Exercises 4 and 5: Linear, Nonlinear, and Robust MPC

Task 1: Linear MPC via Quadratic Programming

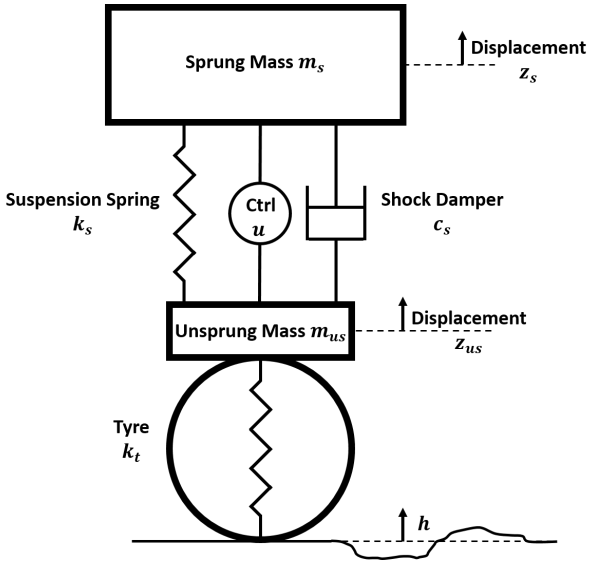

Figure 1: Active vibration damper for automotive suspensions. The vehicle, represented by the sprung mass m_s, is connected to the unsprung mass m_us, which is associated to the tyre and the axle, via a suspension spring (spring constant k_s) and a passive shock absorber (damping constant c_s). The vertical displacements of the masses w.r.t. their static equilibrium position for a perfectly plane road ($h = 0$) are described by z_us and z_s. To control the system, an actuator that is placed between the sprung and the unsprung mass is used to employ an external force as control input u. To account for vertical tyre deflections, the tyre is modeled as a spring with stiffness k_t. The unevenness of the road (h and its rate of change $\dot{h}$, respectively) acts as disturbance that oscillates the system.

The suspension is an essential part of the chassis of vehicles as it connects the vehicle to its wheels and enables relative motion between the two. The suspension consists of springs, shock absorbers, linkages and the tyres and is designed to dampen excitations of the vehicle through road unevenness. Therefore, it is important to have knowledge about the dynamics and the reaction of the suspension system as well as the vehicle body to road unevenness already in the design process. To this end, a suspension system is oftentimes represented by a *quarter-car model* (Fig. 1), which is a dynamical model of a single tyre's suspension and the associated fourth of the vehicle and which can be used to perform simulative studies. Via first principles modeling, the quarter-car model as shown in Fig. 1 is derived as

$$\dot{x} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & -1 \\ -\frac{k_\mathrm{s}}{m_\mathrm{s}} & -\frac{c_\mathrm{s}}{m_\mathrm{s}} & 0 & \frac{c_\mathrm{s}}{m_\mathrm{s}} \\ 0 & 0 & 0 & -1 \\ \frac{k_\mathrm{s}}{m_\mathrm{us}} & \frac{c_\mathrm{s}}{m_\mathrm{us}} & \frac{k_\mathrm{t}}{m_\mathrm{us}} & -\frac{c_\mathrm{s}}{m_\mathrm{us}} \end{bmatrix}}_{=:A} x + \underbrace{\begin{bmatrix} 0 \\ -\frac{1}{m_\mathrm{s}} \\ 0 \\ \frac{1}{m_\mathrm{us}} \end{bmatrix}}_{=:B} u + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}}_{=:V} \dot{h}, \tag{1}$$

where the state x is given by $x = \begin{bmatrix} z_\mathrm{s} - z_\mathrm{us} & \dot{z}_\mathrm{s} & h - z_\mathrm{us} & \dot{z}_\mathrm{us} \end{bmatrix}^\mathsf{T}$, u is the force employed by the actuator and $\dot{h}$ is the rate of change of the road profile (cf. Fig. 1).

