AER1517 Control for Robotics

Assignment 2: Markov Decision Processes, Classical Reinforcement Learning & Model Predictive Control

Prof. Angela Schoellig

Institute for Aerospace Studies
UNIVERSITY OF TORONTO

VECTOR INSTITUTE

DYNAMIC SYSTEMS LAB

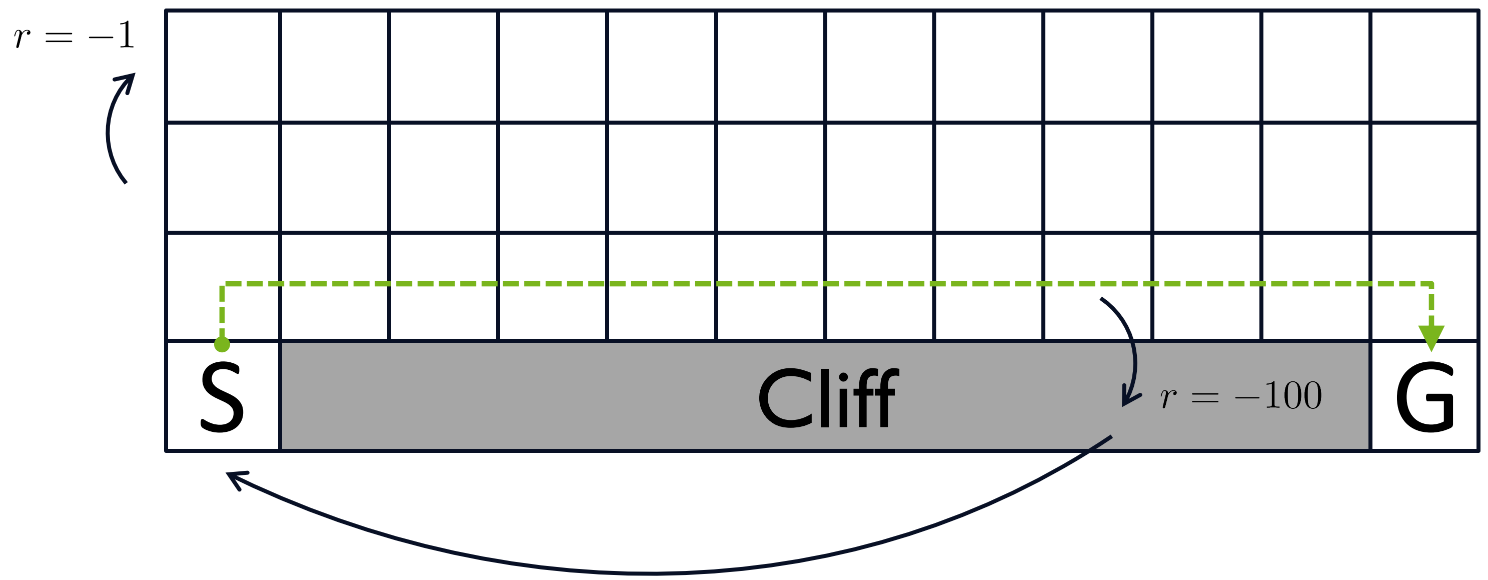
General Information

- Download handout and script templates from Quercus
  - Problem 2.1 Grid World
  - Problem 2.2 Mountain Car
- Due on Mar. 24 (Tuesday) 23:59
- Submission through *Gradescope*
  - A single PDF with solutions and requested scripts
  - Both typed and scanned handwritten solutions are accepted
- Office hour: Mar. 19 (Thursday) 14:00 via Hangout or in-person @UTIAS
- Submit email questions by Mar. 15 or Mar. 22 (Sunday) --- open issues are discussed during lectures

