Reinforcement Learning and Optimal Control for Robotics ROB-GY 6323 Exercise Series 2

October 27, 2024

Shantanu Ghodgaonkar
Univ ID: N11344563
Net ID: sng8399

Links to Jupyter Notebooks -

- Jupyter Notebook for code solution of Exercise 1
- Jupyter Notebook for code solution of Exercise 2
- Jupyter Notebook for code solution of Exercise 3

Exercise: 1 Consider the optimal control problem of Series 1 - Exercise 4 (control of a drone).

- Reusing the notebook of Series 1, write code to solve the same problem when the control is limited to $|5|$ for both rotors and the horizontal and vertical velocities are bounded to $2ms^{-1}$. To solve the resulting QP, use the cvxopt solver available with the qpsolvers library
- Show plots of all the states of the robot as a function of time
- Show plots of the optimal control as a function of time
- Compare the results with the results of Series 1 where no bounds were used

Answer Firstly, let us summarise the solution of the previous one. Let us reformulate the cost as

$$\sum_{n=0}^{N} \left(\frac{1}{2} x_n^T Q x_n + \frac{1}{2} u_n^T R u_n - x_{desired}^T Q x_n \right)$$

This gives us the optimal control problem,

$$\begin{aligned} \min_{x_0, u_0, x_1, u_1, \cdots} & \sum_{n=0}^{N} \left(\frac{1}{2} x_n^T Q x_n + \frac{1}{2} u_n^T R u_n - x_{desired}^T Q x_n \right) \\ \text{Subject to } & x_{n+1} = A x_n + B u_n \\ & x_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \\ \text{Where, } & x_{desired} = \begin{bmatrix} 3 & 0 & 3 & 0 & 0 & 0 \end{bmatrix}^T \end{aligned}$$

The problem can be rewritten in matrix form as

$$\begin{aligned} \min_y \ & \frac{1}{2} y^T G y + g^T y \\ \text{subject to} \quad & My = p \end{aligned}$$

Where,

$$y = \begin{bmatrix} x_0 \\ u_0 \\ x_1 \\ u_1 \\ \vdots \end{bmatrix} \quad G = \begin{bmatrix} Q & 0 & 0 & 0 & \cdots \\ 0 & R & 0 & 0 & \cdots \\ 0 & 0 & Q & 0 & \cdots \\ 0 & 0 & 0 & R & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \quad g^T = \begin{bmatrix} -x_{desired}^T Q & 0 & -x_{desired}^T Q & 0 & \cdots \end{bmatrix}$$

$$M = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 & \cdots \\ A & B & -I & 0 & 0 & 0 & \cdots \\ 0 & 0 & A & B & -I & 0 & \cdots \\ 0 & 0 & 0 & 0 & A & B & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \quad p = \begin{bmatrix} x_0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix}$$

Additionally now, we need to solve the same problem as above but given that the control is limited to $|5|$ for both rotors and the horizontal and vertical velocites are bounded to $|2|$ $m.s^{-1}$. Mathematically, this would mean,

$$u_{min} \leq u_n \leq u_{max} \quad \text{Where, } u_{min} = \begin{bmatrix} -5 & -5 \end{bmatrix}^T \quad \text{and} \quad u_{max} = \begin{bmatrix} 5 & 5 \end{bmatrix}^T$$

$$x_{min} \leq x_n \leq x_{max} \quad \text{Where, } x_{min} = \begin{bmatrix} 0 & -2 & 0 & -2 & 0 & 0 \end{bmatrix}^T \quad \text{and} \quad x_{max} = \begin{bmatrix} 0 & 2 & 0 & 2 & 0 & 0 \end{bmatrix}^T$$

The above equations can be rewritten as

$$\begin{aligned} -u_n &\leq u_{min} & u_n &\leq u_{max} \\ -x_n &\leq x_{min} & x_n &\leq x_{max} \end{aligned}$$

We can rewrite the above equations in matrix form as,

$$Hy \leq h$$

Where,

$$H = \begin{bmatrix} 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \quad y = \begin{bmatrix} x_0 \\ u_0 \\ x_1 \\ u_1 \\ \vdots \end{bmatrix} \quad h = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 2 \\ 5 \\ 5 \\ 5 \\ 5 \\ \vdots \end{bmatrix}$$

Thus, we have formulated the KKT System to obtain the optimal control problem,

$$\min_{x_0, u_0, x_1, u_1, \cdots} \sum_{n=0}^{N} \left(\frac{1}{2} x_n^T Q x_n + \frac{1}{2} u_n^T R u_n - x_{desired}^T Q x_n \right)$$

