Stochastic Optimal Control

K.M.J. Jacobs (s4134621) Zhuoran Liu (s4594851)
Ankur Ankan (s4753828)

January 30, 2017

Contents

1 Introduction **3**

2 Problem statement **3**
2.1 Controlled random walk 3
2.2 Mountain car problem 3

3 Results **4**
3.1 Controlled random walk 4
3.1.1 How does the optimal control depend on parameters ν and T numerically? 4
3.1.2 How does the delayed choice mechanism work? 4
3.2 Mountain car problem 5
3.2.1 What is the optimal control policy for this problem? . . . 6

4 Discussions and conclusions **7**

5 Appendix **9**
5.1 Code for the controlled random walk (MATLAB) 9
5.2 Code for the mountain car problem (iPython Notebook) 16
5.2.1 Imports 16
5.3 Definitions 16
5.4 Base class for simulations 16
5.5 Optimal Control 18
5.6 Approximating $J(x, v, t = 0)$ 18

Document requirements shown by flag `requirementstrue`. Set flag `requirementsfalse`.

1 Guidelines for ML assignments

Note: All reports should be handed in before the end of January 2017. Hard deadline is **January 31 at 23.59**. Any questions about the below points please contact me (Hans-Christian Ruiz). For the assignments please refer to the ML course page.

1.1 Requirements to hand in the reports

- Hand in each report as a SINGLE document with all figures, code, etc (other "extra" files will not be considered). Hand in printed in my post box (Ruiz, wing 8, ground floor).
- In addition, hand in the code that can be run stand-alone. Send us an email with a zip-file containing the codes for all assignments. The zip-file should be named with the last names of all group members. Each code in the zip-file should be named as AssignmentName where AssignmentName is: Ising, BM, MNIST, Lasso, Control
- Your name on each report and the name of the corresponding code file in the zip-file that you sent us via email.
- The report should be well structured and clear.
- Figures need to have a caption explaining in detail what they show. 6. No more than 8 pages (excluding references and code) with 12 pts font.

1.2 Minimal requirements of the content

The report for each exercise addresses following points:

- Introduction: summarizes the underlying theory and algorithm(s). It should be a short description of the theoretical background and explanation of the method considered (no more than a page!).
- Problem statement: Itemizes a number of research questions that you address
- Results: Detailed description of the different numerical studies that you have performed with plots and specification of the parameter settings (a) Precise, clear description of what you have done, with the used formulas and the argumentation of why you have done it; e.g. what measure of convergence you used, are there alternatives? (b) Figures with a detailed explanation of what the relevant results are (c) An analysis of the results (for example: is the result as expected? why?/why not? what does the result mean? etc...)
- General discussion/Conclusions: A summary of the main findings and conclusions (no more than half page!)

- Appendix: If the exercise requires to write a code, add the code in the appendix. The code should have the following characteristics (a) Well documented, comments! (b) Suitable structure for readability, e.g. indentation, proper (mnemonic) variable naming. 6. Note: The code will not be evaluated based on whether it is optimized or not, only if it is clearly readable. But, IT MUST RUN STAND- ALONE!

2 Introduction

The main topic of this assignment is Stochastic Optimal Control. For many problems, it is analytically feasible to compute the optimal control for a given stochastic system. For some problems however, it is only possible to approximate optimal control. The first part of the assignment focusses on a controlled random walk which is one of the easiest problems to look at. The second part of the assignment focusses on the well-known mountain car problem. In the second part of the assignment, MCMC sampling will be applied to approximately compute the optimal control.

3 Problem statement

3.1 Controlled random walk

In the controlled random walk problem, the rate of change of the state is defined as follows:

$$dx = udt + d\xi \tag{1}$$

The state has initial position x at $t = 0$. The noise variance is $\langle d\xi^2 \rangle = \nu dt$.

There are two targets at $t = T$ at locations $x = \pm 1$. The end-cost function ϕ is defined such that it has delta peaks at the targets:

$$\phi(x(T)) = \begin{cases} 0 & x(T) = \pm 1 \\ \infty & \text{otherwise} \end{cases} \tag{2}$$

The optimal control for this problem satisfies:

$$u^*(t, x) = \frac{\tanh(x/(\nu(T-t)) - x}{T - t} \tag{3}$$

The following research questions are discussed for the controlled random walk problem in this report:

- How does the optimal control depend on parameters ν and T numerically?
- How does the delayed choice mechanism work?

3.2 Mountain car problem

The following research questions are discussed in this report for the mountain car problem as described in the assignment:

- What is the optimal control policy for this problem?