The basic tasks of a vehicle's suspension system are

(i) to isolate the vehicle body from road disturbances to support ride comfort, i.e., to minimize the acceleration $\dot{x}_2 = \ddot{z}_\mathrm{s}$ of the sprung mass,

(ii) to support road holding and handling, which is achieved by minimizing the vertical tyre deflection $x_3 = h - z_\mathrm{us}$, and

(iii) to minimize the relative displacement $x_1 = z_\mathrm{s} - z_\mathrm{us}$ between the sprung and the unsprung mass.

It is important to note that (i) and (ii) are at odds with each other, such that a well-designed suspension systems makes a trade-off between ride comfort and vehicle handling. Those performance characteristics, $y = \begin{bmatrix} \ddot{z}_\mathrm{s} & h - z_\mathrm{us} & z_\mathrm{s} - z_\mathrm{us} \end{bmatrix}^\mathsf{T}$, can be computed from the states and the input according to

$$y = \underbrace{\begin{bmatrix} -\frac{k_\mathrm{s}}{m_\mathrm{s}} & -\frac{c_\mathrm{s}}{m_\mathrm{s}} & 0 & \frac{c_\mathrm{s}}{m_\mathrm{s}} \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{=:C} x + \underbrace{\begin{bmatrix} -\frac{1}{m_\mathrm{s}} \\ 0 \\ 0 \end{bmatrix}}_{=:D} u. \tag{2}$$

Suppose that the full system state is measured at discrete time points $t_k = kT_\mathrm{s},\ k \in \mathbb{N}$, where T_s is the sampling time, such that $x(t_k) = x_k$. Furhtermore, the input is chosen to be constant during each sampling period, i.e., $\forall t \in [t_k, t_{k+1}] : u(t) = u_k \in \mathcal{U} \subseteq \mathbb{R}$ and it is assumed that $\forall t \in [t_k, t_{k+1}] : \dot{h}(t) = \dot{h}_k$. Therewith, it is possible to derive a sampled-data (discretized) formulation of system (1), (2) as

$$x_{k+1} = \mathcal{A}x_k + \mathcal{B}u_k + \mathcal{V}\dot{h}_k \tag{3a}$$
$$y_k = Cx_k + Du_k \tag{3b}$$

Assuming the disturbance (the road profile) to be unknown a-priori, we can therefrom derive the *nominal* (i.e., in the absence of disturbances) discrete-time model

$$\hat{x}_{k+1} = \mathcal{A}x_k + \mathcal{B}u_k \tag{4a}$$
$$y_k = Cx_k + Du_k\ . \tag{4b}$$

For this task, you are provided with four Matlab files:

`Simulation.m`	This Matlab script is for computing the closed-loop simulation and for plotting the results. The uncontrolled case (passive vibration absorber) is already implemented, whereas the implementation of the controlled case has to be added during this exercise.
`SuspensionSystem.m`	This Matlab function implements the sampled-data dynamics (3a) (and (4a) respectively by setting $\dot{h}_k = 0$) by discretizing the continuous dynamics (1) using a Runge-Kutta method of fourth order.
`ControlledQuantities.m`	This Matlab function implements the computation of the performance characteristics (2) from a given state vector and input.
`Parameters_ActiveSystem.m`	This Matlab script provides the system parameters, the simulation time and the sampling time for Tasks 1 and 2.

Attention: There is a global variable called `system` defined in `Simulation.m`. This variable will later be used to change the system under consideration (Task 3 and 4), but is currently set to `'active'`, indicating that the active suspension system (Tasks 1 and 2) is considered. There is currently no other case implemented.

(a) Run the Matlab script `Simulation.m` to simulate the passive vibration absorber (autonomous system, i.e., $u = 0$). The results as well as the road profile and its rate of change are plotted.

