AWESCO Winter School on Numerical Optimal Control with DAE - University of Freiburg

Exercise 2: Nonlinear Programming and Root-finding Problems

Andrea Zanelli, Joel Andersson, Joris Gillis, Sebastien Gros, Moritz Diehl

Let us return to the constrained optimization problems from Exercise 1, Tasks 1.4 and 1.5:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & \underline{x} \leq x \leq \overline{x} \\ & \underline{g} \leq g(x) \leq \overline{g}, \end{aligned} \tag{1}$$

where $x \in \mathbb{R}^{n_x}$ is the decision variable as before, but with $f : \mathbb{R}^{n_x} \to \mathbb{R}$ and $g : \mathbb{R}^{n_x} \to \mathbb{R}^{n_g}$ being general twice continuously differentiable functions. This nonlinear program (NLP) can be solved with CasADi using a syntax very similar to the QP in Exercise 1, just calling the function `nlpsol` instead of `qpsol` and leaving the rest identical:

```
solver = nlpsol('solver','ipopt',prob);
```

Depending on the solver plugin used – here the open-source NLP solver IPOPT – CasADi will then automatically generate the required derivative information and pass it to the solver.

Tasks:

2.1 Modify the solution code from Exercise 1 to use IPOPT instead of qpOASES and solve Task 1.4. Since the problem is convex, the choice of initial guess should not matter.

2.2 Formulate and solve the following version of the famous Rosenbrock problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & x_1^2 + 100\, x_3^2 \\ \text{subject to} \quad & x_3 + (1 - x_1)^2 - x_2 = 0 \end{aligned}$$

Using $x = [2.5, 3.0, 0.75]$ as a starting point, how many iterations does the solver need to converge to the solution? Does it change if we instruct IPOPT to use a limited-memory BFGS approximation? This can be done by passing the following options dictionary as the forth argument to `nlpsol`:

```
opts = struct;
opts.ipopt.hessian_approximation = 'limited-memory';
```

2.3 Manually eliminate x_3 from the problem formulation using the constraint equation and resolve the now unconstrained problem with only two variables. How does the number of iterations change?

Nonlinear root-finding problems in CasADi

A special case of an NLP is a root-finding problem. We will write them in the form:

$$\begin{aligned} g_0(z, x_1, x_2, \ldots, x_n) &= 0 \\ g_1(z, x_1, x_2, \ldots, x_n) &= y_1 \\ g_2(z, x_1, x_2, \ldots, x_n) &= y_2 \\ &\vdots \\ g_m(z, x_1, x_2, \ldots, x_n) &= y_m, \end{aligned} \tag{2}$$

where the first equation uniquely defines z as a function of $x_1, \ldots, x_n$ by the *implicit function theorem* and the remaining equations define the auxiliary outputs $y_1, \ldots, y_m$. Given a function g for evaluating $g_0, \ldots, g_m$, we can use CasADi to automatically formulate a (differentiable) function $G : \{z_{\text{guess}}, x_1, x_2, \ldots, x_n\} \to \{z, y_1, y_2, \ldots, y_m\}$. This function includes a guess for z to handle the case when the solution is non-unique. The syntax for this, assuming $n = m = 1$, is:

```
z = SX.sym('x',nz);
x = SX.sym('x',nx);
g0 = (some expression of x and z)
g1 = (some expression of x and z)
g = Function('g', {z, x}, {g0, g1});
G = rootfinder('G', 'newton', g);
```

where the `rootfinder` function, similar to `nlpsol` and `qpsol`, expects a display name, the name of a solver plugin (here a simple full-step Newton method) and the problem formulation, here expressed as a residual function.

Tasks:

2.4 Starting with the unconstrained version of the Rosenbrock problem from Task 2.3, use CasADi's `gradient` function to get a new expression for the gradient of the objective function. According to the first order necessary conditions for optimality, this gradient must be zero. Formulate and solve this as a root-finding problem in CasADi. Use the same initial condition as before.