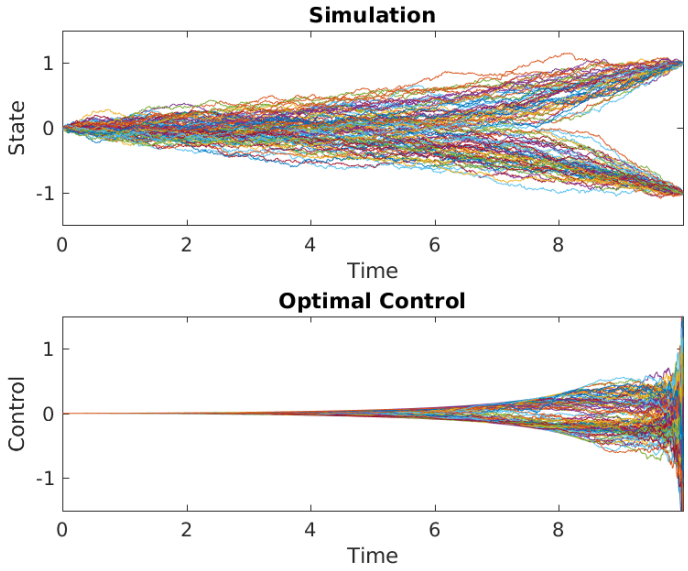

Figure 1: Optimal control for the Controlled Random Walk problem. With $\nu = 0.1$, $T = 10$, $dt = 0.01$ for which 100 trials are shown.

4 Results

4.1 Controlled random walk

4.1.1 How does the optimal control depend on parameters ν and T numerically?

If $\nu \to \infty$, then $\tanh(\frac{x}{\nu(T-t)}) \to 0$ and then $u^*(t,x) \to 0$. If there is a large amount of noise, the the control goes to 0 and if $t \to T$, then there will be a large amount of control.

4.1.2 How does the delayed choice mechanism work?

There is no control unless it is clear that control can steer the system towards the optimal solution. It makes no sense to steer towards an optimum in a state in which the noise can push the state away from the optimal solution. Therefore, it is better to wait until it is ensured that control can steer the system to the optimal solution. This mechanism is shown in figure 1.

If one would choose a lower horizon ($T = 1$ as in figure 2 instead of $T = 10$ as in figure 1), the system has less time and thus less cumulative noise is expected. Therefore, it is advantageous to control the system at the start. Once the system has reached the optimal state ($x = -1$ or $x = 1$), it will steer the system such that it remains in the optimal state.

So when there is more time, there will be more cumulative noise and the system will delay its choice. If there is less time, it will be advantageous to steer the system towards to optimal state at the start.

If, however, there is more noise, the system will try to keep the system stable

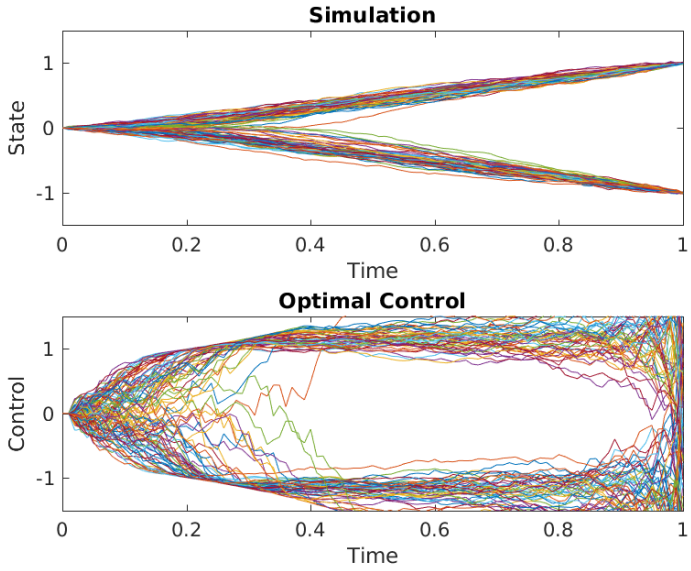

Figure 2: Optimal control for the Controlled Random Walk problem. With $\nu = 0.1$, $T = 1$, $dt = 0.01$ for which 100 trials are shown.

from the beginning on by steering the system such that the optimum remains in reach. So, up to some level, more noise results into larger control. This can be seen by comparing figure 2 to figure 3.

