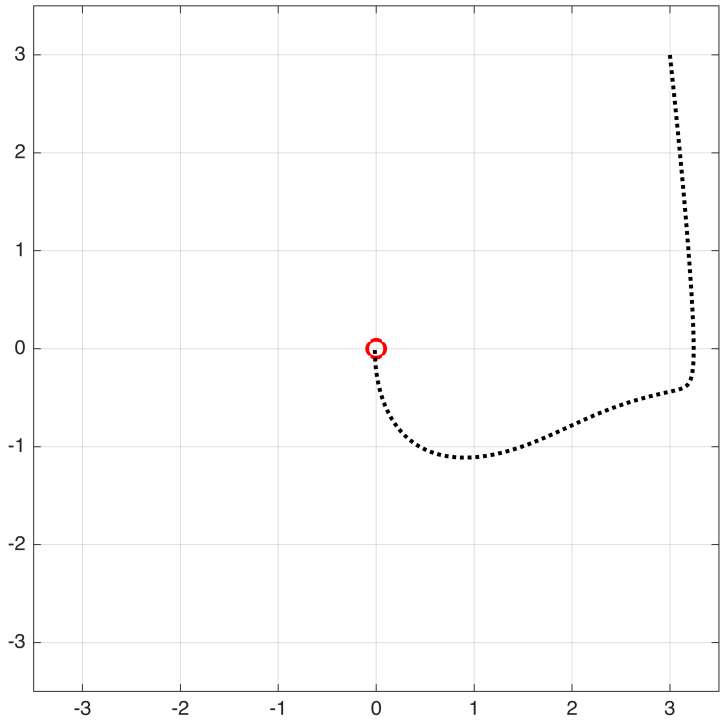

3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3


3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

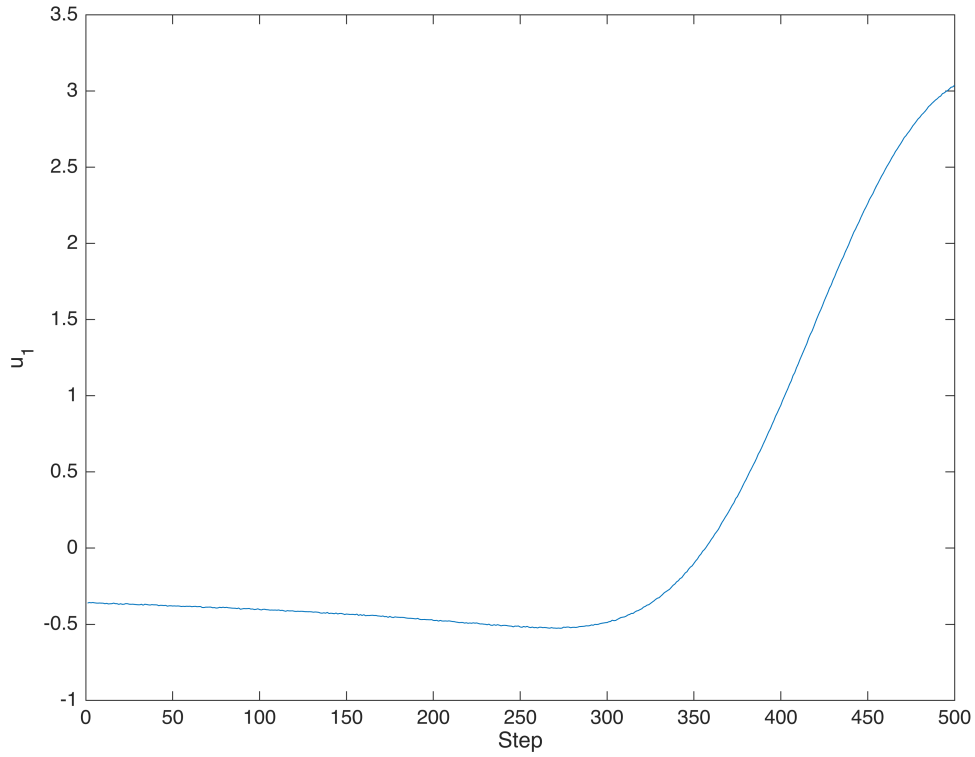

3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