Problem 1.1 Grid World | Overview

- Goal: Plan a path from ‘S’ to ‘G’ without transitioning to obstacle states

Reward of -1 for
each state transition

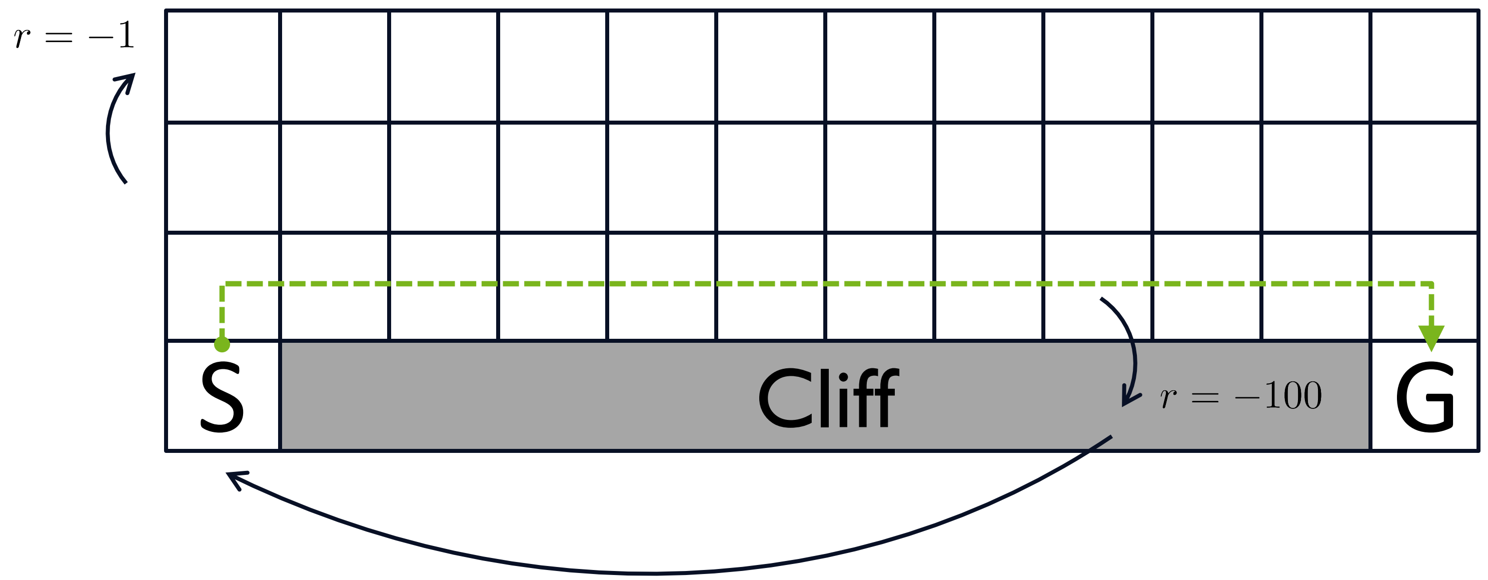


Episode terminates when
the goal state is reached

Reward of -100 if transitioning to obstacle states
(‘Cliff’); agent returns to the start state ‘S’

Problem 1.1 Grid World | Overview

- Goal: Plan a path from 'S' to 'G' without transitioning to obstacle states



- Task: Solve the grid world problem with
  - Model-based approach: Generalized Policy Iteration (GPI)
  - Sample-based approaches: Monte-Carlo (on-policy) and Q Learning (off-policy)
  - **[Extra Credit]** Linear Programming