4.2 Mountain car problem

For the first part of the mountain car problem assignment, it is asked to find parameters with $(x_0, v_0) = (0.5, 0)$ such that the problem is too easy and all trajectories reach the top of the hill and it is asked to find parameters such the problem is harder, but still some trajectories will reach the top of the hill.

First, it is easy to gain some insights by plotting the graph for the hill $L(x)$ and by visualizing the gravitational force $F_g(x)$.

What can be seen from the formula of $F_g(x)$, is that the derivative of $L(x)$ forces the car to either go back to the bottom of the valley (this happens when $g < 0$ or it pushes the car further up the hill (when $g > 0$). With $g = 0$, there is no gravitational force at all.

In figures 6 and 7, it can be seen that $g \neq 0$ forces the car back to the bottom of the valley. $g = 0$ has no influence on the trajectories and does not necessarily keep the trajectories in the valley. For large values of ν, it is more often the case that trajectories escape the valley. If the noise is extremely large, then the car escapes the valley fairly quickly. By the large noise, it is unlikely that the car is accidentally pushed back into the valley.

For the second part of the second part of the assignment, it is asked to compute the optimal cost-to-go $J(x, v, t = 0)$ for $x = -2 : 0.1 : 2$ and $v = -2 : 0.1 : 2$ using MCMC. In order to do so, for each (x_0, v_0) pair, n simulations were computed and the optimal cost-to-go is simply calculated from this. The results

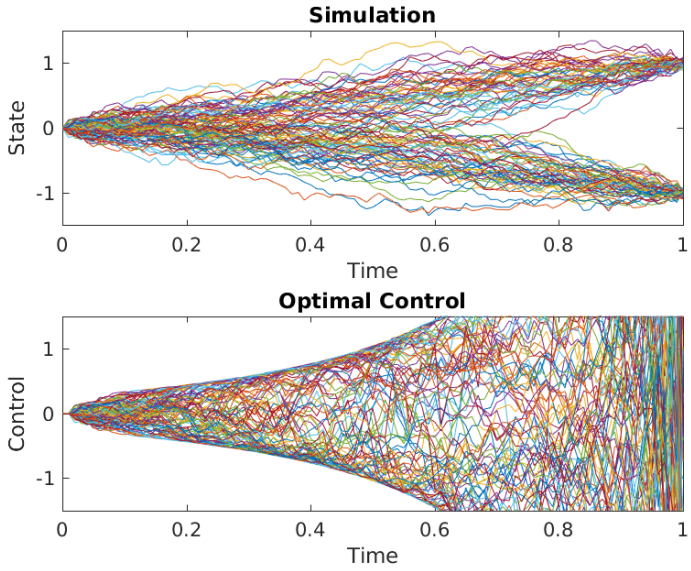

Figure 3: Optimal control for the Controlled Random Walk problem. With $\nu = 0.5$, $T = 1$, $dt = 0.01$ for which 100 trials are shown.

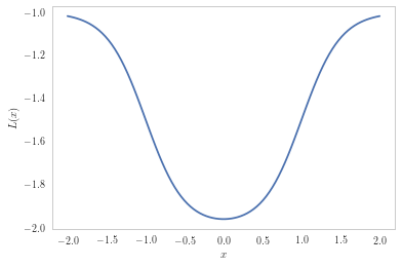

Figure 4: The plot of the hill $L(x)$.

are shown in figure 8. $J = 0$ is the result when all trajectories stay inside the valley (so $x_T = 0$ for all simulations). $J = -1$ is the result when all trajectories are outside the valley (so $x_T = A$ for all simulations). From the figure, it can be seen that most of the trajectories stay inside the valley if $x_0 \approx v_0 \approx 0$. The initial velocity has more influence on the final state than the initial position. $v_0 = 0$ has almost no escaping trajectories whereas it is possible to find escaping trajectories for $x_0 = 0$ (by setting $|v_0| \approx 2$).

4.2.1 What is the optimal control policy for this problem?

The optimal control can be approximated by using MCMC sampling. The following formula expresses how the optimal control can be computed:

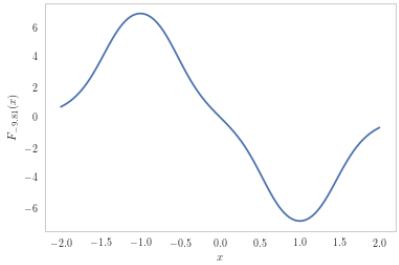

Figure 5: The gravitational force $F_g(x)$ for $g = -9.81$.