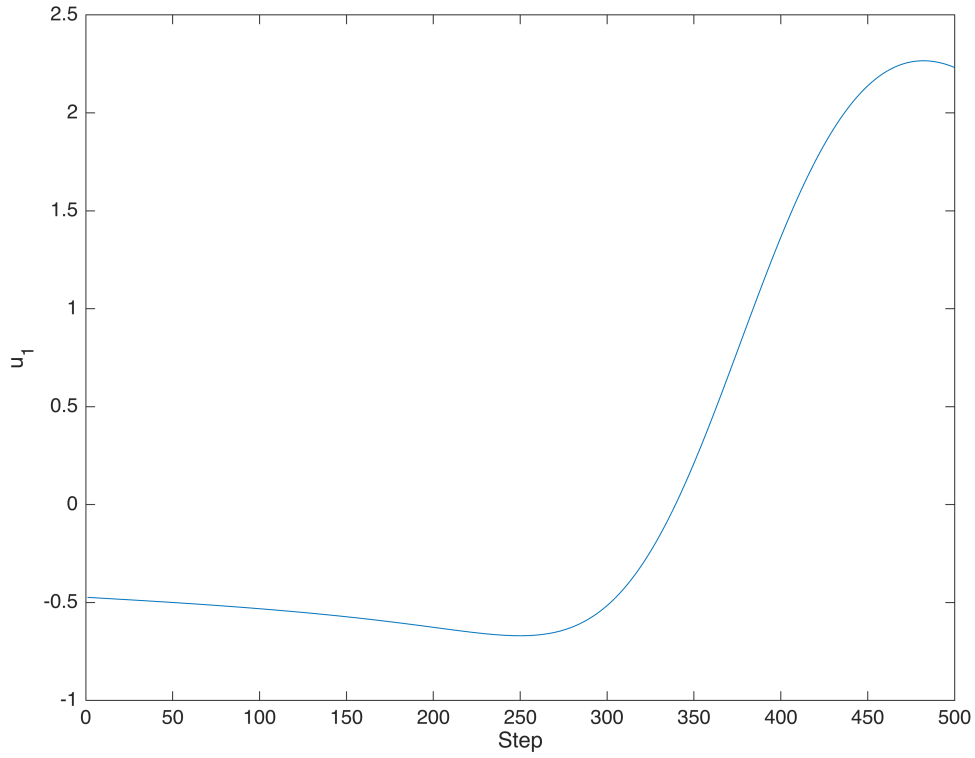

3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

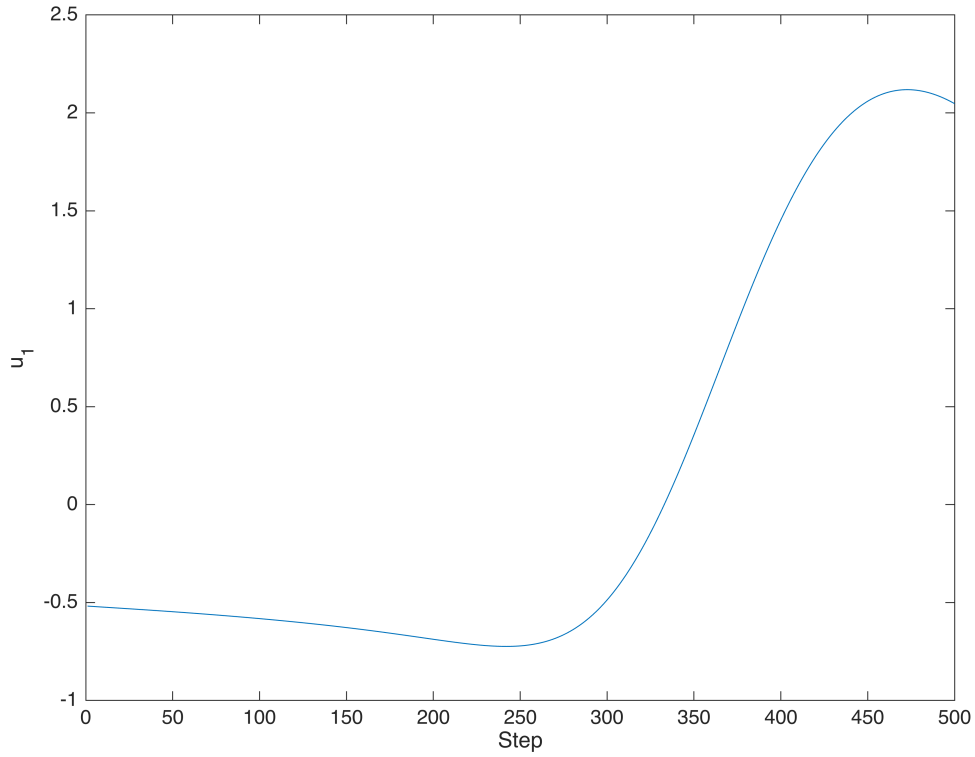

3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