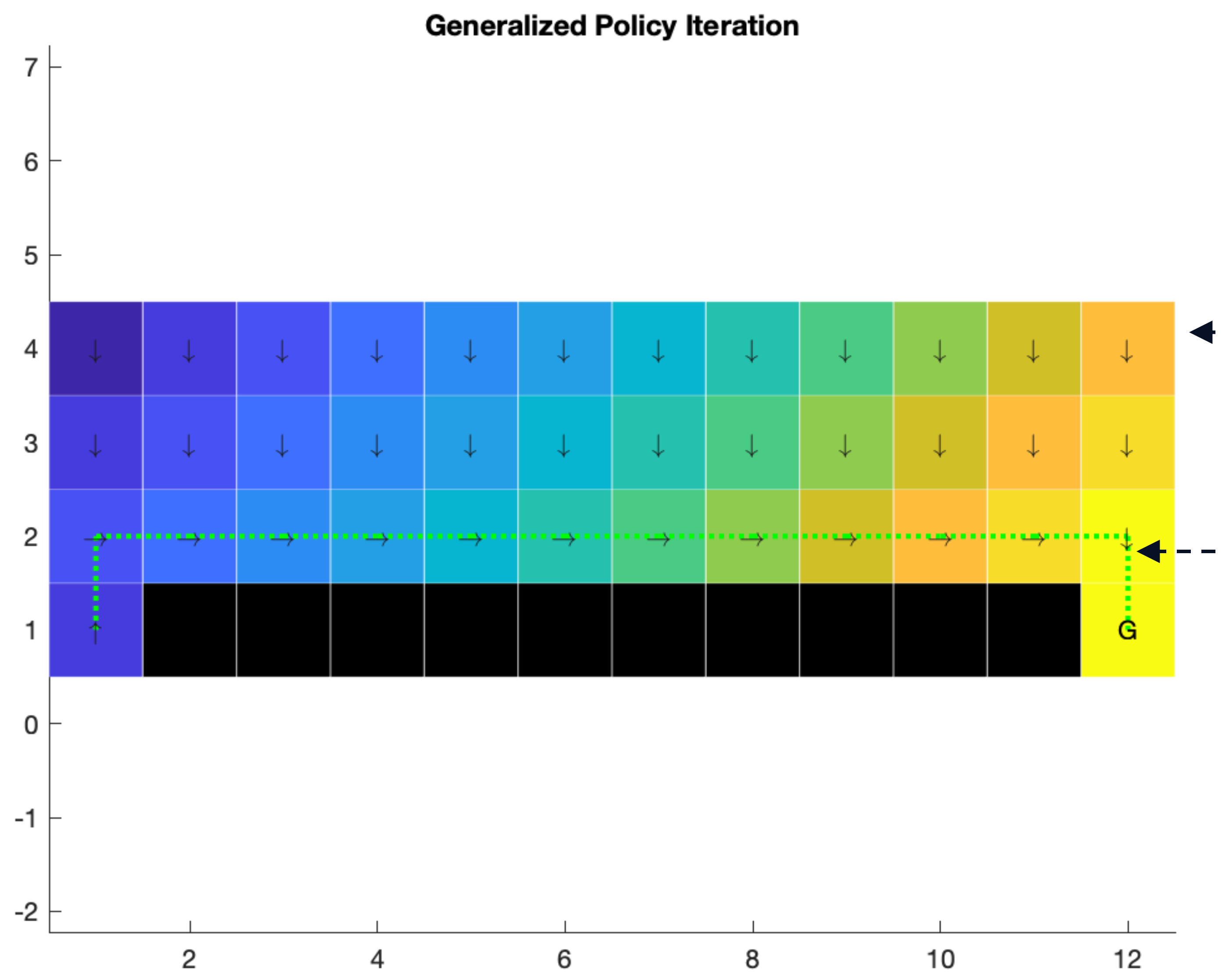
Problem 1.1 Grid World | Code Structure

main_p1_gw.m

- Load model
- References
- Default parameters
- Algorithm (e.g., GPI)
- Visualization

```matlab
%% General
% Load world
load('./gridworld_model/grid_world');

% Add path
addpath(genpath(pwd));

% Result and plot directory
save_dir = './results/';
mkdir(save_dir);

%% Problem 2.1: (a) Generalized Policy Iteration (GPI)
% Instruction: Implement the GPI algorithm to solve the grid world problem
% Reference: Section 2.8 of [1]

% Parameters of the GPI algorithm
precision_pi = 0.1;
precision_pe = 0.01;
max_ite_pi = 100;
max_ite_pe = 100;

% ====================== [TODO] GPI Implementation ======================
% Complete implementation in 'generalized_policy_iteration'
[v_gpi, policy_gpi] = generalized_policy_iteration(world, precision_pi, ...
    precision_pe, max_ite_pi, max_ite_pe);
% =======================================================================

% Visualization
plt_title = 'Generalized Policy Iteration';
plt_path = true;
plt_gpi = visualize_gw_solution(world, v_gpi, policy_gpi, ...
    plt_title, plt_path);

% Save results and figure to report
save(strcat(save_dir, 'gpi_results.mat'), 'v_gpi', 'policy_gpi');
saveas(plt_gpi, strcat(save_dir, 'gpi_plot.png'), 'png');
```

Problem 1.1 Grid World | Example Result with GPI



Heatmap corresponds to state value function (or, cost-to-go)

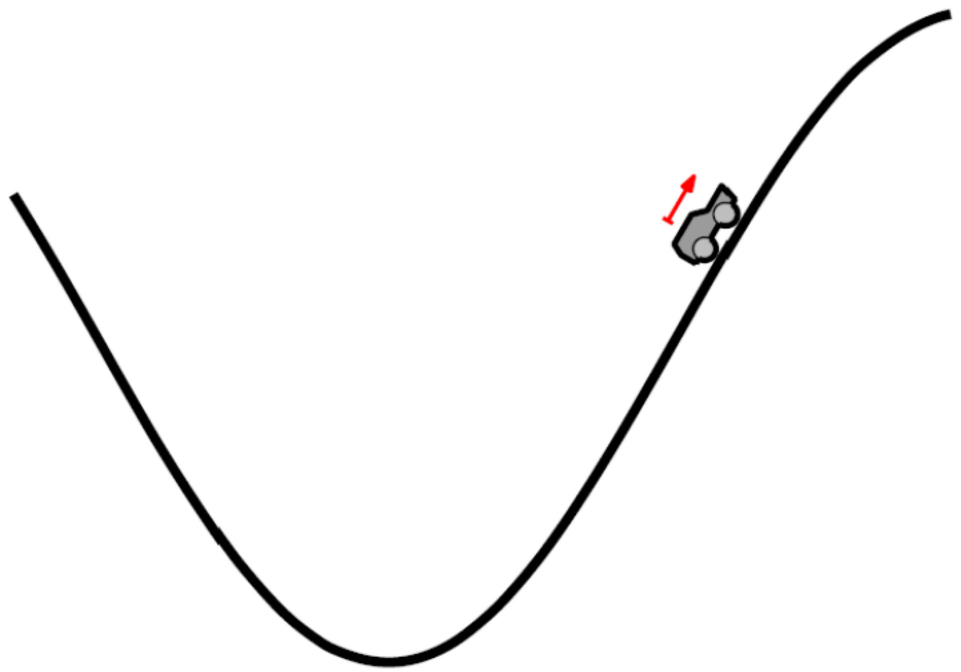
Planned path from the start state to the goal state

- Goal: Drive an under-powered car to the top of the hill
  - States: position x and velocity v
  - Input: acceleration a
  - System dynamics:

$$v_{k+1} = v_k + 0.001 a_k - 0.0025 \cos(3x_k)$$
$$x_{k+1} = x_k + v_{k+1}$$

  - Bounds on states and input:

$$x_k \in [-1.2, 0.5], v_k \in [-0.07, 0.07], a_k \in [-1, 1]$$



- Task:
  - Create stochastic Markov Decision Process (MDP) and solve with classical RL
  - Formulate and solve with Model Predictive Control (MPC)

Problem 1.2 Mountain Car | Code Structure (MDP Design)