To achieve a better performance of the vibration absorber, a discrete-time nominal model predictive controller, based on the nominal sampled-data model (4) (as we assume to not know the disturbance a-priori), with horizon N is to be used in this exercise, defined by the finite-horizon optimal control problem (OCP)

$$\min_{\hat{\boldsymbol{u}}_{\boldsymbol{k}}} \quad \left\{ J_{\mathrm{N}} = \sum_{i=0}^{N-1} \mathcal{L}\left(\hat{y}_{i|k}, \hat{u}_{i|k}\right) + E\left(\hat{y}_{N|k}\right) \right\} \tag{5a}$$

$$\text{s. t.} \quad \forall i \in \mathcal{I}_{0:\mathrm{N}-1} : \hat{x}_{i+1|k} = \mathcal{A}\hat{x}_{i|k} + \mathcal{B}\hat{u}_{i|k} \tag{5b}$$

$$\forall i \in \mathcal{I}_{0:\mathrm{N}} : \hat{y}_{i|k} = C\hat{x}_{i|k} + D\hat{u}_{i|k} \tag{5c}$$

$$\hat{x}_{0|k} = x_k \ . \tag{5d}$$

Therein, $\hat{\cdot}_{i|k}$ denotes the i-step ahead prediction within the horizon starting at time point k, $\mathcal{I}_{0:\mathrm{N}-1} := \{i \in \mathbb{N} \mid 0 \leq z \leq N-1\}$, and the possible system states $\hat{x}_{i|k}$ and outputs $\hat{y}_{i|k}$ are constrained to follow the sampled-data system dynamics (4) initialized with the measurement x_k. The optimization is performed over the control input sequence $\hat{\boldsymbol{u}}_{\boldsymbol{k}} = [\hat{u}_{0|k}, \ldots, \hat{u}_{N-1|k}]$.

(b) In `Parameters_ActiveSystem.m`, use the Matlab command `ss()` to construct a state space model of the time-continuous system (1), (2) without the disturbance term. Afterwards, discretize it for the sampling time T_{s} using `c2d()` to derive model (4).

Given the performance characteristics y of the suspension system and additionally considering the control effort, it is straightforward to derive the stage cost as

$$\mathcal{L}\left(\hat{y}_{i|k}, \hat{u}_{i|k}\right) = \hat{y}_{i|k}^{\mathsf{T}} \tilde{Q} \hat{y}_{i|k} + \hat{u}_{i|k}^{\mathsf{T}} \tilde{R} \hat{u}_{i|k} \ , \tag{6}$$

which penalizes deviations from the desired set-point $(x^{\mathsf{ref}}, u^{\mathsf{ref}}, y^{\mathsf{ref}}) = (\begin{bmatrix}0 & 0 & 0 & 0\end{bmatrix}^{\mathsf{T}}, 0, \begin{bmatrix}0 & 0 & 0\end{bmatrix}^{\mathsf{T}})$. Therein, the diagonal matrix $\tilde{Q} = \mathrm{diag}(\varrho_1, \varrho_2, \varrho_3)$ defines a weighting for the opposed performance tasks of the suspension system and $\tilde{R}$ is the weighting for the additional criterion of minimizing the control effort. As we will see later, it is beneficial to reformulate the stage cost (6) in terms of the states $\hat{x}_{i|k}$ using the output equation (4b), yielding

$$\mathcal{L}\left(\hat{x}_{i|k}, \hat{u}_{i|k}\right) = \hat{x}_{i|k}^{\mathsf{T}} \underbrace{\left(C^{\mathsf{T}} \tilde{Q} C\right)}_{=:Q} \hat{x}_{i|k} + \hat{x}_{i|k}^{\mathsf{T}} \underbrace{\left(C^{\mathsf{T}} \tilde{Q} D\right)}_{=:S} \hat{u}_{i|k} + \hat{u}_{i|k}^{\mathsf{T}} \underbrace{\left(D^{\mathsf{T}} \tilde{Q} C\right)}_{=:S^{\mathsf{T}}} \hat{x}_{i|k} + \hat{u}_{i|k}^{\mathsf{T}} \underbrace{\left(\tilde{R} + D^{\mathsf{T}} \tilde{Q} D\right)}_{=:R} \hat{u}_{i|k} \ . \tag{7}$$

(c) In `Parameters_ActiveSystem.m`, implement the matrices Q, S and R, using $\tilde{R} = 10^{-5}$ and $\tilde{Q} =$ diag$(\begin{bmatrix}1 & 5 \cdot 10^4 & 5 \cdot 10^3\end{bmatrix})$.

