TEMPO Spring School: Theory and Numerics for Nonlinear Model Predictive Control

Exercise 1: Nonlinear programming

J. Andersson M. Diehl J. Gillis J. Rawlings M. Zanon

University of Freiburg, March 26, 2015

Nonlinear programming in CasADi

CasADi can be used to solve parametric NLPs of the following form:

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

where $x \in \mathbb{R}^n$ is the decision variable and $p \in \mathbb{R}^m$ is a fixed (and known) parameter vector. Equality constraints are formulated by having upper and lower bound equal, i.e. $\underline{g}^{(k)} = \overline{g}^{(k)}$ for some k. In the following, p is absent.

In order to allocate an solver, we construct a CasADi function that takes x (and possibly a set of known parameters p) as inputs and returns f and g. This can be done with the syntax:

```
x = SX.sym("x",3)
f = ...
g = ...
nlp = SXFunction(nlpIn(x=x),nlpOut(f=f,g=g))
```

This function is than used to construct an NLP solver instance as follows:

```
solver = NlpSolver("ipopt",nlp)
solver.setOption("option_name", option_value)
solver.init()
```

where we use CasADi's interface to the open-source NLP solver IPOPT. From the symbolic expressions, the interface will then automatically generate the information that it might need to solve the NLP, which may be solver and option dependent. Typically, an NLP solver will need a function that gives the Jacobian of the constraint function and a Hessian of the Lagrangian function ($L(x, \lambda) = f(x) + \lambda^{\mathrm{T}} g(x)$) with respect to x.

NLP solvers are *functions* in CasADi that are "evaluated" to get the solution as outlined in Section 4.1 of the user guide, e.g.:

```
solver.setInput(x0,   "x0")          Initial guess for x
solver.setInput(lbx,  "lbx")         Lower bound on x
solver.setInput(ubx,  "ubx")         Upper bound on x
solver.setInput(lbg,  "lbg")         Lower bound on g(x)
solver.setInput(ubg,  "ubg")         Upper bound on g(x)
solver.evaluate()                    Solve the NLP
f_opt = solver.getOutput("f")        Get optimal cost
x_opt = solver.getOutput("x")        Get optimal solution
```

You will find the input and output schemes in the CasADi API documentation on the website or by using the question mark in Python.

```
NlpSolver?
```

Tasks:

1.1 Go to the CasADi website and locate the user guide. With a Python interpretor in front of you, quickly skim through Chapter 3 as well as Sections 4.1, 4.2 and 4.3 in Chapter 4.

1.2 Formulate and solve the Rosenbrock problem:

$$\begin{array}{ll} \underset{x\in\mathbb{R}^3}{\text{minimize}} & x_0^2 + 100\, x_2^2 \\ \text{subject to} & x_2 + (1 - x_0)^2 = x_1 \end{array} \tag{1}$$

Use $x_0 = 2.5$, $x_1 = 3.0$, $x_2 = 0.75$, as a starting point. How many iterations do you need to to converge using default options?

1.3 By default, IPOPT will use exact Hessian information. To avoid having to calculate second order information, we can instruct IPOPT to use a limited-memory BFGS approximation for the Hessian using the command:

```
solver.setOption("hessian_approximation","limited-memory")
```

How does this influence the number of iterations?

1.4 **Extra**: A function might have multiple local minima. Consider the function:

$$f(x) = \exp(-x_0^2 - x_1^2)\, \sin(4\,(x_0 + x_1 + x_0\, x_1^2)) \tag{2}$$

in the domain $[-1, 1] \times [-1, 1]$. You can visualize the function using the following lines in Python:

```
from numpy import *
from matplotlib import pylab as plt

# Domain
[X0,X1] = plt.meshgrid(linspace(-1,1,100),linspace(-1,1,100))

# Function
F = exp(-X0**2-X1**2)*sin(4*(X0+X1+X0*X1**2))

# Plot the function
plt.clf()
plt.contour(X0,X1,F)
plt.colorbar()
plt.jet()
plt.xlabel('x0')
plt.ylabel('x1')
plt.show()
```

Find the unconstrained minimizer of the function starting at different starting points, e.g. $[0, 0]$, $[0.9, 0.9]$, $[-0.9, -0.9]$. What do you see? To solve unconstrained problems with CasADi, simply leave out the second argument to `nlpOut`.