DYNAMIC SYSTEMS LAB

create_mountain_car.m

- Build MDP

create_mountain_car.m | main_p2_mc_rl.m

```
%% Problem 2.2:  (a)-(b)  Create stochastic MDPs for the mountain car problem
% Instruction: Implement the Nearest Neighbour and Linear Interpolation
%              approaches for creating discrete stochastic MDPs
% Reference: see [4] for linear_interp implementation

% ======================== [TODO] Discretization ========================
% Complete implementations in 'build_stochastic_mdp_nn' and
% 'build_stochastic_mdp_li'
switch mdp_approach
    case 'nearest_neighbour'
        % for each state-action pair run generate multiple samples
        num_samples = 50;

        % build stochastic MDP based on nearest neighbour
        [T, R] = build_stochastic_mdp_nn(world, T, R, num_samples);

    case 'linear_interp'
        % build stochastic MDP based on linear interpolation
        [T, R] = build_stochastic_mdp_li(world, T, R);
end
% =======================================================================
```

main_p2_mc_rl.m

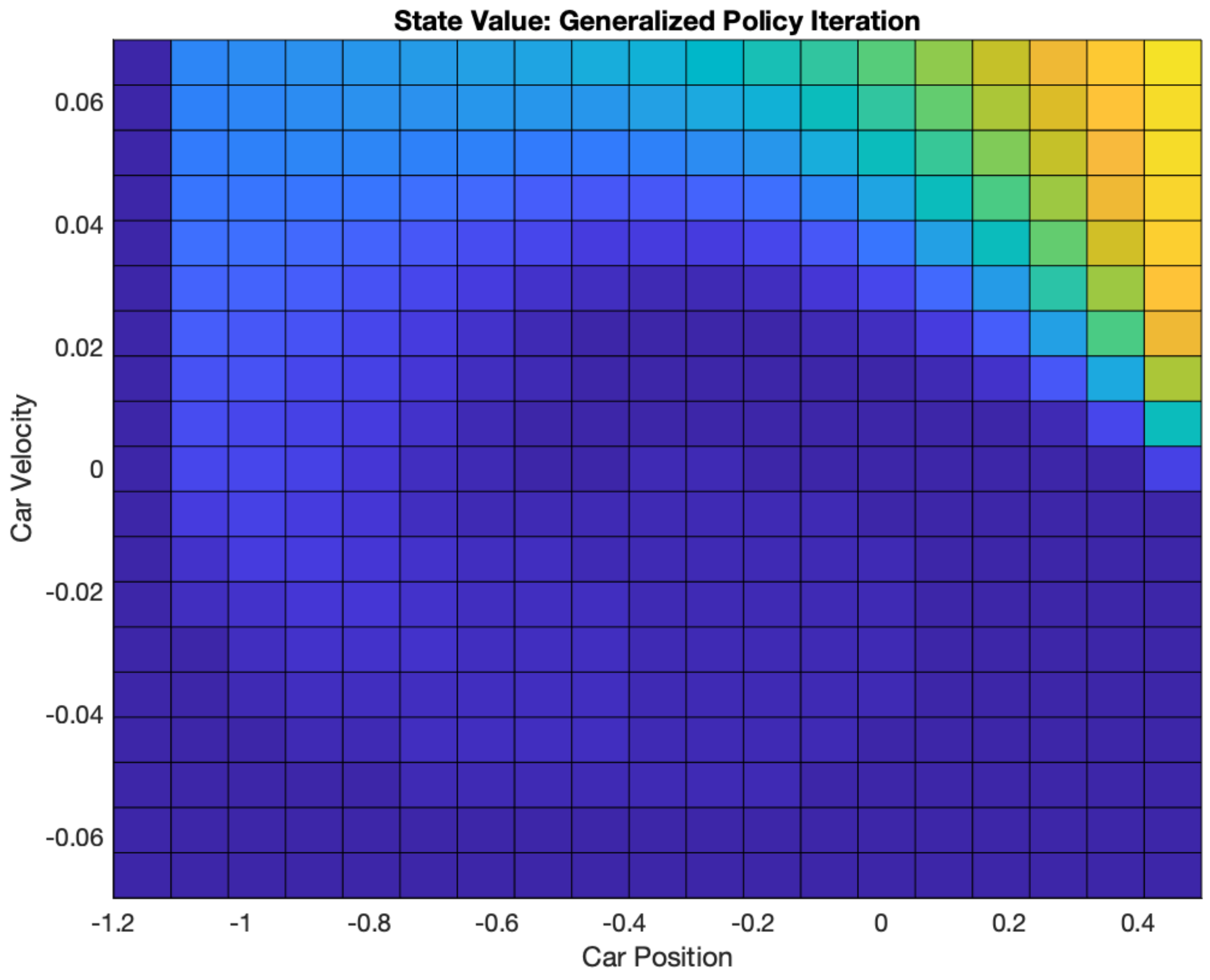
- Load model
- Solve MDP
- Visualization