To design the terminal cost $E\left(\hat{y}_{N|k}\right)$, we will follow a strategy known as the *dual-mode prediction paradigm*. If the OCP (5) was tractable for infinite horizon $N \to \infty$, i.e., using the performance criterion

$$J_\infty = \sum_{i=0}^{\infty} \mathcal{L}\left(\hat{x}_{i|k}, \hat{u}_{i|k}\right) = \sum_{i=0}^{\infty} \hat{x}_{i|k}^{\mathsf{T}} Q \hat{x}_{i|k} + \hat{x}_{i|k}^{\mathsf{T}} S \hat{u}_{i|k} + \hat{u}_{i|k}^{\mathsf{T}} S^{\mathsf{T}} \hat{x}_{i|k} + \hat{u}_{i|k}^{\mathsf{T}} R \hat{u}_{i|k} \ , \tag{8}$$

solving this problem would result in an infinite control input sequence that would optimally steer the system to the origin and stabilize it there[1]. As the OCP is only tractable for finite horizon $N < \infty$, using the truncated

[1] In general, this requires the LQR assumptions to hold true, which can be easily checked to be the case for this example.

performance measure

$$J_{\mathrm{N}} = \sum_{i=0}^{N} \mathcal{L}\left(\hat{x}_{i|k}, \hat{u}_{i|k}\right) = \sum_{i=0}^{N} \hat{x}_{i|k}^{\mathsf{T}} Q \hat{x}_{i|k} + \hat{x}_{i|k}^{\mathsf{T}} S \hat{u}_{i|k} + \hat{u}_{i|k}^{\mathsf{T}} S^{\mathsf{T}} \hat{x}_{i|k} + \hat{u}_{i|k}^{\mathsf{T}} R \hat{u}_{i|k}$$

could (and will in general) introduce suboptimality to the solution. This is because the optimizer could choose a trajectory that is optimal for the next N steps but leaves the system afterwards in a state from which the desired steady state is reachable only at high cost. In other words, suboptimality is introduced as the optimizer is lacking knowledge about the *remaining infinite-horizon cost* after taking N steps. In the dual-mode prediction paradigm, the terminal cost is designed to model this remaining infinite-horizon cost for a terminal state $\hat{x}_{N|k}$. Such a terminal cost function is given by

$$E\left(\hat{x}_{N|k}\right) = \hat{x}_{N|k}^{\mathsf{T}} P \hat{x}_{N|k} \ , \tag{9}$$

where P is the solution of the algebraic Riccati equation

$$P = \mathcal{A}^{\mathsf{T}} P \mathcal{A} - (\mathcal{A}^{\mathsf{T}} P \mathcal{B} + S)(\mathcal{B}^{\mathsf{T}} P \mathcal{B} + R)^{-1}(\mathcal{B}^{\mathsf{T}} P \mathcal{A} + S^{\mathsf{T}}) + Q \tag{10}$$

associated with the infinite-horizon linear-quadratic regulator (LQR) problem for system (4) and infinite-horizon cost (8). Note that (9) is, contrary to the formulation in (5a), obtained as a function of the state. Furthermore, the dual-model prediction paradigm actually also requires a terminal controller (usually the associated LQR) and a terminal region (with specific properties). The complete idea is then to use the MPC to steer the system safely into this terminal region and to employ the terminal controller afterwards to keep the system in the terminal region and to asymptotically stabilize the origin. However, we will not use the entire scheme but only the idea of computeing and exploiting the remaining infinite horizon cost.

