Exercise: Steady-state of a CSTR

In this exercise, we will be exploring a continuous stirred-tank reactor (CSTR). The reactor tank hosts two chemical substances A and B interacting dynamically as described by the following Modelica model:

```
model cstr
  // Parameters
  parameter Real V(unit="m3") = 1.0 "Volume of the reactor";
  parameter Real k(unit="m3/mol/s") = 0.5 "Reaction rate constant";

  // Inputs
  input Real q_in(unit="m3/s") "Inlet flow rate";
  input Real C_A_in(unit="mol/m3") "Inlet concentration of A";
  input Real C_B_in(unit="mol/m3") "Inlet concentration of B";

  // Variables
  Real C_A(unit="mol/m3", start = 0.5) "Concentration of A in the reactor";
  Real C_B(unit="mol/m3", start = 0.5) "Concentration of B in the reactor";

equation
  // Material balances for the reactor
  V*der(C_A) = q_in*(C_A_in - C_A) - V*k*C_A*C_B;
  V*der(C_B) = q_in*(C_B_in - C_B) + V*k*C_A*C_B;

end cstr;
```

Listing 1: `cstr.mo`: Modelica model of a CSTR

From Modelica to FMU

In this tutorial, we recommend to simply use the FMUs supplied by us. For details on how to create model-exchange FMUs adhering to FMI 2.0 or 3.0, consult your Modelica compiler documentation.
Trivial models like this one can also be created using the `compile_modelica` helper function:

Listing 2: Compiling `cstr.mo` from Python

```
from fmi_helpers import compile_modelica
compile_modelica('cstr', 'resources/cstr.mo', 'OpenModelica')
```

The mathematical structure of this dynamic model is that of a differential equation:

$$\dot{x} = f(x, u), \tag{1}$$

with the states $x = \left[C^A, C^B\right]^T$ corresponding to well-mixed substance concentrations, and with the controls $u = \left[C^A_{\text{in}}, C^B_{\text{in}}, q_{\text{in}}\right]^T$ corresponding to inflow concentrations and the inflow rate.

Finding a steady-state

Tasks:

1. Follow the CasADi install instructions from `https://web.casadi.org/get/#37`. Create a DaeBuilder object from the FMU as follows:

   ```
   import casadi as ca
   dae = ca.DaeBuilder('cstr', 'resources/cstr.fmu')
   dae.disp(True)
   ```

   Verify that the printed FMU model description starts with:

   ```
   nx = 2, nz = 0, nq = 0, ny = 0, np = 0, nc = 2, nd = 0, nw = 2, nu = 3
   Model variables:
     c = [V, k]
     x = [C_A, C_B]
     w = [der(C_A), der(C_B)]
     u = [C_A_in, C_B_in, q_in]
   ```

2. Next, build a mathematical function from it, in the form of a CasADi Function:

   ```
   f = dae.create('f', ['x', 'u'], ['ode'])
   print(f)
   ```

   As the printout reveals, the Function expects a 2-vector and a 3-vector as input arguments to evaluate numerically:

   ```
   f:(x[2],u[3])->(ode[2]) FmuFunction
   ```

   Verify, by numerical evaluation, that $f([1,2]^T, [3,4,5]^T)$ results in $[9,11]^T$.

   Evaluating CasADi functions

   CasADi function objects can be evaluated by either providing either a list of all arguments or by providing keyword arguments. The inputs are always (sparse) CasADi matrices, but if you provide a scalar or numpy array, it will be converted automatically. Importantly, CasADi functions can also be evaluated with symbolic arguments.
   A vector can be created by vertical concatenation: `ca.vertcat(a,b)`

3. Let us say $w = \left[C^A, q_{\text{in}}\right]^T$ are unknowns and that $C^B = 0.3$, $C^A_{\text{in}} = 1$, and $C^B_{\text{in}} = 0$ are fixed. We need to figure out a $w^\star$ that drives $\dot{x}$ to zero (= steady-state operation). We will use $w_0 = [1, 1]^T$ as an initial guess for $w^\star$.

   While this is essentially a root-finding problem, it is convenient to formulate this problem as a nonlinear program (NLP):

$$\begin{aligned} \min_{C^A,\, q_{\text{in}}} \quad & 0 \\ \text{subject to} \quad & f\Big(\left[C^A, 0.3\right]^T, [1, 0, q_{\text{in}}]^T\Big) = 0. \end{aligned}$$

   Using the `Opti` high-level environment within CasADi, we can find solve this NLP as follows:

```
opti = ca.Opti()              # Opti context

opti.minimize(0)              # Trivial objective (no degrees of freedom)
c_A  = opti.variable()        # Decision variables
q_in = opti.variable()

x = ca.vertcat(c_A, 0.3)      # State vector
u = ca.vertcat(1, 0, q_in)    # Control vector
xdot = f(x, u)                # Evaluate f symbolically
opti.subject_to(xdot == 0)    # Constrain state derivatives to be zero

opti.set_initial(c_A, 1)      # Set initial guesses
opti.set_initial(q_in, 1)

opti.solver('ipopt')          # Choose solver
sol = opti.solve()            # Optimize

c_A_opt = sol.value(c_A)      # Get solution
q_in_opt = sol.value(q_in)
```

   Verify that the you get the solution $w^\star = [0.7, 0.35]^T$. Play around with with different initial guesses and values of the fixed variables.

4. (Optional) Now let us switch back to the race car model from the demo. The corresponding FMU is called `kloeser2020.fmu`. Using the script from Task 3 as a template, find the particular motor input required to maintain a constant speed of $3\,\mathrm{m/s}$.

   Note that the controls of the model are rates:

```
for u in dae.u():
  print(u,":",dae.description(u))
```

```
D_der : Rate of electric motor duty cycle
delta_der : Rate of steering angle
```

   Hint: Fix all the states at their initial values except the throttle (`D`). The control inputs should be zero. Solve the problem of finding the throttle needed to maintain a constant speed, i.e. the time derivative of the velocity (`v`) should be zero.

   Verify that you obtain $D^\star = 0.80769$.