Subject to $x_{n+1} = Ax_n + Bu_n$

$$x_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

$$x_{desired} = \begin{bmatrix} 3 & 0 & 3 & 0 & 0 & 0 \end{bmatrix}^T$$

$u_{min} \leq u_n \leq u_{max}$ Where, $u_{min} = \begin{bmatrix} -5 & -5 \end{bmatrix}^T$ and $u_{max} = \begin{bmatrix} 5 & 5 \end{bmatrix}^T$

$x_{min} \leq x_n \leq x_{max}$ Where, $x_{min} = \begin{bmatrix} 0 & -2 & 0 & -2 & 0 & 0 \end{bmatrix}^T$ and $x_{max} = \begin{bmatrix} 0 & 2 & 0 & 2 & 0 & 0 \end{bmatrix}^T$

The problem can be rewritten in matrix form as

$$\begin{aligned} & \min_y \frac{1}{2} y^T G y + g^T y \\ \text{Subject to} \quad & My = p \\ & Hy \leq h \end{aligned}$$

This optimal control problem is of the form

$$\min_x \quad \frac{1}{2} x^T P x + q^T x \quad \text{subject to} \quad Ax = b \quad Gx \leq h$$

And so, we can simply use the **cvxopt** solver from **qpsolvers** to find the optimal solution. Link to Jupyter Notebook - Exercise_1.ipynb. Please note that the notebook contains the solution to the exercise from the previous homework as well, so, one must scroll further down to find the solution to this problem.

Exercise: 2 We have seen in class two methods to minimize an arbitrary (twice continuously differentiable) function: gradient descent and Newton's method. Implement in a Jupyter Notebook both gradient descent and Newtons method using a backtracking line search where $c = 10^{-4}$. Use these algorithms to minimize the following functions:

- $-e^{-(x-1)^2}$, starting with $x_0 = 0$
- $(1-x)^2 + 100(y-x^2)^2$, starting with $x_0 = y_0 = 1.2$
- $x^T \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} x + \begin{bmatrix} -1 \\ 1 \end{bmatrix}^T x$, starting with $x_0 = \begin{pmatrix} 10 \\ 10 \end{pmatrix}$
- $\frac{1}{2} x^T \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 4 \end{bmatrix} x - \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}^T x$, starting with $x_0 = \begin{pmatrix} -10 \\ -10 \\ -10 \end{pmatrix}$

For each function:

- Show the convergence of each method by plotting (in the same plot) the norm of the distance to the optimum as a function of the number of iterations.
- Plot the value of α in the backtracking line search as a function of the number of iterations.
- Print the optimum and explain which criteria you used to stop the algorithms.

Do not invert the Hessian matrix, but instead use the `solve` function from NumPy (cf. Series 1) to find a step p_k that satisfies $\nabla^2 f(x_k) p_k = -\nabla f(x_k)$. Note also that when the Hessian is not positive definite, there will be an issue and you can add a small multiple of the identity to it until it is positive definite, i.e. use $\nabla^2 f(x_k) + \gamma I$ where γ is small instead of the Hessian.

Answer For each of the problems, common functions have been created in the Jupyter Notebook for code solution of Exercise 2. Here, we shall briefly dicsuss the functions and observe the plots in the Jupyter Notebook.

- **Gradient Descent without backtracking:** For this, we have used the simple formula of

$$x \leftarrow x - \alpha \nabla f(x) \tag{1}$$

As it is gradient descent, we shall use the first order necessary condition,

$$||\nabla f(x^*)|| = 0 \tag{2}$$

to test for convergence of the algorithm. Given below is an outline of the algorithm implemented in our function -

1. Initialize the starting point x_i with the initial guess x_0.
2. Store the initial point in a list to keep track of each step.
3. For each iteration (up to a maximum number of iterations):
   (a) Compute the gradient at the current point x_i.
   (b) Update x_i by moving in the opposite direction of the gradient, using a fixed step size α, *i.e.*, $x_{new} = x_i - \alpha \nabla f(x)$.
   (c) Add the new x_i value to the tracking list.
   (d) Check if the gradient's norm is below a tolerance level. If yes, stop and note the number of iterations taken.
4. If the gradient norm never falls below the tolerance within the maximum iterations, indicate non-convergence.
5. Calculate the distances of each x value in the history to the final computed optimum.
6. Print the optimum and plot the convergence graph showing distance to the computed optimum over iterations.