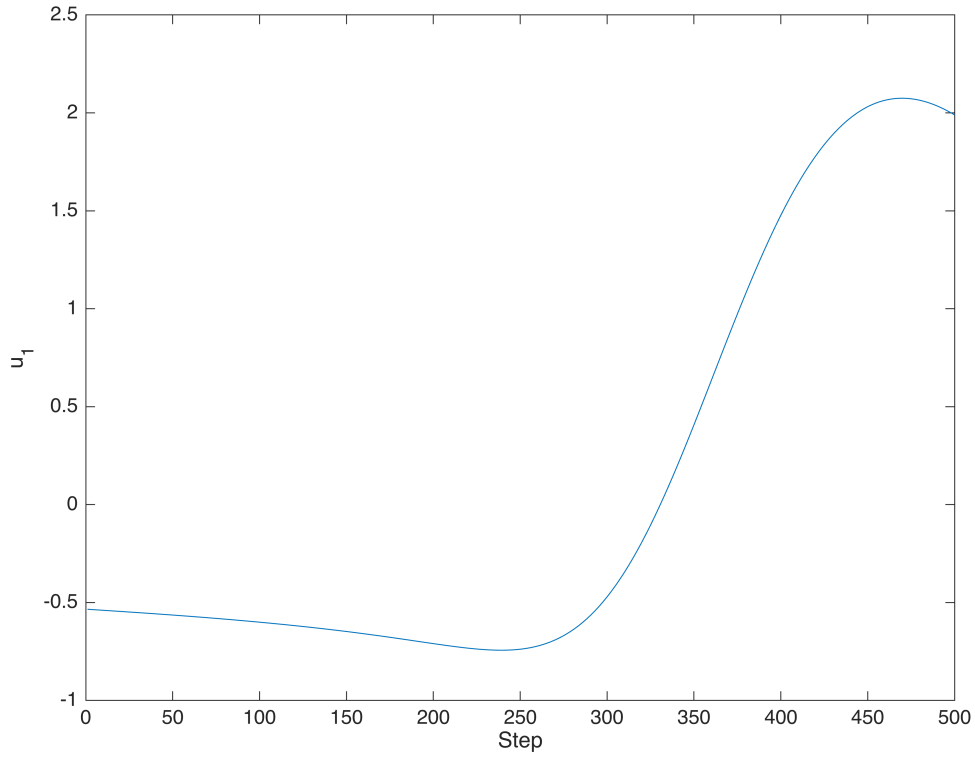


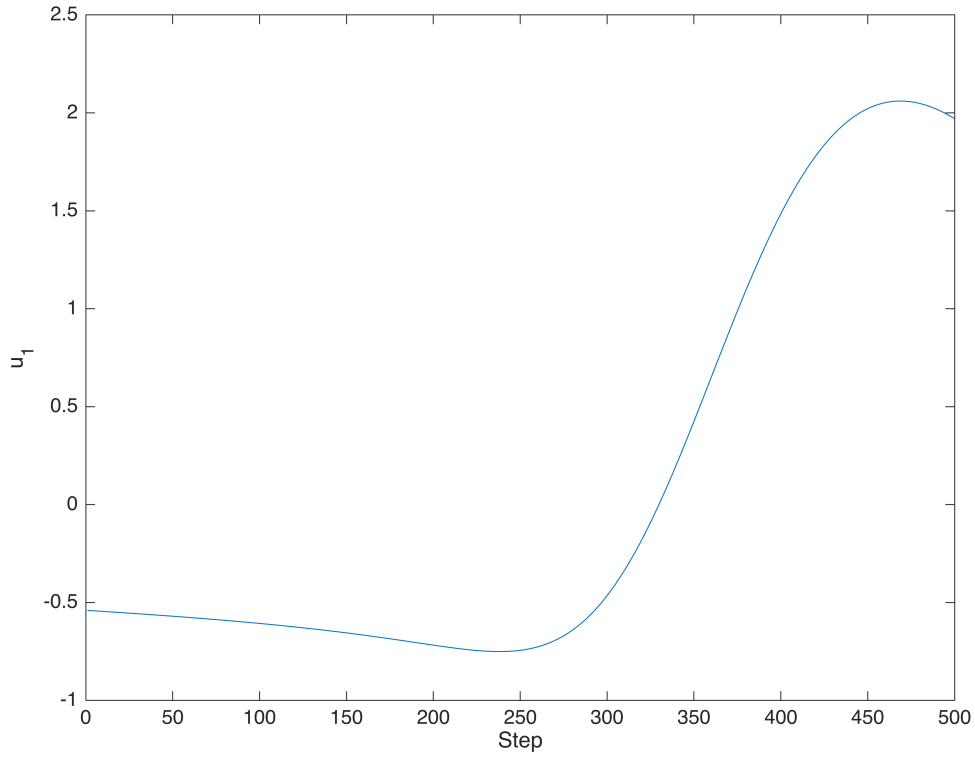

```
step = linspace(1,200,200);
for i = 1:10
    figure;
    title(['Iteration: ',{i}])
    plot(ins(:,i))
    xlabel('Step'), ylabel('u_1')
end
```