create_mountain_car.m | main_p2_mc_rl.m

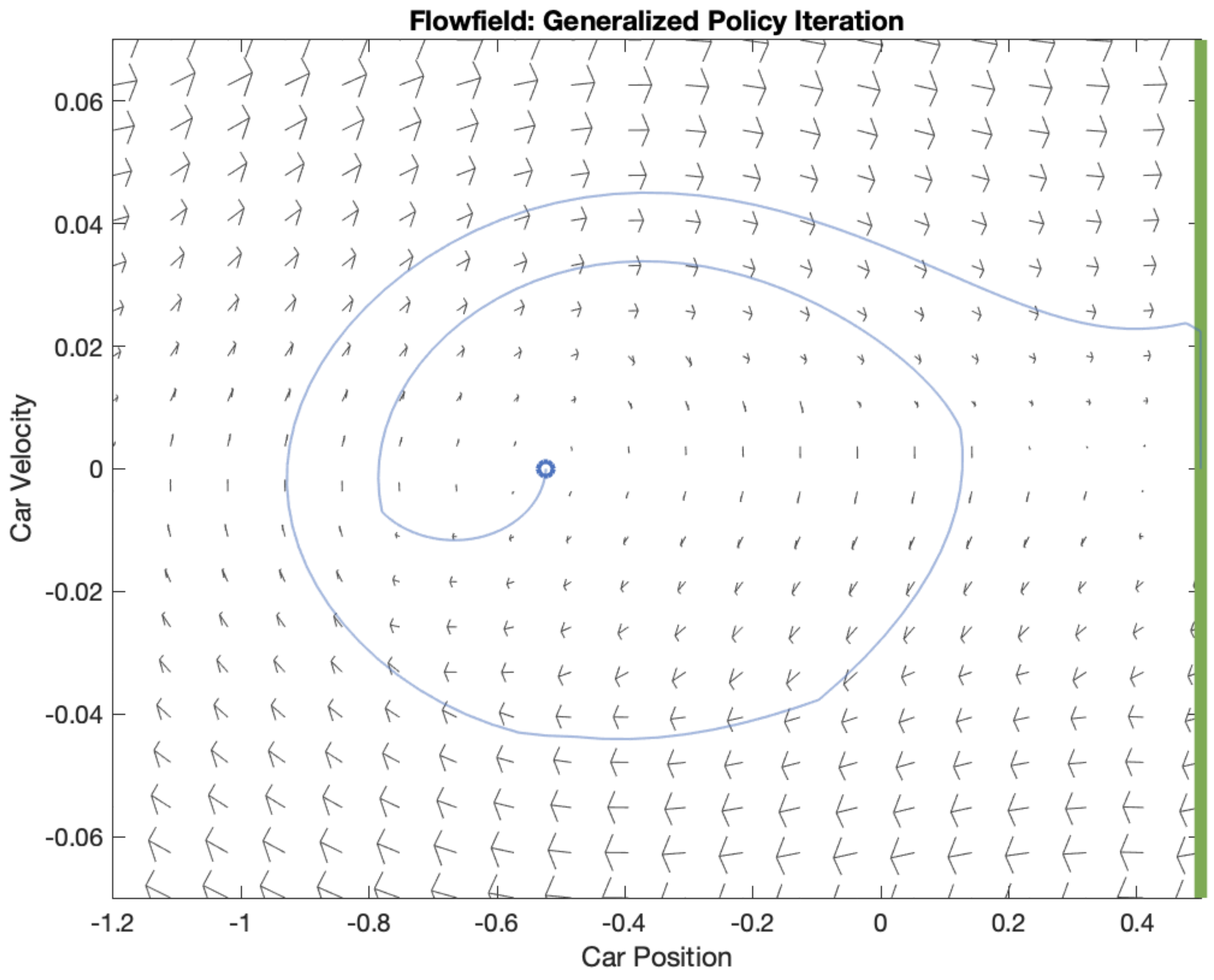
```

%% Problem 2.2 (a)-(b) Create stochastic MDPs for the mountain car problem
% [TODO] Load mountain car model
% change model name correspondingly:
%     (a) 'mountain_car_nn' for the nearest neighbour method
%     (b) 'mountain_car_li' for the linear interpolation approach
load('./mountain_car_model/mountain_car_nn');

%% Generalized policy iteration
% Algorithm parameters
precision_pi = 0.1;
precision_pe = 0.01;
max_ite_pi = 100;
max_ite_pe = 100;

% Solve MDP
[v_gpi, policy_gpi] = generalized_policy_iteration(world, precision_pi, ...
    precision_pe, max_ite_pi, max_ite_pe);

% Visualization
plot_value = true;
plot_flowfield = true;
plot_visualize = true;
plot_title = 'Generalized Policy Iteration';
hdl_gpi = visualize_mc_solution(world, v_gpi, policy_gpi, plot_value, ...
    plot_flowfield, plot_visualize, plot_title, save_dir);
```

