**INSTALLING EXTRA SOLVERS**

While python comes with some default solvers, for better performances it is recommended to install some other solvers:

**CPLEX**

Cplex is the most powerful linear solver, gains of times against GLPK are from a factor 10 to 100 (1000?) for larger problems.

**Install cplex:**

A library can be installed directly in python, however it might also require to have the full cplex installed (next steps) for larger problems. You can run in anaconda prompt:

conda install -n YOURPYTHONENVIRONMENT -c ibmdecisionoptimization cplex

In the WHATIF\_main.py or WHATIF\_scenario.py, choose the cplex solver:

solver = SolverFactory('cplex')

Cplex is free for academics and can be downloaded here (watch out to not get the free-edition, but the full edition, free for academics):

<https://www.ibm.com/products/ilog-cplex-optimization-studio>

Instruction to setup the python environment are here:

<https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.1/ilog.odms.cplex.help/CPLEX/GettingStarted/topics/set_up/Python_setup.html>

**Tip Using cplex on the DTUHPC cluster:**

There seems to be a little bug with cplex, it will always use as many cpu as physically available on a node, independently from how many you required, this is also a problem if you run scenarios in parallel. You can limit the number of threads cplex is using (N=max number of threads cplex is allowed to use):

solver = SolverFactory('cplex')

solver.options['threads']=N

**IPOPT**

Ipopt is the best open-source non-linear solver available with pyomo. You need to find an ipopt python package and install it, sometimes you also need to specify the path to the "executable" (which is ipopt.exe in windows, but no .exe in linux)

solver = SolverFactory('ipopt')

or

solver = SolverFactory ('ipopt',executable="~/miniconda3/pkgs/ipopt\_bin-3.7.1-10/bin/ipopt")

**Use ipopt with HSL**

(REM: these steps can be tricky!)

The default linear solver in ipopt, 'mumps', is not very performant. The HSL library is a linear solver you can combine to ipopt (making ipopt much faster than with mumps). You can get a precompiled hsl, that ipopt can load, however it seems hard to get it to work, compiling ipopt with hsl seems the way to go.

Main instructions: <https://www.coin-or.org/Ipopt/documentation/>

1 Download ipopt <https://www.coin-or.org/Ipopt/documentation/node12.html#SECTION00042300000000000000>

2 Install thirdparties

ASL : <https://www.coin-or.org/Ipopt/documentation/node13.html>

HSL: <https://www.coin-or.org/Ipopt/documentation/node13.html>

METIS: same link, seems necessary for HSL 77, 86 and 97

(get the source code from hsl and place it in the ThirdParty directory of the ipopt folder with the correct name)

3 Compile the code <https://www.coin-or.org/Ipopt/documentation/node14.html>

(these line work on the DTU HPC cluster)

configure --with-blas="-L/usr/lib64/atlas -lsatlas"

If installing HSL 77-97, instead run:

ADD\_CFLAGS=-fopenmp ADD\_FFLAGS=-fopenmp ADD\_CXXFLAGS=-fopenmp ./configure --with-blas="-L/usr/lib64/atlas -lsatlas"

make

make test

make install

you should get a bin directory with a ipopt file, if you correctly installed the ASL thirdparty

**Ipopt solver options**

To specify options, one solution is two add a file ipopt.opt with the required options (somewhere in the active path) – easier is to use pyomo's interface (not all options are accessible however) (see below)

Some description of the options can be found:

<https://www.gams.com/latest/docs/S_IPOPT.html>

<https://www.pserc.cornell.edu/matpower/docs/ref/matpower5.0/ipopt_options.html>

<https://list.coin-or.org/pipermail/ipopt/2012-February/002766.html>

if solver.name == 'ipopt':

#solver.options['linear\_solver']='mumps'

#solver.options['tol']=1e-02

#solver.options['dual\_inf\_tol']=10

#solver.options['bound\_relax\_factor']=0

solver.options['mu\_strategy']='adaptive'

#solver.options['max\_iter']=10000

#solver.options['dual\_inf\_tol'] = 0.1

#solver.options['warm\_start\_init\_point']='yes'

#solver.options['mehrotra\_algorithm ']='yes'

The parameter 'adaptive' for the 'mu\_strategy' option, seems to speed up the solving time – However it leads to instability when used in a non-linear MPC framework

Using a warm start, speeds up considerably if resolve the same (or similar?) problem, but might also create some troubles

'bound\_relax\_factor' instores a tolerance to violate the constraints, when not set to a low value (e.g. 10\*\*-12 while default is 10\*\*-8), can generate some problems in a non-linear MPC framework.

TEST OF IPOPT OPTIONS IN PARALLEL

solver = SolverFactory('ipopt') #Probably also need to indicate the path to executable

|  |  |
| --- | --- |
| Command |  |
| linear\_solver='mumps' | 'ma27', 'ma97' … choose linear solver |
| mu\_strategy='adaptive' | Seems to make all solver go significantly faster (x2 to x3): |
| bound\_relax\_factor=10\*\*-12 | In MPC seems to help to the constraint violation problem for all solvers. for mumps solver can be set to 0, for other solvers seems to considerably slow down + crashes |
| [linear\_system\_scaling](https://www.gams.com/latest/docs/S_IPOPT.html#IPOPTlinear_system_scaling)= | 'mc19' (which is supposed to be default, but does not seem to be) provides slight speed increase but crashes in MPC – few tests – this could be verified |
|  |  |