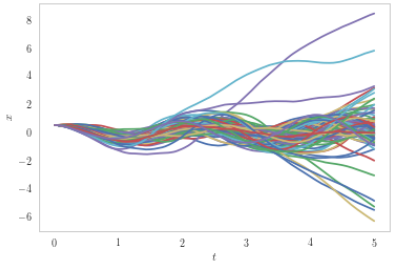

Figure 6: 100 simulations for $x_0 = 0.5$, $v_0 = 0$, $\nu = 1$, $T = 5$ and $g = -10$.

$$S = \phi(x_T^\mu) \tag{4}$$

$$udt = \frac{\sum_\mu d\xi^{(\mu)} \exp(\frac{-S^{(\mu)}}{\lambda})}{\sum_\mu \exp(\frac{-S^{(\mu)}}{\lambda})} \tag{5}$$

It is computed by generating N trajectories $x_{t:T}^\mu$ starting at (x, v, t) and by computing the costs for the final state and by using the initial noise $d\xi^{(\mu)}$.

In figure 9 and 10, there clearly is a difference in the end states for simulations without any control (so $u = 0$) and in simulations with optimal control. Optimal control resulted in only a few simulations in which $-2 < x_T < 2$. Most of the simulations escaped the valley. It is also interesting to see that the control is kept fairly small. The heaviest controls are performed after some time. Here, the delayed choice mechanism also applies because of that.

5 Discussions and conclusions

For both problems, the delayed choice mechanism applies. It is not optimal for the control, to steer the system into a direction, since noise can cancel these effects. If the noise is too large, the no control is applied at all.

For the controlled random walk problem, it was easy to gain insights into the dynamics of the system by looking at the formula. The results were verified by running many simulations. For extremely large noise ($v \to \infty$), the optimal control goes to zero. The more $t \to T$, the more control is performed. The

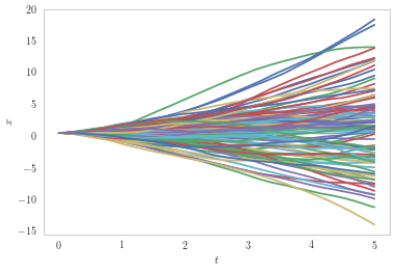

Figure 7: 100 simulations for $x_0 = 0.5$, $v_0 = 0$, $\nu = 1$, $T = 5$ and $g = 0$.

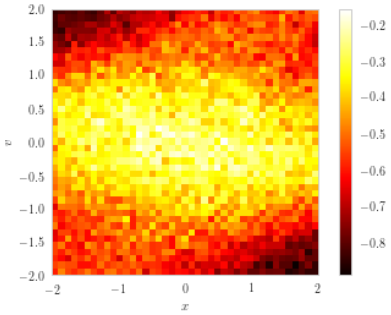

Figure 8: The optimal cost-to-go at $t = 0$, $\nu = 4$, $g = -10$ and without control (so $u = 0$) computed by calculating 100 simulations per (x_0, v_0) pair.

delayed control mechanism is caused by the fact that control has more influence at the end state at a relatively late point in time than control which is performed at the beginning of the time.

The second-order differential equations in the mountain car problem made the problem quite hard analytically. However, by approximating the optimal control by using MCMC sampling, we obtained satisfying results.

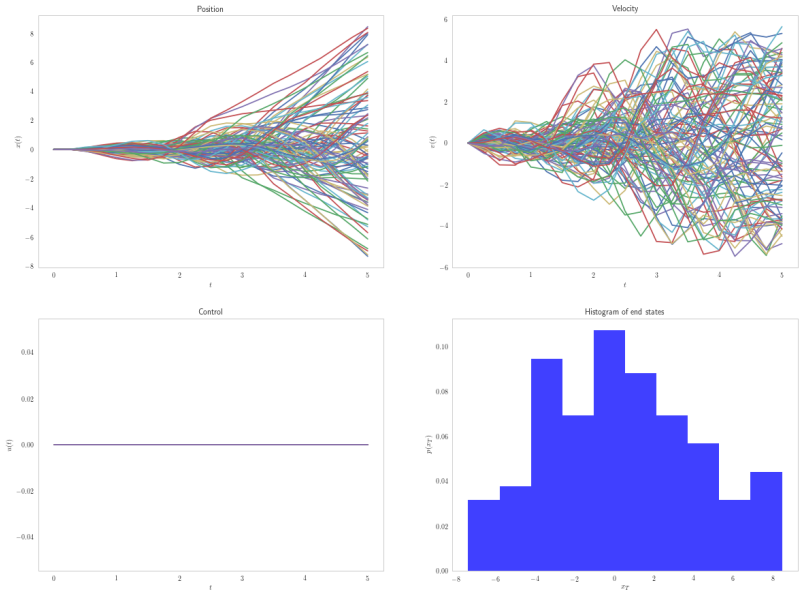

Figure 9: 100 simulations without control (so $u = 0$) with $(x_0, v_0) = (0, 0)$, $A = -1$, $R = 1$, $\nu = 0.2$, $T = 5$, $g = -10$ and $dt = 0.25$.