(d) In `Parameters_ActiveSystem.m`, compute matrix P by solving (10) using `lqr()` with the time-discrete system model and the matrices Q, S and R.

Using (7) and (9), the cost function in (5a) reads

$$J_{\mathrm{N}} = \sum_{i=0}^{N-1} \mathcal{L}\left(\hat{x}_{i|k}, \hat{u}_{i|k}\right) + E\left(\hat{x}_{N|k}\right) = \sum_{i=0}^{N-1} \hat{x}_{i|k}^{\mathsf{T}} Q \hat{x}_{i|k} + \hat{x}_{i|k}^{\mathsf{T}} S \hat{u}_{i|k} + \hat{u}_{i|k}^{\mathsf{T}} S^{\mathsf{T}} \hat{x}_{i|k} + \hat{u}_{i|k}^{\mathsf{T}} R \hat{u}_{i|k} + \hat{x}_{N|k}^{\mathsf{T}} P \hat{x}_{N|k} \ ,$$

which can be equivalently reformulated in a vectorized form,

$$J_{\mathrm{N}} = \hat{\boldsymbol{x}}_{\boldsymbol{k}}^{\mathsf{T}} \, L_{\mathrm{x}} \, \hat{\boldsymbol{x}}_{\boldsymbol{k}} + \hat{\boldsymbol{x}}_{\boldsymbol{k}}^{\mathsf{T}} \, L_{\mathrm{xu}} \, \hat{\boldsymbol{u}}_{\boldsymbol{k}} + \hat{\boldsymbol{u}}_{\boldsymbol{k}}^{\mathsf{T}} \, L_{\mathrm{xu}}^{\mathsf{T}} \, \hat{\boldsymbol{x}}_{\boldsymbol{k}} + \hat{\boldsymbol{u}}_{\boldsymbol{k}}^{\mathsf{T}} \, L_{\mathrm{u}} \, \hat{\boldsymbol{u}}_{\boldsymbol{k}} \ , \tag{11}$$

where

$$L_{\mathrm{x}} := \mathsf{blkdiag}(\overbrace{Q, \ldots, Q}^{N \text{ times}}, P) \ , \quad L_{\mathrm{u}} := \mathsf{blkdiag}(\overbrace{R, \ldots, R}^{N \text{ times}}) \ , \quad L_{\mathrm{xu}} := \begin{bmatrix} \mathsf{blkdiag}(\overbrace{S, \ldots, S}^{N \text{ times}}) \\ \mathbb{0} \end{bmatrix}$$

and $\mathsf{blkdiag}(\cdot)$ denotes a block-diagonal matrix. The output constraint (5c) can now be removed from the OCP formulation.

(e) In `Parameters_ActiveSystem.m`, implement the matrices L_{x}, L_{xu} and L_{u} and add them to the parameter structure.

We now observe that the cost function (11) is a quadratic in the input sequence $\hat{\boldsymbol{u}}_{\boldsymbol{k}}$ and the state sequence $\hat{\boldsymbol{x}}_{\boldsymbol{k}}$, whereupon the latter one is, via the dynamics constraints (5b), a function of the former one and the

known initial condition x_k. Exploiting the linearity of the model, we can write this functional dependency explicitly as

$$\begin{aligned}
\hat{x}_{0|k} &= x_k \\
\hat{x}_{1|k} &= \mathcal{A}x_k + \mathcal{B}\hat{u}_{0|k} \\
\hat{x}_{2|k} &= \mathcal{A}\hat{x}_{1|k} + \mathcal{B}\hat{u}_{1|k} = \mathcal{A}(\mathcal{A}x_k + \mathcal{B}\hat{u}_{0|k}) + \mathcal{B}\hat{u}_{1|k} \\
&= \mathcal{A}^2 x_k + \mathcal{A}\tilde{B}\hat{u}_{0|k} + \mathcal{B}\hat{u}_{1|k} \\
&\vdots \\
\hat{x}_{i|k} &= \mathcal{A}^i x_k + \mathcal{A}^{i-1}\mathcal{B}\hat{u}_{0|k} + \mathcal{A}^{i-2}\mathcal{B}\hat{u}_{1|k} + \ldots + \mathcal{B}\hat{u}_{i-1|k} \ ,
\end{aligned}$$