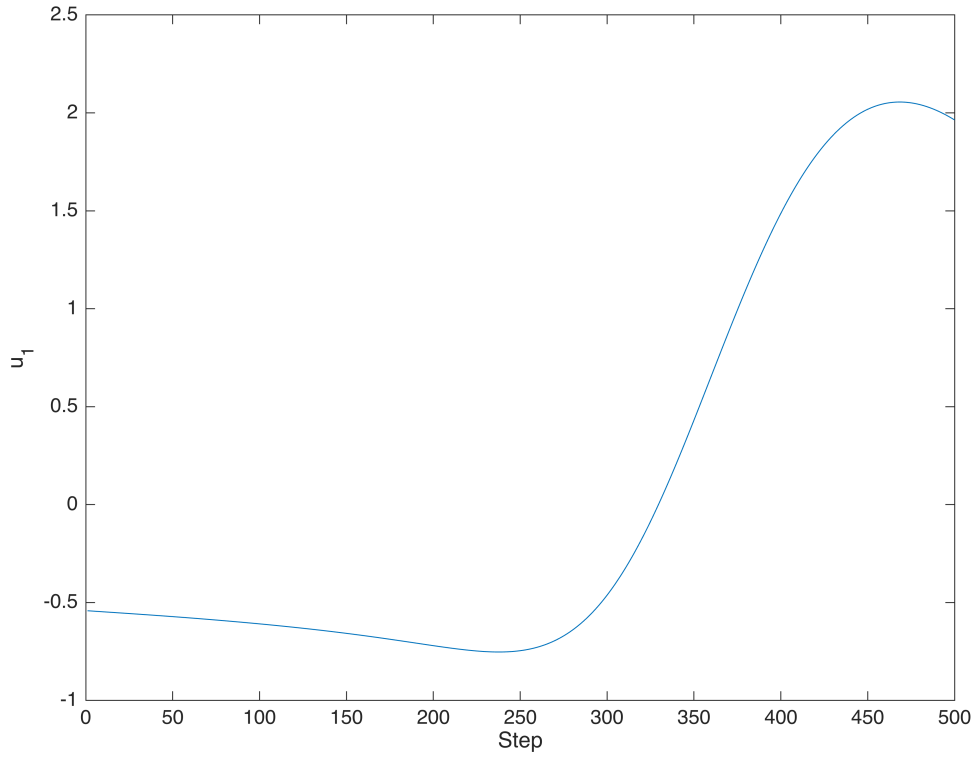

3.5
3
2.5
2
1.5
1
0.5
0
-0.5
-1
u_1
0
50
100
150
200
250
300
350
400
450
500
Step

