Feedback linearization using the Symbolic Math Toolbox

A primer on symbolic computations with MATLAB

With the Symbolic Math Toolbox, symbolic computations can be performed within MATLAB and the resulting expression can be used to define MATLAB functions. Albeit being less powerful than specialized softwares like MAPLE or Mathematica, this toolbox enables us to perform useful computations in the context of Control Theory and in particular feedback linearization. One of its main drawback is that the numerical evaluation of symbolic expression is very slow. Hence it is important to create a "standard" MATLAB function from a symbolic expression once we computed all the necessary expressions.

Some useful functions from the Symbolic Math Toolbox used in this lab session:

- `sym('x', [n, 1])`: creates a symbolic variable $x \in \mathbb{R}^{n\times 1}$. Its components can be accessed by their index, like for matrices: `x(1)` is the first component of the symbolic variable x.
- `syms u`: creates a scalar symbolic variable `u`.
- `f = [x(1); 42*u]`: defines a symbolic expression from the symbolic variable `x` and `u`.
- `diff(f, u)`: differentiates `f` with respect to the symbolic variable `u`.
- `symvar(f)`: return all the symbolic variables found in the symbolic expression `f`.
- `fnum = matlabFunction(f, 'vars', [x; u])`: return a numerical function `fnum` from the symbolic expression `f` with arguments `x` and `u`. Note that the resulting function `fnum` expects only scalar arguments; if in our example `x = [x_1, x_2]` and `u` is scalar, then `fnum` is called in the following way: `fnum(arg1, arg2, arg3)`, where `arg1` is the value of `x_1`, `arg2` the value of `x_2` and `arg3` the value of `u`.

More info are available online at `http://www.mathworks.fr/help/symbolic/index.html`.

The function used to compute the feedback linearization

The function `computeLinearization` is working for a particular subclass of dynamical systems: SISO nonlinear affine in control systems, which have the following state-space representation:

$$\begin{aligned}\dot{x} &= f_x(x) + g(x)u\\ y &= h(x),\end{aligned}$$

with $x \in \mathbb{R}^n$, $f_x, g\colon \mathbb{R}^n \to \mathbb{R}^n$, $y \in \mathbb{R}$, $h\colon \mathbb{R}^n \to \mathbb{R}$ and $u\colon \mathbb{R} \to \mathbb{R}$. The arguments of the `computeLinearization` function are the followings:

- `f`: the vector field as a symbolic expression of the state and control input.
- `h`: the symbolic expression for the function of the state defining the scalar output y.
- `x`: the symbolic variable for the state of the system.
- `u`: the symbolic variable for the control input.

The `computeLinearization` function return 3 MATLAB functions: a and b defined as $y^{(r)} = a + bu$, with r the relative degree. The last function is the diffeomorphism T for the change of variable $z = T(x)$. Be careful that it is fully defined only when $r = n$. If when the output is y the system has zero dynamics, then the given function has to be completed.

The algorithm behind the `computeLinearization` function

The basic idea is to compute the Lie derivative with respect to the vector field and to detect whether at the current iteration the control input is present in the expression. This is implemented by differentiating the Lie derivative with respect to u and checking whether the resulting expression is null or not. The algorithm can be represented in the following way.

Algorithm 1 Computation of a feedback linearization

Require: h the function defining the scalar output of the system
Require: f the vector field as symbolic expression
Require: n the dimension of the system
  $T_1 = h$ {Construction of the diffeomorphism}
  $i = 0$
  $L_f^i h = h$
  done = **false**
  while $i < n$ **and** done **not true do**
    $dh = \dfrac{\partial L_f^i}{\partial x}$ {Computation of the Jacobian}
    $L_f^i = dh \cdot f$
    if $\dfrac{\partial L_f^i}{\partial u} \not\equiv 0$ **then**
      done = **true**
    end if
    $T_{i+1}(x) = L_f^i$ {Construction of the diffeomorphism}
    $i = i + 1$
  end while
  $b = \dfrac{\partial L_f^i}{\partial u}$ $\{a = L_{f_x}^r h\}$
  $a = L_f^i - bu$ $\{b = L_g L_{f_x}^{r-1} h\}$
  $r = i$ {The relative degree}
  return T a diffeomorphism for the change of variable $z = T(x)$
  return two functions a, b with $y^{(r)} = a + bu$.