- **Gradient Descent with backtracking line search:** For this, not much changes, except that α is update at every iteration, following the Backtracking Line Search Algorithm -

$$\begin{aligned}\text{Choose } \bar{\alpha} > 0,\ \rho \in (0,1),\ c \in (0,1);\ \text{Set } \alpha \leftarrow \bar{\alpha};\\ \textbf{repeat until } f(x_k + \alpha p_k) \leq f(x_k) + c\alpha \nabla f_k^T p_k\\ \alpha \leftarrow \rho\alpha;\\ \textbf{end (repeat)}\\ \text{Terminate with } \alpha_k = \alpha.\end{aligned}$$

  Note that, $c = 10^{-4}$ has been given and $p_k = -\nabla f(x)$. The criteria for convergence remains the same. Given below is an outline of the algorithm implemented in our function -

  1. Initialize the starting point x_i with the initial guess x_0.
  2. Set the initial step size α and store it in a list to track step sizes.
  3. For each iteration (up to a maximum number of iterations):
     (a) Compute the gradient at the current point x_i.
     (b) Set the descent direction as the negative gradient.
     (c) Use backtracking line search to adjust α by reducing it until the function value satisfies the sufficient decrease condition.
     (d) Store the current α value for analysis.
     (e) Update x_i by moving in the direction of the descent using the current α, *i.e.*, $x_{new} = x_i + \alpha p_k$.
     (f) Add the new x_i value to the tracking list.
     (g) Check if the gradient's norm at the new x_i is below a tolerance level. If yes, stop and record the number of iterations taken.
  4. If the gradient norm never falls below the tolerance within the maximum iterations, indicate non-convergence.
  5. Calculate the distances of each x value in the history to the final computed optimum.
  6. Print the optimum, the function value at the optimum, and the final α value.
  7. Plot two graphs: (i) distance to the computed optimum over iterations and (ii) step size α over iterations.

- **Newton's method with backtracking line search:** For this method as well, we shall implement the same Backtracking Line search Algorithm with a minor change of

$$p_k = -\frac{\nabla f(x)}{\nabla^2 f(x)}$$

  Another thing that we have to check is whether $\nabla^2 f(x) \succ 0$, else we haven't found an appropriate descent direction.

  The criteria for convergence remains the same.

  Given below is an outline of the algorithm implemented in our function -

  1. Initialize the starting point x_i with the initial guess x_0.
  2. Set the initial step size α and prepare lists to track x values and α values.
  3. For each iteration (up to a maximum number of iterations):
     (a) Compute the gradient and Hessian at the current x_i.
     (b) Determine the descent direction p_k:
        i. If the Hessian is positive definite, solve p_k directly.
        ii. If the Hessian is not positive definite, apply regularization and solve p_k using the modified Hessian.
     (c) Use backtracking line search to adjust α by reducing it until the function value satisfies a decrease condition.
     (d) Store the current α value for analysis.
     (e) Update x_i by moving in the direction of p_k with step size α , *i.e.*, $x_{new} = x_i + \alpha p_k$.
     (f) Add the new x_i value to the tracking list.

(g) Check if the gradient's norm at the new x_i is below a tolerance level. If yes, stop and record the number of iterations taken.

4. If the gradient norm never falls below the tolerance within the maximum iterations, indicate non-convergence.
5. Calculate the distances of each x value in the history to the final computed optimum.
6. Print the optimum, the function value at the optimum, and the final α value.
7. Plot two graphs: (i) distance to the computed optimum over iterations and (ii) step size α over iterations.

An additional check that has been done in this algorithm is to find out whether we are dealing with scalars or matrices. If we are dealing with scalars, we can easily find $\nabla^2 f(x)^{-1}$. But in case we are dealing with matrices, we use numpy.linalg.solve to find p_k using the equation,

$$\nabla^2 f(x) \cdot p_k = -\nabla f(x)$$

Exercise: 3 Implementation of a SQP for nonlinear optimal control.

Answer Please follow the link to the Jupyter Notebook for code solution of Exercise 3. Discussed below, in brief, is the algorithm used for this exercise.