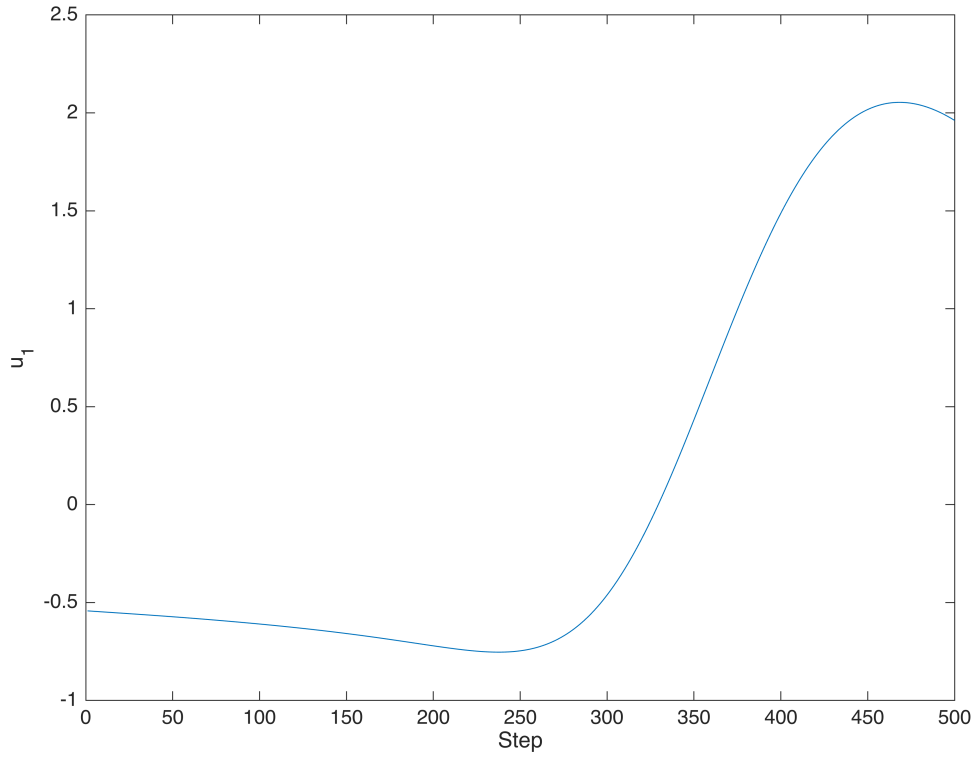

2.5
2
1.5
1
0.5
0
-0.5
-1
u_1
0
50
100
150
200
250
300
350
400
450
500
Step

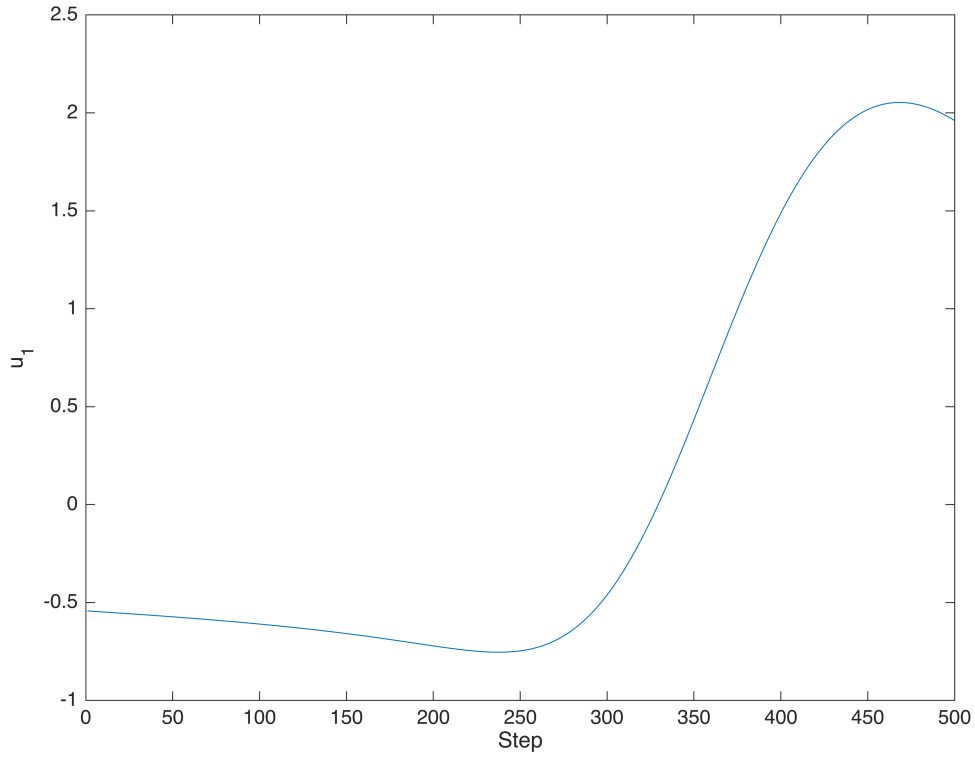

2.5
2
1.5
1
0.5
0
-0.5
-1
u_1
0
50
100
150
200
250
300
350
400
450
500
Step

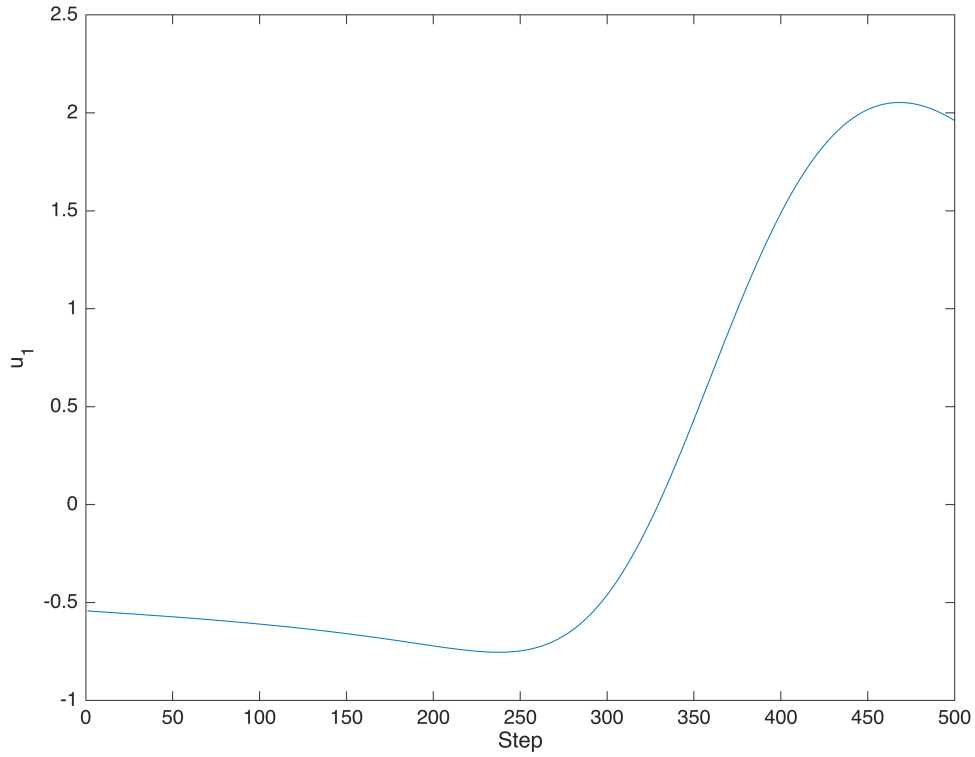

2.5
2
1.5
1
0.5
0
-0.5
-1
u_1
0
50
100
150
200
250
300
350
400
450
500
Step

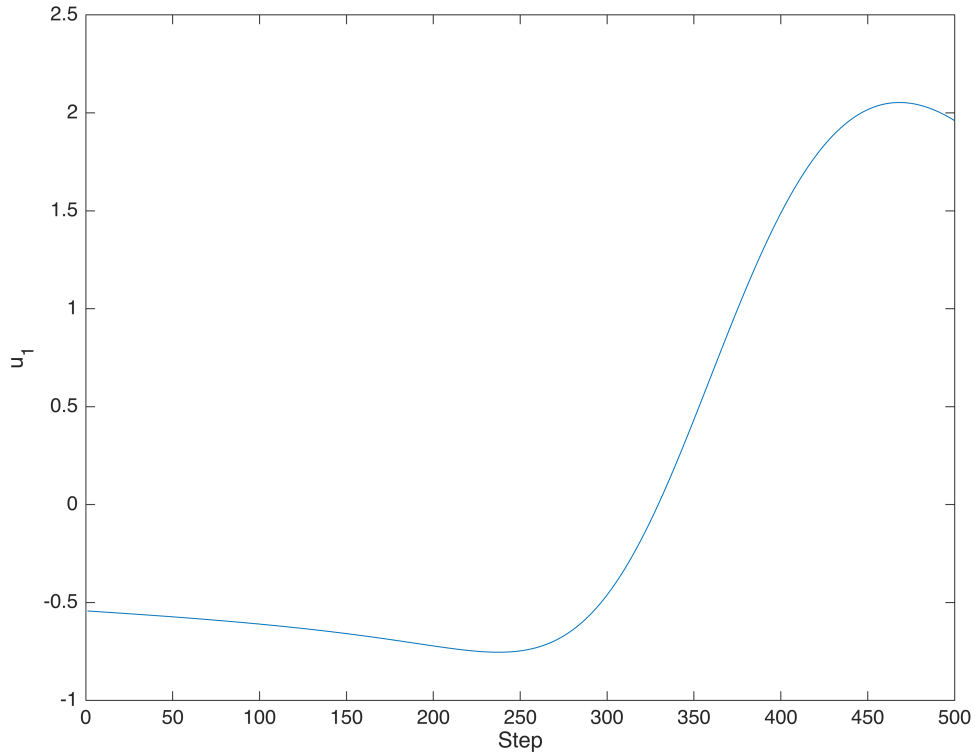

2.5
2
1.5
1
0.5
0
-0.5
-1
u_1
0
50
100
