Hands-on CasADi course on optimal control, Yacoda

Exercise: Rootfinding

In this exercise, we'll be finding a root of a 2-dimensional residual function $g : \mathbb{R}^2 \mapsto \mathbb{R}^2$:

$$g\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} \tanh((x+2)y^2\frac{1}{25} - 0.5) \\ \sin(x) - 0.5y + 1 \end{bmatrix}. \tag{1}$$

1 Rootfinding, finite-differences

Tasks:

1. Write an `m-function` that implements g:

   ```
   function z=g(w)
     x = w(1); y = w(2);
     g1 = tanh((x+2)*y^2/25-0.5);
     g2 = sin(x)-0.5*y + 1;
     z = [g1;g2];
   end
   ```

   Verify that `g([-0.8;2])` yields `[-0.2986;-0.7174]`.

2. Write a script that computes the Jacobian of g using finite differences with $\epsilon =$ `1e-8`, at the point $x_0 = \begin{bmatrix} -0.8 \\ 2 \end{bmatrix}$. Verify that the Jacobian at this location is `[0.1457 0.1749; 0.6967 -0.5000]`.

3. Perform 5 Newton steps starting from x_0. Verify that you end up at `[0.1945;2.3866]`.

2 Rootfinding, using a CasADi Jacobian

For now, all you need to know about CasADi is:

1. Symbols are created as in `x = MX.sym('x')`.

2. Symbols, numbers and operations can be composed into symbolic expressions e.g. `sin(x)-0.5`.

3. Symbolic expressions can be evaluated by constructing a CasADi Function using a list of input symbols and list of output expressions:

   ```
   f = Function('f',{x},{sin(x)-0.5});
   f(0.8) % evaluate for x=0.8
   ```

4. The output of a CasADi Function evaluation is a CasADi numeric type (`DM`). Use 'full' or 'sparse' to convert it to a regular Matlab matrix:

```
f_value = full(f(0.8))
```

Tasks:

1. Follow the CasADi install instructions from `http://install35.casadi.org`. Verify that you can run the example on that webpage.

2. After importing CasADi into the global namespace (`import casadi.*`), create a two-by-one symbolic matrix `x` as follows:

```
X = MX.sym('X',2);
```

   Verify that `g(X)` evaluates without error. Use Matlab's `class(.)` function to check the datatype of $g(X)$, and `size(.)` to check its dimension: it is a 2-by-1 symbolic expression. Can you make sense of print-representation? Inspect the symbolic expression `J=jacobian(g(X),X)`. What are its datatype and dimensions?

3. Create a CasADi Function called `Jf` (see syntax in step 3 above) that maps from X to J.The print representation[1] of this Function will look like:

```
Jf:(i0[2])->(o0[2x2]) MXFunction
```

   Keeping in mind that a print representation should convey something insightful in a compact way, what do you think `[2]` and `[2x2]` mean here?

4. Now that we created a Function out of J, evaluate it numerically (see syntax in step 4 above) at $x_0 = \begin{bmatrix} -0.8 \\ 2 \end{bmatrix}$. Verify the result with the finite difference result obtained earlier.

5. Perform 5 Newton steps starting from x_0 and using a Jacobian computed by CasADi.

Here, we will pause the exercise and dive a bit deeper into CasADi basics. . .

---

[1] `disp(Jf)`

3 CasADi's rootfinder

Instead of writing out the Newton steps ourselves, we may also use a built-in rootfinder of CasADi. Mathematically, the expected form for the residual function is:

$$g(x, p) = 0, \tag{2}$$

with $x \in \mathbb{R}^n$ the unknowns, and $p \in \mathbb{R}^m$ parameters.
Syntax-wise, the construction of this CasADi Function looks like:

```
rf = rootfinder('rf','newton',struct('x',...,'p',...,'g',...));
```

where `'rf'` is a label, `'newton'` identifies a particular solver implementation, and the dots are placeholders for symbolic expressions. The `x` and `p` expressions should be symbols, while the `g` expression should depend on those (and only those) symbols. The `p` keyword and expression may also be omitted (indeed, here we have no parameters i.e. $m = 0$).

Tasks:

1. Create a CasADi rootfinder Function object `rf` that can be used to solve Equation 1. Use only expressions defined earlier in the exercise. Verify that the print representation is as follows:

   ```
   rf:(x0[2],p[])->(x[2]) Newton
   ```

   This means that the function expects 2 inputs:

   (a) The initial guess for the unknown `x`, a two-vector.

   (b) The parameter vector `p`, an empty vector `[]`.

2. Verify that the evaluation of the rootfinder Function object `rf` gives the same result as the hand-coded Newton iterations for the same initial guess:

   ```
   rf([-0.8;2],[])
   ```

3. Have a look at rf.stats(). How many iterations did the rootfinder take?

4. The rootfinder constructor takes an optional fourth argument: a structure of options.

   Try out what information you can get with a `'print_iteration'` option set to true.