6 Appendix

6.1 Code for the controlled random walk (MATLAB)

```
function out = Control(assignment, action, varargin)
    % Control  Script for performing tasks related to the Control theory
    % lectures. The first argument specifies the assignment to run. The
    % seconds argument specifies which action to run.
    %
    % The options for the first argument are the following:
    %   'crw': Controlled Random Walk
    %   'mcp': Mountain Car Problem
    %
    % The following global options are optional:
    %    seed  The seed for the random number generator to use
    %          (default: -1 which will not use any seed).
    %    path  The path for the output (default: 'output/').
    %
    % The following script calls are possible:
    %
    %   Control('crw', 'simulate')
    %     Run a simulation and store the result in the given output file.
    %     Options (optional):
    %       noise          The noise level.
    %       horizon        The end-time.
    %       dt             Change in time per step.
```

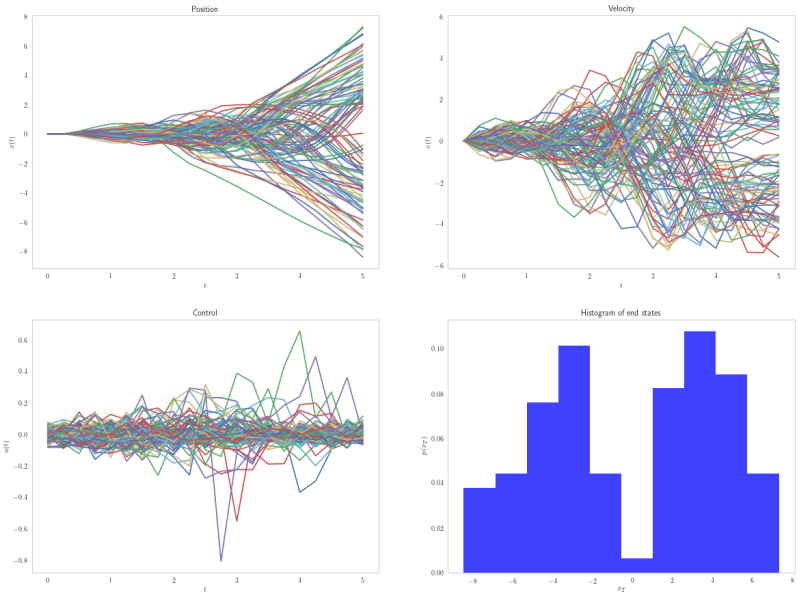

Figure 10: 100 simulations with optimal control (computed with 100 samples per timestep) with $(x_0, v_0) = (0, 0)$, $A = -1$, $R = 1$, $\nu = 0.2$, $T = 5$, $g = -10$ and $dt = 0.25$.

```
%       trials          Number of trials.
%       show_state      When true, a plot of the states will be shown.
%       show_control    When true, a plot of the optimal control will be
%                       shown.
%     Example calls:
%       Control('crw', 'simulate');
%       Control('crw', 'simulate', 'noise', 0.1, 'horizon', 1, 'dt', 0.01,
%
%   Control('crw', 'explore')
%     Options (optional):
%       noise_min           Minimum noise level.
%       noise_stepsize      Noise step size.
%       noise_max           Maximum noise level.
%       horizon_min         Minimum horizon.
%       horizon_stepsize    Horizon step size.
%       horizon_max         Maximum horizon.
%       dt                  Change in time per step.
%       trials              Number of trials.
%
%     Example calls:
%       Control('crw', 'explore', 'noise_min', 1.0, 'noise_max', 10.0, 'noi
%       Control('crw', 'explore', 'noise_min', 1.0, 'noise_max', 15.0, 'noi
%       Control('crw', 'explore', 'noise_min', 1.0, 'noise_max', 10.0, 'noi

% Check arguments and global options
```

```
    p = inputParser;
    p.KeepUnmatched = true;
    addRequired(p, 'assignment', @(x) any(validatestring(x, {'crw', 'mcp'})));
    addRequired(p, 'action', @ischar);
    addParameter(p, 'seed', -1, @isnumeric);
    addParameter(p, 'path', 'output/', @ischar);
    parse(p, assignment, action, varargin{:});

    % Set the seed for the RNG (if not -1)
    if p.Results.seed ~= -1
        rng(p.Results.seed);
    end

    % Create the output directory if it does not exist
    path = p.Results.path;
    if exist(path, 'dir') ~= 7
        mkdir(path);
    end

    % Execute code for the Controlled Random Walk assignment
    if strcmp(p.Results.assignment, 'crw')
        CRW(p.Results.action, path, varargin{:});
    elseif strcmp(p.Results.assignment, 'mcp')

    end
end

%% Mountain Car problem
function out = MCP(action, path, varargin)

end

%% Controlled Random Walk problem
function out = CRW(action, path, varargin)
    % Check arguments
    p = inputParser;
    p.KeepUnmatched = true;
    addRequired(p, 'action', @(x) any(validatestring(x, {'simulate', 'explore'}
    parse(p, action);