- **Question 1 Algorithm**
  1. Reformulate the cost to accurately track the trajectory of the pendulum as per the example given in Lecture 2 - LQ problems with KKT and QP.ipynb.
  2. Reformulate the equality constraints by finding the Taylor Expansion to linearize them about $\Delta\bar{x}$, such that we have $G(\bar{x})\Delta x = g(\bar{x})$ where $G(\bar{x})$ is the Jacobian of the equality constraint matrix.
  3. Form the Lagrangian of the optimal control problem in the form $\mathcal{L}(x, \lambda) = f(x) + \lambda^T g(x)$.
  4. Find the gradient of the cost function.
  5. Find the hessian of the Lagrangian while ignoring the second-order differntials of the constraints.
  6. Combine all the matrices found in the above steps to form the QP,

$$\min_p \frac{1}{2} p^T \nabla^2_{xx} \mathcal{L}(x_k) p + p^T \nabla f(x_k)$$
$$\text{subject to } \nabla g(x_k)^T p + g(x_k) = 0$$

     Which is the same as solving

$$\begin{bmatrix} \nabla^2_{xx} \mathcal{L}(x_k) & \nabla g(x_k)^T \\ \nabla g(x_k) & 0 \end{bmatrix} \begin{pmatrix} p_k \\ \lambda_{k+1} \end{pmatrix} = \begin{pmatrix} -\nabla f(x_k) \\ -g(x_k) \end{pmatrix}$$

  7. Compute the total cost violation.
  8. Start with a guess $\bar{x} = [0]$.
  9. Use line search with the Merit Function *(combination of cost and total constraint violation* to find an appropirate value for α. Here, p_k would be found by solving the KKT system formulated in Step 6.
  10. Update the guess using $\bar{x} = \bar{x} + \alpha p_k$.
  11. Check if the total constraint violation is less than a chosen tolerance of 10^{-4}. If yes, then the SQP has converged.
  12. Repeat Steps 9 to 11 until convergence is reached. If the max iterations have completed and the SQP convergence condition is not met, then more tuning is required or the problem formulation is incorrect.
  13. Animate and Plot the solution and the necessary parameters.
- **Question 2 Algorithm**
  1. Reformulate the cost to accurately track the trajectory of the pendulum as per the example given in Lecture 2 - LQ problems with KKT and QP.ipynb.
  2. Reformulate the equality constraints by finding the Taylor Expansion to linearize them about $\Delta\bar{x}$, such that we have $G(\bar{x})\Delta x = g(\bar{x})$ where $G(\bar{x})$ is the Jacobian of the equality constraint matrix.

3. Reformulate the inequality constraints to linearize them about $\Delta\bar{x}$, such that we have $H(\bar{x})\Delta x = h(\bar{x})$ where $H(\bar{x})$ is the Jacobian of the inequality constraint matrix.
4. Form the Lagrangian of the optimal control problem in the form $\mathcal{L}(x, \lambda) = f(x) + \lambda^T g(x)$.
5. Find the gradient of the cost function.
6. Find the hessian of the Lagrangian while ignoring the second-order differntials of the constraints.
7. Combine all the matrices found in the above steps to form the QP,

$$\begin{aligned} \min_{p} \quad & p^T \nabla^2_{xx} \mathcal{L}(x_k) p + p^T \nabla f(x_k) \\ \text{subject to} \quad & \nabla g(x_k)^T p + g(x_k) = 0 \\ & \nabla h(x_k)^T p + h(x_k) \leq 0 \end{aligned}$$

   Which is the same as solving

$$\begin{bmatrix} \nabla^2_{xx} \mathcal{L}(x_k) & \nabla g(x_k)^T & \nabla h(x_k)^T \\ \nabla g(x_k) & 0 & 0 \\ \nabla h(x_k) & 0 & 0 \end{bmatrix} \begin{pmatrix} p_k \\ \lambda_{k+1} \\ \mu_{k+1} \end{pmatrix} = \begin{pmatrix} -\nabla f(x_k) \\ -g(x_k) \\ -h(x_k) \end{pmatrix}$$

   Where $\nabla g(x_k)$ is the Jacobian of the equality constraints and $\nabla h(x_k)$ is the Jacobian of the inequality constraints. Please note that the signs on the right-hand side of the equation sometimes need to be modified, depending on how these matrices have been computed. Here, we shall use a QP solver instead of the **solve** function. We use the qpsolvers and use **cvxopt** as QP solver, following this example.
8. Compute the total cost violation, including both equality and inequality constraints
9. Start with a guess $\bar{x} = [0]$.
10. Use line search with the Merit Function *(combination of cost and total constraint violation* to find an appropirate value for α. Here, p_k would be found by solving the KKT system formulated in Step 6.
11. Update the guess using $\bar{x} = \bar{x} + \alpha p_k$.
12. Check if the total constraint violation is less than a chosen tolerance of 10^{-4}. If yes, then the SQP has converged.
13. Repeat Steps 9 to 11 until convergence is reached. If the max iterations have completed and the SQP convergence condition is not met, then more tuning is required or the problem formulation is incorrect.
14. Animate and Plot the solution and the necessary parameters.