Problem 1.2 Mountain Car | Example Result with RL

- Solution with GPI

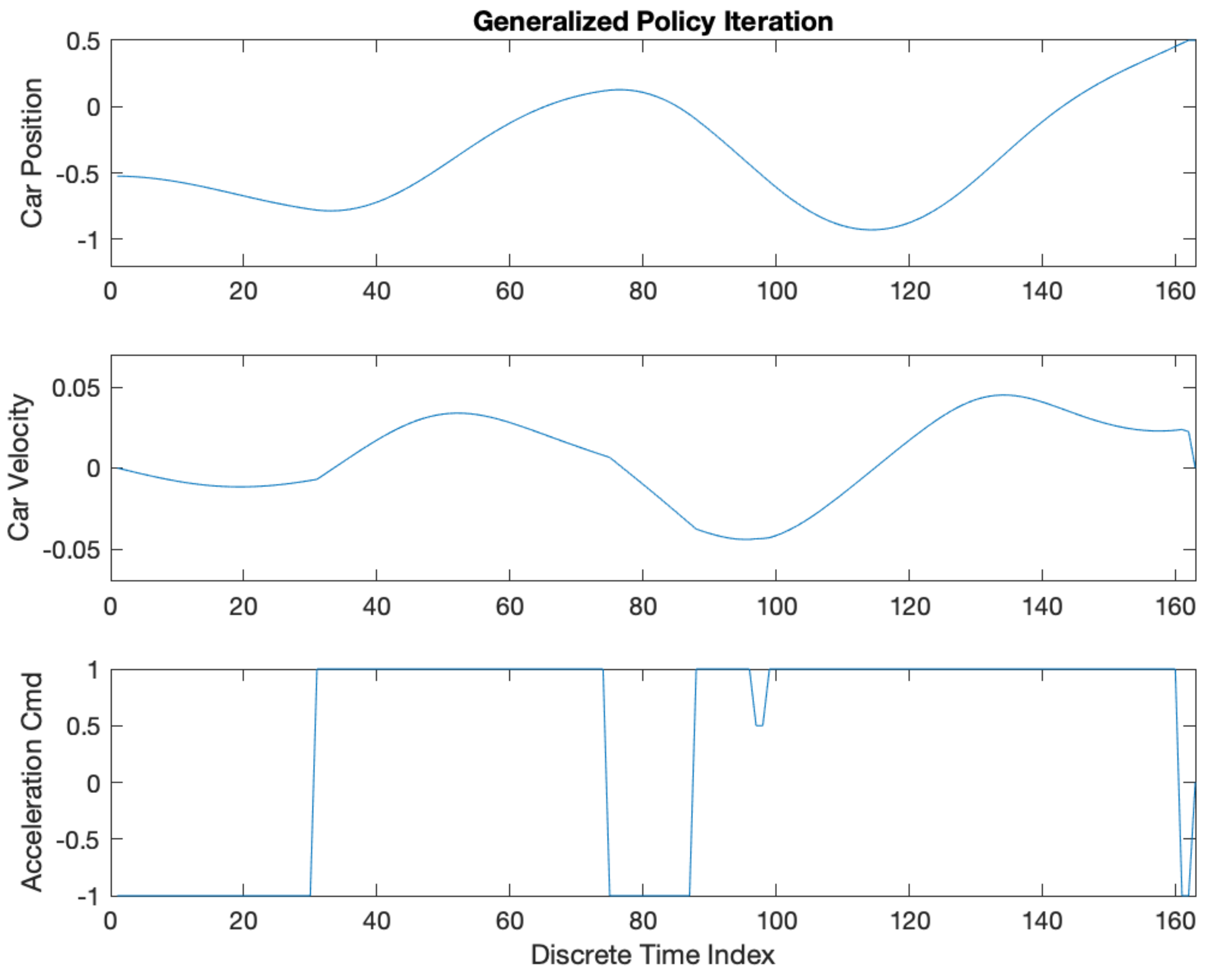


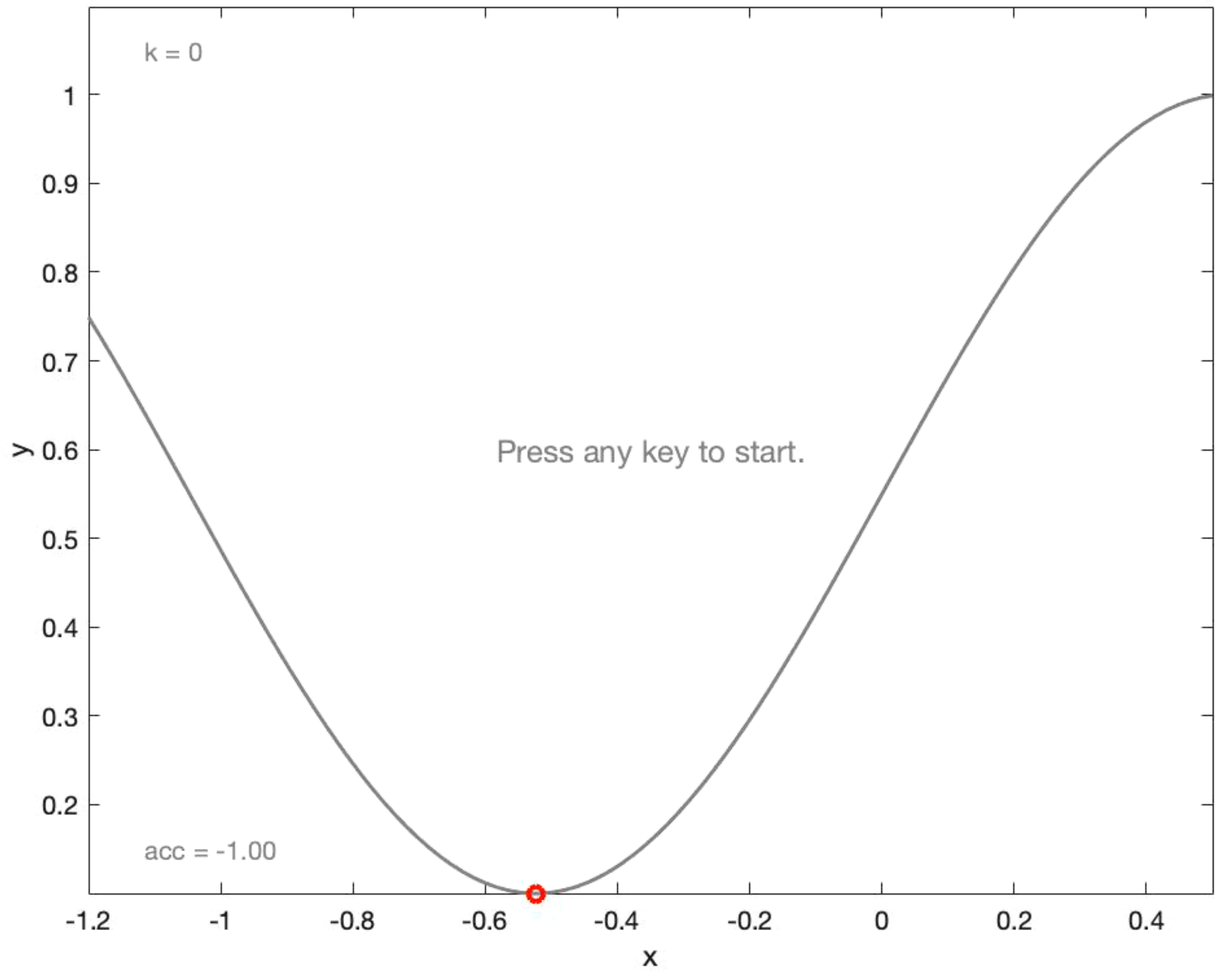
State value function (or, cost-to-go)