    % Run the simulation action
    if strcmp(action, 'simulate')
        CRW_simulate(path, varargin{:});
    end

    if strcmp(action, 'explore')
        CRW_explore(path, varargin{:});
    end
end
```

```
function out = CRW_simulate(path, varargin)
    % Parse the options
    p = inputParser;
    p.KeepUnmatched = true;
    addParameter(p, 'noise', 0.1, @isnumeric);
    addParameter(p, 'seed', -1, @isnumeric);
    addParameter(p, 'horizon', 1, @isnumeric);
    addParameter(p, 'dt', 0.01, @isnumeric);
    addParameter(p, 'trials', 1, @isnumeric);
    addParameter(p, 'show_state', true, @islogical);
    addParameter(p, 'show_control', true, @islogical);
    parse(p, varargin{:});

    % Store the options in variables
    noise = p.Results.noise;
    seed = p.Results.seed;
    horizon = p.Results.horizon;
    dt = p.Results.dt;
    trials = p.Results.trials;
    show_state = p.Results.show_state;
    show_control = p.Results.show_control;

    % Run the simulations
    steps = int32(horizon / dt) + 1;
    states = zeros(trials, steps);
    controls = zeros(trials, steps);
    for trial = 1:trials
        [x, t, steps, xi, u_star] = crw_simulate_optimal_control(noise, horizon
        states(trial, :) = x;
        controls(trial, :) = u_star;
    end

    % Make a figure
    figure('Visible', 'off');

    % Plot the state during the simulation
    if show_state && show_control
        subplot(2, 1, 1);
    end
    if show_state
        plot(t, states);
        title('Simulation');
        xlabel('Time');
        ylabel('State');
        axis([min(t), max(t), -1.5, 1.5]);
    end

    % Plot the optimal control
    if show_state && show_control
        subplot(2, 1, 2);
```

```
    end
    if show_control
        plot(t, controls);
        title('Optimal Control');
        xlabel('Time');
        ylabel('Control');
        axis([min(t), max(t), -1.5, 1.5]);
    end

    % Save the figure
    filename = strcat(path, 'crw_simulate');
    filename = strcat(filename, '_noise=', num2str(noise));
    filename = strcat(filename, '_horizon=', num2str(horizon));
    filename = strcat(filename, '_dt=', num2str(dt));
    filename = strcat(filename, '_trials=', num2str(trials));
    filename = strcat(filename, '_show-state=', num2str(show_state));
    filename = strcat(filename, '_show-control=', num2str(show_control));
    filename = strcat(filename, '_seed=', num2str(seed));
    filename = strcat(filename, '.png');
    saveas(gcf, filename);
end

function out = CRW_explore(path, varargin)
    % Parse the options
    p = inputParser;
    p.KeepUnmatched = true;
    addParameter(p, 'noise_min', 0.1);
    addParameter(p, 'noise_max', 2.0);
    addParameter(p, 'noise_stepsize', 0.1);
    addParameter(p, 'horizon_min', 1.0);
    addParameter(p, 'horizon_max', 5.0);
    addParameter(p, 'horizon_stepsize', 2.0);
    addParameter(p, 'seed', -1, @isnumeric);
    addParameter(p, 'dt', 0.01, @isnumeric);
    addParameter(p, 'trials', 1, @isnumeric);
    parse(p, varargin{:});

    % Store the options in variables
    noise_min = p.Results.noise_min;
    noise_max = p.Results.noise_max;
    noise_stepsize = p.Results.noise_stepsize;
    horizon_min = p.Results.horizon_min;
    horizon_max = p.Results.horizon_max;
    horizon_stepsize = p.Results.horizon_stepsize;
    seed = p.Results.seed;
    dt = p.Results.dt;
    trials = p.Results.trials;

    % Calculate the noise and horizon