150
200
250
300
350
400
450
500
Step


2.5
2
1.5
1
0.5
0
-0.5
-1
u_1
0
50
100
150
200
250
300
350
400
450
500
Step


2.5
2
1.5
1
0.5
0
-0.5
-1
u_1
0
50
100
150
200
250
300
350
400
450
500
Step

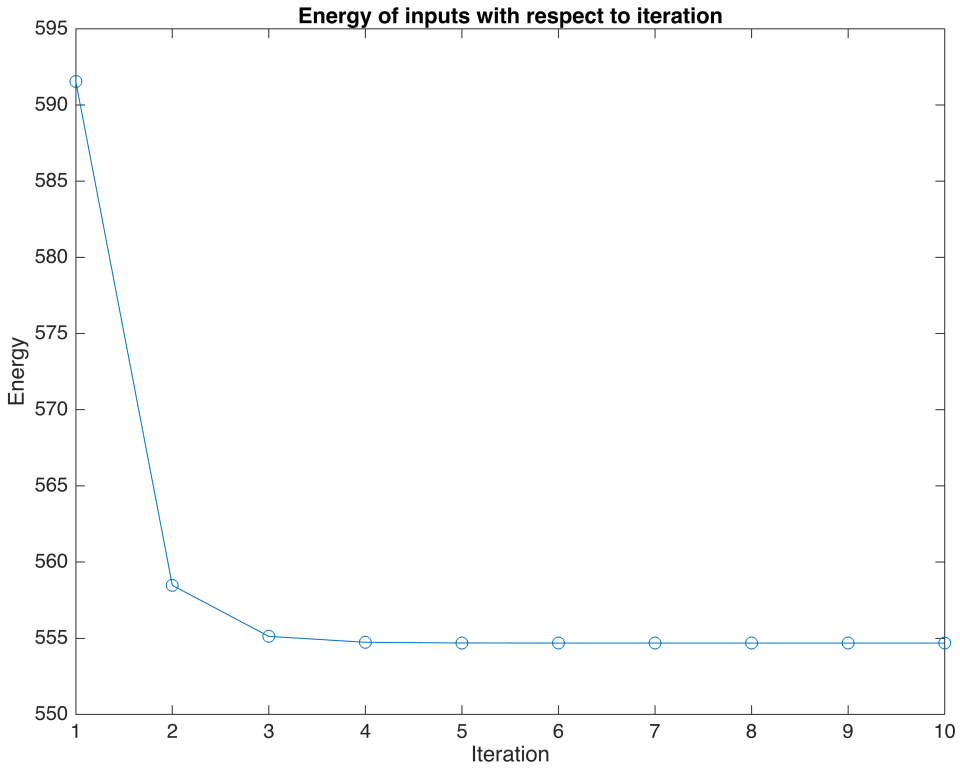

2.5
2
1.5
1
0.5
0
-0.5
-1
u_1
0
50
100
150
200
250
300
350
400
450
500
Step






```
% Preallocate arrays to hold energy values for each iteration
energy_u1 = zeros(1, 10);
```

```
% Calculate energy for each iteration
for j = 1:10
    energy_u1(j) = sum(ins(:,j).^2);
end

% Now plot the energy for each input with respect to iteration
figure;
plot(1:10, energy_u1, '-o', 'DisplayName', 'Energy of u_1');
xlabel('Iteration');
ylabel('Energy');
title('Energy of inputs with respect to iteration');
```




```
function [Phi,Psi_p,JPhi] = compute_Phi_and_JPhi(p,F,xi,dt)
    Id = [];
    for i = 1:length(xi)
        Id = [Id; xi(i)];
    end
    Psi = Id;
     J = eye(length(xi));

     Phi_syms = Psi;
     JPhi_syms = J;
```

```
    for k = 1:p
        Psi = simplify(J*F);
        J = simplify(jacobian(Psi,xi));

        Phi_syms = Phi_syms + (dt^k/factorial(k))*Psi;
        JPhi_syms = JPhi_syms + (dt^k/factorial(k))*J;
    end

    Phi = matlabFunction(simplify(Phi_syms));
    Psi_p = matlabFunction(simplify(Psi));
    JPhi = matlabFunction(simplify(JPhi_syms));

end
```