which can be equivalently expressed in matrix form as

$$\hat{\boldsymbol{x}}_{\boldsymbol{k}} = \underbrace{\begin{bmatrix} \mathbb{I} \\ \mathcal{A} \\ \mathcal{A}^2 \\ \vdots \\ \mathcal{A}^N \end{bmatrix}}_{=:\mathbf{M}} x_k + \underbrace{\begin{bmatrix} \mathbb{0} & \mathbb{0} & \ldots & \mathbb{0} & \mathbb{0} \\ \mathcal{B} & \mathbb{0} & \mathbb{0} & \ldots & \mathbb{0} \\ \mathcal{A}\mathcal{B} & \mathcal{B} & \mathbb{0} & \ldots & \mathbb{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathcal{A}^{N-1}\mathcal{B} & \mathcal{A}^{N-2}\mathcal{B} & \mathcal{A}^{N-3}\mathcal{B} & \ldots & \mathcal{B} \end{bmatrix}}_{=:W} \hat{\boldsymbol{u}}_{\boldsymbol{k}} \ . \tag{12}$$

By plugging (12) into (11), we can eliminate the dependency of the cost function on the state sequence, yielding

$$\begin{aligned}
J_{\mathrm{N}} = &\hat{\boldsymbol{u}}_{\boldsymbol{k}}^{\mathsf{T}} \left(L_u + W^{\mathsf{T}} L_{xu} + L_{xu}^{\mathsf{T}} W + W^{\mathsf{T}} L_x W \right) \hat{\boldsymbol{u}}_{\boldsymbol{k}} && \text{(13a)} \\
&+ 2x_k^{\mathsf{T}} M^{\mathsf{T}} \left(L_x W + L_{xu} \right) \hat{\boldsymbol{u}}_{\boldsymbol{k}} && \text{(13b)} \\
&+ x_k^{\mathsf{T}} M^{\mathsf{T}} L_x M x_k \ , && \text{(13c)}
\end{aligned}$$

which is now a function of only the known initial condition x_k and the optimization variable $\hat{\boldsymbol{u}}_{\boldsymbol{k}}$. Finally, this allows us to reformulate OCP (5) as an unconstrained *quadratic program* (QP) of the form

$$\min_{\hat{\boldsymbol{u}}_{\boldsymbol{k}}} \left\{ V_{\mathrm{QP}} = \frac{1}{2} \hat{\boldsymbol{u}}_{\boldsymbol{k}}^{\mathsf{T}} H \hat{\boldsymbol{u}}_{\boldsymbol{k}} + f^{\mathsf{T}} \hat{\boldsymbol{u}}_{\boldsymbol{k}} \right\} \ , \tag{14a}$$

where we obtain by comparison with (13) that

$$\begin{aligned}
\text{(13a)}: \quad & H = 2 \left(L_u + W^{\mathsf{T}} L_{xu} + L_{xu}^{\mathsf{T}} W + W^{\mathsf{T}} L_x W \right) && \text{(14b)} \\
\text{(13b)}: \quad & f^{\mathsf{T}} = 2x_k^{\mathsf{T}} M^{\mathsf{T}} \left(L_x W + L_{xu} \right) \ . && \text{(14c)}
\end{aligned}$$

Note that term (13c) is independent of the optimization variable and has thus no effect on the optimal solution $\hat{\boldsymbol{u}}_{\boldsymbol{k}}^*$, which is why it is neglected when reformulating the OCP as QP.

In summary, we have reformulated the constrained OCP (5) as an unconstrained QP (14) by exploiting the special problem structure (quadratic cost, linear system, linear constraints). For quadratic programs, there exist highly efficient solvers that exploit their special structure[2].