    noise_steps = floor((noise_max - noise_min) / noise_stepsize) + 1;
```

```
horizon_steps = floor((horizon_max - horizon_min) / horizon_stepsize) + 1;
horizon_data = zeros(horizon_steps, 1);

% Create the figure
figure('Visible', 'off');

% Create the plots
hold on;
for horizon_step = 1:horizon_steps
    % Initialize the noise, horizon and error
    noise_data = zeros(noise_steps, 1);
    error_data = zeros(noise_steps, 1);
    horizon = horizon_min + (horizon_step - 1) * horizon_stepsize;
    horizon
    horizon_data(horizon_step, 1) = horizon;

    % Loop through all noise levels
    for noise_step = 1:noise_steps
        noise = double(noise_min + (noise_step - 1) * noise_stepsize);
        noise
        error = 1;

        % Loop through all trials
        for trial = 1:trials
            [x, t, steps, xi, u_star] = crw_simulate_optimal_control(noise,
            error = error + log(0.5 * normpdf(x(steps), -1, 0.5) + 0.5 * no
        end

        % Update the error and noise data
        error_data(noise_step, 1) = error / double(noise_steps);
        noise_data(noise_step, 1) = noise;
    end
    % Create the plot
    plot(noise_data(:, 1), error_data(:, 1), 'DisplayName', strcat('T = ',
end
hold off;

% Update plot attributes
legend(gca, 'show', 'Location', 'NorthWest');
title('Explore parameter space');
xlabel('\nu');
ylabel('Error');

% Save the figure
filename = strcat(path, 'crw_explore');
filename = strcat(filename, '_noise-min=', num2str(noise_min));
filename = strcat(filename, '_noise-stepsize=', num2str(noise_stepsize));
filename = strcat(filename, '_noise-max=', num2str(noise_max));
filename = strcat(filename, '_horizon-min=', num2str(horizon_min));
filename = strcat(filename, '_horizon-stepsize=', num2str(horizon_stepsize)
```

```
    filename = strcat(filename, '_horizon-max=', num2str(horizon_max));
    filename = strcat(filename, '_dt=', num2str(dt));
    filename = strcat(filename, '_trials=', num2str(trials));
    filename = strcat(filename, '_seed=', num2str(seed));
    filename = strcat(filename, '.png');
    saveas(gcf, filename);
end

function [x, t, steps, noise, u_star] = crw_simulate_optimal_control(nu, T, dt)
    % CRW_SIMULATE  Simulate optimal control in the Control Random Walk
    % problem with noise parameter NU and horizon T with change in time DT.
    % It returns the states x(t=0), ..., x(t=T), the times t=0, ..., t=T,
    % the number of simulation steps, the noise from t=0 to t=T and the
    % optimal control u*(t=0), ..., u*(t=T).

    % Compute the number of steps
    steps = int64(T / dt) + 1;

    % Initialize the variables
    x = zeros(steps, 1);
    noise = zeros(steps, 1);
    u_star = zeros(steps, 1);
    t = zeros(steps, 1);

    % Perform the simulation
    for step = 2:steps
        % Compute the optimal control
        u_star(step) = crw_optimal_control(x(step - 1), t(step - 1), nu, T);

        % Compute the noise
        dxi = sqrt(dt) * normrnd(0, nu);

        % Compute the change in state
        dx = u_star(step) * dt + dxi;

        % Update the state
        x(step) = x(step - 1) + dx;

        % Update the time
        t(step) = t(step - 1) + dt;
    end
end

function u_star = crw_optimal_control(x, t, nu, T)
    % CRW_OPTIMAL_CONTROL  Compute the optimal control for the Controlled
    % Random Walk problem at state X and time T for noise parameter NU and
    % horizon T.
    u_star = (tanh(x / (nu * (T - t))) - x) / (T - t);
end
```