Resulting flowfield from the optimal policy

- Solution with GPI





Link to animation: https://drive.google.com/file/d/1OxFt75O-OB8dfkJXxCmuutl0iIVLPBK_/view

Problem 1.2 Mountain Car | Code Structure (MPC Design)

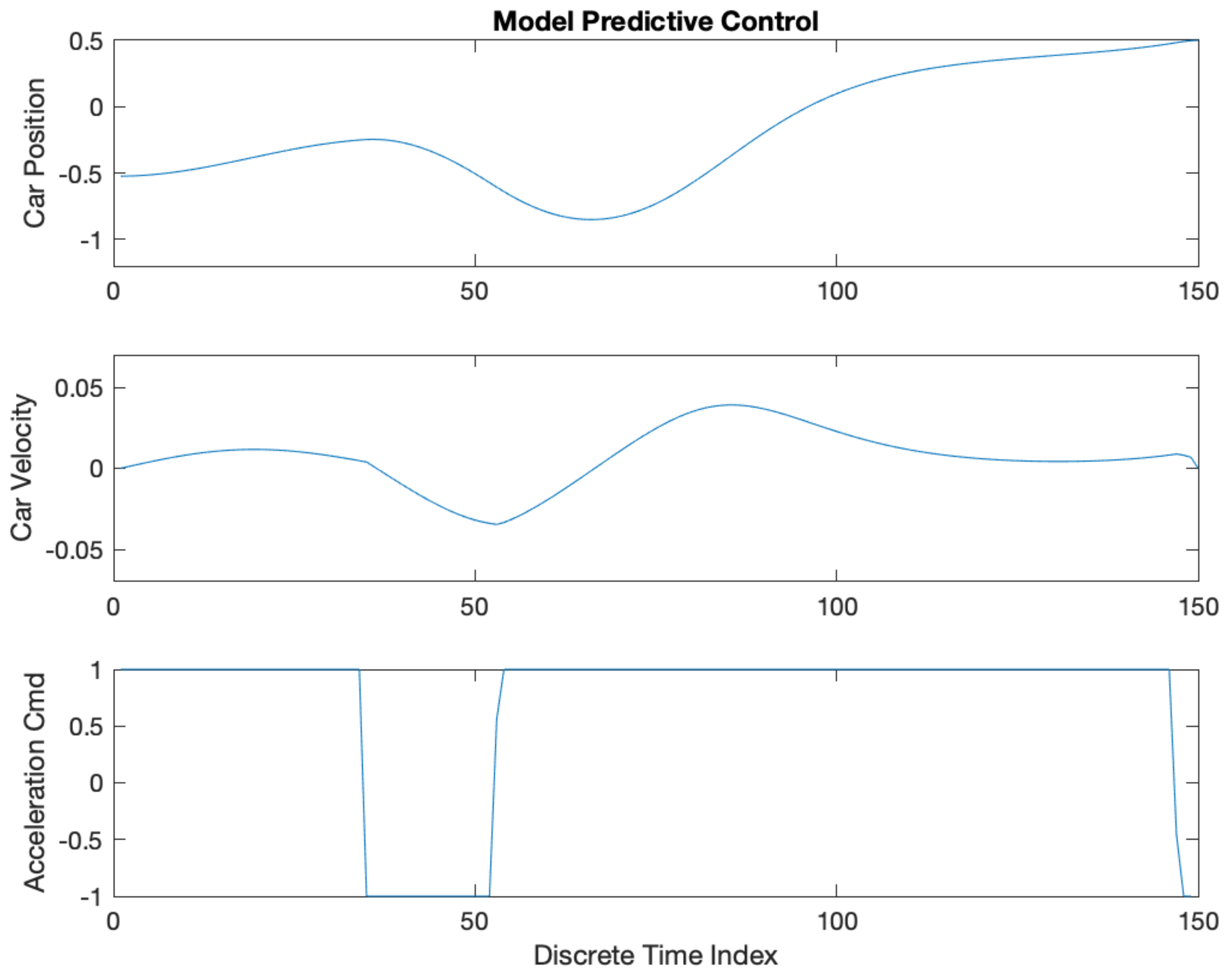
main_p2_mc_mpc.m

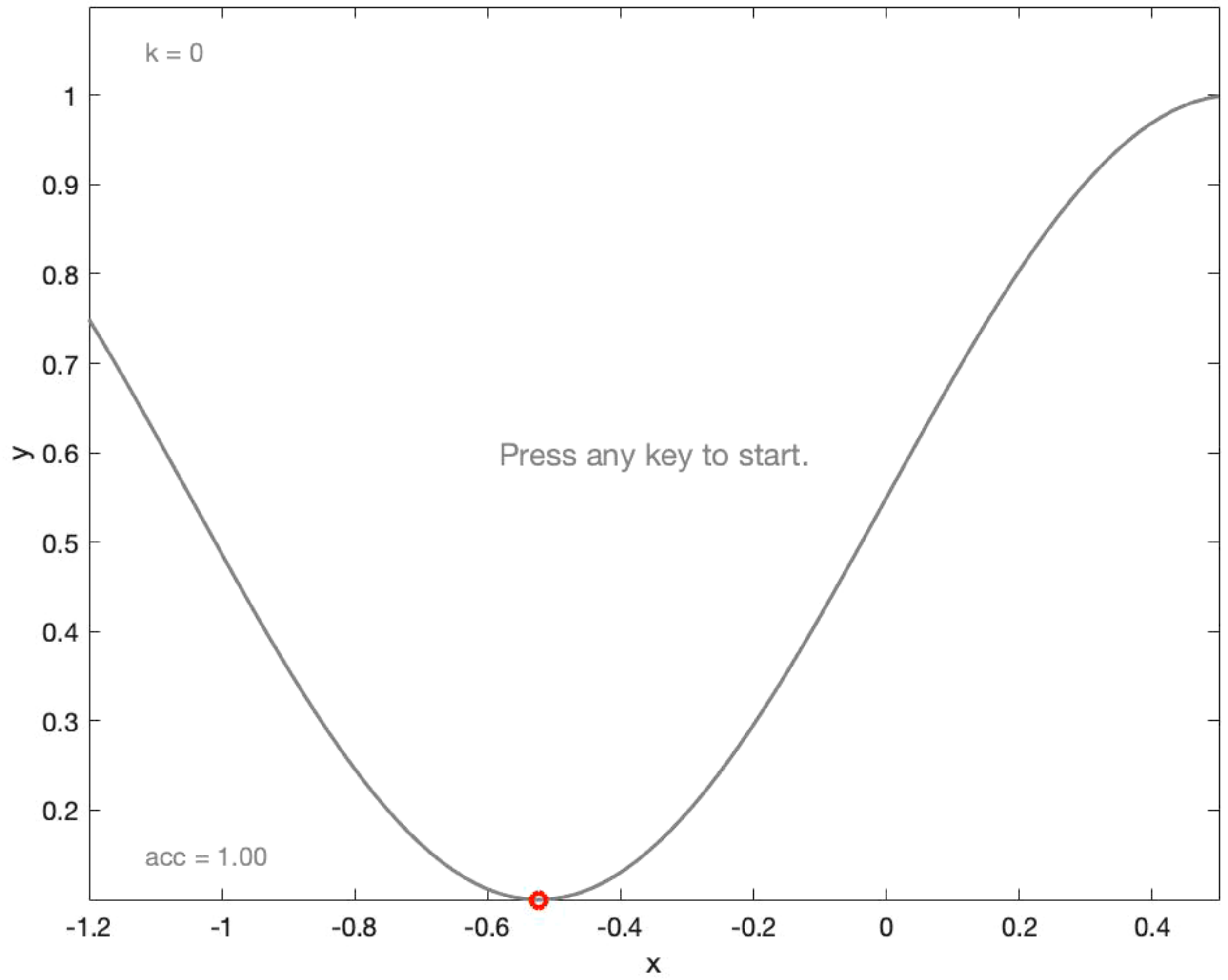
- Nonlinear MPC
  (first time step)

- Linearized Problem (SQP)
  (subsequent time steps)

```
            initial_guess = x;
        end

        % Cost function
        sub_states = [repmat(0,n_lookahead,1); ...
            repmat(goal_state, n_lookahead,1)];
        fun = @(x) (x - sub_states)'*S*(x - sub_states);

        % Temporary variables used in 'dyncons'
        save('params', 'n_lookahead', 'dim_state', 'dim_action');
        save('cur_state', 'cur_state');

        % Solve nonlinear MPC
        % x is a vector containing the inputs and states over the
        % horizon [input,..., input, state', ..., state']^T
        options = optimoptions(@fmincon, 'MaxFunctionEvaluations', ...
            1e5, 'MaxIterations', 1e5, 'Display', 'iter');
        [x,fval] = fmincon(fun, initial_guess, [], [], [], [], ...
            lb, ub, @dyncons, options);
    else
        % ================= [TODO] QP Implementation ================
        % Problem 2.2: (d) Quadratic Program optimizing state and
        % action over prediction horizon

        % Feedback state used in MPC updates
        % 'cur_state' or 'cur_state_noisy'
        cur_state_mpc_update = cur_state;

        % Solve QP (e.g., using Matlab's quadprog function)
        % Note 1: x is a vector containing the inputs and states over
        %         the horizon [input,..., input, state', ..., state']^T
        % Note 2: The function 'get_lin_matrices' computes the
        %         Jacobians (A, B) evaluated at an operation point


        % x = ...;
        % ==========================================================
    end
```

- Solution with MPC





Link to animation: https://drive.google.com/file/d/192yo6gTwrGZHf9pPFfXgQbuOFvVHESkC/view