(f) In `Parameters_ActiveSystem.m`, precompute matrices H (numerical array/matrix) and f^{T} (function handle dependent on the initial condition x_k) and add them to the parameter structure. Attention: H should be actually symmetric but it is not due to numerical errors. Symmetrize H by replacing it with $\frac{1}{2}(H + H^{\mathsf{T}})$ to compensate for those numerical errors.

[2]If the original problem formulation contained further linear constraints (in vectorized form) $\mathcal{F}_x \hat{\boldsymbol{x}}_{\boldsymbol{k}} + \mathcal{F}_u \hat{\boldsymbol{u}}_{\boldsymbol{k}} \leq b_1$ and/or $\mathcal{G}_x \hat{\boldsymbol{x}}_{\boldsymbol{k}} + \mathcal{G}_u \hat{\boldsymbol{u}}_{\boldsymbol{k}} = b_2$, they would need to be reformulated in terms of only $\hat{\boldsymbol{u}}_{\boldsymbol{k}}$ by using (12) and to be imposed in the QP again.

(g) Implement a model predictive controller that solves in each time step the quadratic program (14) **given the initial condition** x_k**.**

- **(i) Define a new secondary function named** `Linear_MPC_quadprog` **that accepts the initial condition** x_k**, an initial guess of the optimal input sequence** $\hat{\boldsymbol{u}}_{\boldsymbol{k}}^*$ **and the parameter structure as inputs and returns the optimal input sequence for the current horizon.**
- **(ii) Compute matrix** f^T **of the QP's cost function by evaluating the precomputed function handle for the given initial condition** x_k**.**
- **(iii) Solve the QP** (14) **using Matlab's quadratic programming solver** `quadprog()`**. Pass the matrices** H **and** f^T **and the initial guess for the optimal input sequence. There are no further constraints.**

(h) In the simulation loop in `Simulation.m`**, add the code for simulating the controlled system.**

- **(i) Prepare an initial guess for the optimal input sequence. In the first iteration, use a vector of zeros. In any further iteration, use the previous optimal solution shifted by one time step. Duplicate the last entry to restore the correct length.**
- **(ii) Call** `Linear_MPC_quadprog` **with the current state, the prepared initial guess for the horizon's optimal input sequence and the parameter structure to compute the optimal input sequence. Record the computation time for this step in the variable** `cpu_time_quadprog`**.**
- **(iii) Extract the first element of the first input sequence and apply it as control input until the next sampling time point to the system.**

(i) Run the simulation.

Task 2: Linear MPC via Universal (Nonlinear Programming) Solvers

We consider again the set-up from Task 1, where we implemented a linear-quadratic model predictive controller (linear constraints, including the system dynamics, and quadratic cost function) for set-point regulation. In other words, we wanted to optimally steer the linear system (1), (2) to the origin and maintain (stabilize) it there despite unknown disturbances (road unevenness). To efficiently solve the optimal control problem (5), we have exploited the problem structure in two ways, namely

(i) by reformulating the quadratic cost function in vectorized form that already includes the linear dynamics constraints, i.e., by reformulating the problem as a quadratic program (QP), and

(ii) by choosing a highly specialized quadratic programming solver that exploits the specific structure and convexity property of a QP.

While quadratic cost functions are standard for set-point regulation and reference tracking problems, many real-world systems show nonlinear behavior or are subject to nonlinear constraints. In consequence, the OCP cannot be reformulated as a QP in many MPC applications and one has to fall back on less specialized, universal solvers that can handle nonlinearity and nonconvexity.

In this task, we now want to investigate the effect of using universal (nonlinear programming) solvers with the example from Task 1. To this end, we will first solve the QP using a nonlinear programming (NLP) solver that is not aware of the specific problem properties like convexity anymore. Afterwards, we will further resolve the QP-specific structure towards the standard MPC formulation again.

(a) Implement a model predictive controller that solves the quadratic program (14) **using a universal, nonlinear programming solver given the initial condition** x_k**.**