6.2 Code for the mountain car problem (iPython Notebook)

6.2.1 Imports

```
import numpy as np
import math
import matplotlib.pyplot as plt
from matplotlib import rc
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation, rc
from IPython.display import HTML

% matplotlib inline

rc('text', usetex=True)
sns.set_style("whitegrid", {'axes.grid' : False})
```

6.3 Definitions

```
def L(x):
    # The definition of the valley
    return -1 - 0.5 * (np.tanh(2 * x + 2) - np.tanh(2 * x - 2))

def sech(x):
    # The sech function
    return 2. / (np.exp(x) + np.exp(-x))

def dLdx(x):
    # dL / dx evaluated at point x
    return np.power(sech(2 * x + 2), 2) - np.power(sech(2 * x - 2), 2)

def F(x, g):
    # The gravitational force
    return -g * dLdx(x) / np.sqrt(1 + np.power(dLdx(x), 2))
```

6.4 Base class for simulations

```
class MountainCarProblem:

    def __init__(self, x_0, v_0, A=-1, R=1, nu=1, T=1, g=1, dt=0.01, u=0, t_0=
        """
        Initialize the Mountain Car Problem.

        x_0: The initial position.
        v_0: The initial velocity.
        A:   The reward for ending outside the valley (default: -1).
        R:   Relation between the noise level and the optimal control (default:
        nu:  Noise level.
```

```
        T:    Horizon.
        g:    Gravitational constant (default: 1).
        u:    Constant control (default: 0). For dynamic control, implement the
        t_0: The start time of the system such that 0 <= t_0 <= T (default: t_0
        """
        self.x_0, self.v_0, self.A, self.R, self.nu, self.T, self.g, self.dt, s
        self.x_min, self.x_max = -2, 2
        self.x = self.x_0
        self.v = self.v_0
        self.t = self.t_0

    def run(self):
        """
        Run the simulation. It will return the following result:

        [xs, vs, ts, dus, dxis] in which:
        xs:   The computed positions over time.
        vs:   The computed velocities over time.
        ts:   The computed times (simply ranging from 0 to T).
        dus:  The computed controls over time.
        dxis: The computed noise per timestep.
        """
        xs, vs, ts, dus, dxis = [], [], [], [], []
        x, v, t = self.x_0, self.v_0, self.t_0
        steps = math.floor((self.T - self.t_0) / self.dt) + 1

        for step in range(steps):
            dudt = self.compute_control()
            noise = 0 if self.nu == 0 else np.random.normal(0, self.nu)
            dxi = np.sqrt(self.dt) * noise

            xs.append(x)
            vs.append(v)
            ts.append(t)
            dus.append(dudt)
            dxis.append(dxi)

            dx = v * self.dt
            dv = F(x, self.g) * self.dt + dudt + dxi

            x += dx
            v += dv
            t += self.dt

            self.x = x
            self.v = v
            self.t = t

        phi = 0 if self.x_min < self.x and self.x < self.x_max else self.A
```

```
        return xs, vs, ts, dus, dxis, phi

    def compute_control(self):
        """
        Compute the control. This method has access to state variables self.x (
        """
        return self.u
```

6.5 Optimal Control

```
class OptimalControlMCP(MountainCarProblem):
    def run_uncontrolled(self, x, v, t):
        mcp = MountainCarProblem(x_0=x, v_0=v, A=self.A, R=self.R, nu=self.nu,
        return mcp.run()

    def compute_control(self):
        w = []
        xi = []
        N = 100
        for _ in range(N):
            xs, vs, ts, dus, dxis, phi = self.run_uncontrolled(self.x, self.v,
            step = math.floor(self.t / self.dt)
            if len(dxis) > 0:
                dxis = np.asarray(dxis)
                if phi == 0:
                    w.append(0)
                else:
                    w.append(1. / N * np.exp(-phi))
                xi.append(dxis[0])
            else:
                w.append(0)
        w = np.array(w)
        xi = np.array(xi)
        ones = np.ones(w.shape)
        denom = np.dot(w, ones)
        if denom == 0:
            return 0
        udt = np.dot(w, xi) / denom
        return udt
```

6.6 Approximating $J(x, v, t = 0)$

```
xs = np.linspace(-2, 2, 41)
vs = np.linspace(-2, 2, 41)
n = 100
Js = []

for x in xs:
    row = []
    for v in vs:
```

```
            S = 0
            for mu in range(n):
                mcp = MountainCarProblem(x, v, nu=4, g=-10, u=0, A=-1)
                _, _, ts, dus, dxis, phi = mcp.run()
                S += np.exp(-phi)
            S /= n
            J = -np.log(S)
            row.append(J)
        Js.append(row)

fig, ax = plt.subplots(1, 1)
plt.imshow(Js, cmap='hot', extent=(-2, 2, -2, 2))
plt.colorbar()
plt.xlabel('$x$')
plt.ylabel('$v$')
```