1.3 Running Pyomo on the Notre Dame CRC Cluster

Preliminaries

Before proceeding with this tutorial, you need to obtain a CRC account and install a private copy of Pyomo on the CRC cluster.

Request a CRC Account

First, you must register for a CRC account. These are free for all ND researchers.

Install Pyomo on CRC Cluster

Per CRC policies, standalone Python packages such as Pyomo are not centrally installed or maintained. Instead, users must install a private copy of standalone Python packages on the CRC cluster. This can be easily done in a few steps:

First, `ssh` into a CRC interactive node. For example,

```
ssh crcfe02.crc.nd.edu
```

Next (optional), may load your preferred version of Python. If you do not, the system default (2.7) will be used. Pyomo supports both 2.7 and 3.x:

```
module load python/3.6.4
```

Then, install Pyomo using `pip`, a popular package manager for Python:

```
pip install --user pyomo
```

By default, Pyomo will be installed in `.local/bin`. See the ND CRC wiki instructions for adding `.local/bin` to your PATH.

Running on the CRC

There are two ways to run Pyomo on the CRC cluster: on an interactive mode (for testing only) or by submitting a job to the queue. The following shows how to do this for a simple optimization problem.

Test Problem

We will consider the following simple linear program to test Pyomo on the CRC:

```
# Load Pyomo
from pyomo.environ import *

# Create model
m = ConcreteModel()
m.x = Var([1,2,3], domain=NonNegativeReals)
m.c1 = Constraint(expr = m.x[1] + m.x[2] + m.x[3] <= 1)
m.c2 = Constraint(expr = m.x[1] + 2*m.x[2] >= 0.3)
m.OBJ = Objective(expr = m.x[3], sense=maximize)

# Specify solver
solver=SolverFactory('ipopt')

# Solve model
solver.solve(m, tee=True)
```

Copy this code and save it on your CRC AFS space as `pyomo_test.py`.

Running From the Command Line

The easest option is to directly run this code on an interactive node on the CRC cluster.

1. Login to the CRC cluster:

```
ssh crcfe02.crc.nd.edu
```

1. Load your preferred version of python and your favorite optimization solver.

```
module load ipopt
```

1. Run the Python script

```
python pyomo_test.py
```

You should see the following output:

```
Ipopt 3.12.8:

******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

This is Ipopt version 3.12.8, running with linear solver ma27.

Number of nonzeros in equality constraint Jacobian...:        0
Number of nonzeros in inequality constraint Jacobian.:        5
Number of nonzeros in Lagrangian Hessian.............:        0

Total number of variables............................:        3
                     variables with only lower bounds:        3
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints                .:        0
```

```
Total number of equality constraints.................:        0
Total number of inequality constraints...............:        2
        inequality constraints with only lower bounds:        1
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        1

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 -9.9999900e-03 2.70e-01 6.00e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1 -2.3096613e-02 1.48e-01 3.42e+00  -1.7 1.87e-01    -  1.10e-01 4.70e-01h  1
   2 -7.1865937e-02 0.00e+00 8.39e-01  -1.7 1.42e-01    -  4.87e-01 1.00e+00f  1
   3 -8.1191843e-01 0.00e+00 4.70e-01  -1.7 1.47e+00    -  1.00e+00 5.02e-01f  1
   4 -7.9286713e-01 0.00e+00 2.00e-07  -1.7 1.91e-02    -  1.00e+00 1.00e+00f  1
   5 -8.4402296e-01 0.00e+00 3.48e-03  -3.8 5.61e-02    -  9.40e-01 9.12e-01f  1
   6 -8.4949514e-01 0.00e+00 1.50e-09  -3.8 7.04e-03    -  1.00e+00 1.00e+00f  1
   7 -8.4999437e-01 0.00e+00 1.84e-11  -5.7 4.99e-04    -  1.00e+00 1.00e+00f  1
   8 -8.5000001e-01 0.00e+00 2.51e-14  -8.6 5.65e-06    -  1.00e+00 1.00e+00f  1

Number of Iterations....: 8

                                   (scaled)                 (unscaled)
Objective...............:  -8.5000001246787615e-01   -8.5000001246787615e-01
Dual infeasibility......:   2.5143903682766679e-14    2.5143903682766679e-14
Constraint violation....:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.........:   2.5342284749259295e-09    2.5342284749259295e-09
Overall NLP error.......:   2.5342284749259295e-09    2.5342284749259295e-09


Number of objective function evaluations             = 9
Number of objective gradient evaluations             = 9
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 9
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 9
Number of Lagrangian Hessian evaluations             = 8
Total CPU secs in IPOPT (w/o function evaluations)   =      0.004
Total CPU secs in NLP function evaluations           =      0.000

EXIT: Optimal Solution Found.
```

Submitting to a Queue

Per CRC policies, the interactive nodes should only be used for testing short computational jobs. All other jobs should be submitted to a queue.

To submit the previous project on the CRC queue, write a script that contains the following:

```
#!/bin/csh

#$ -M netid@nd.edu     # Email address for job notification
#$ -m abe                    # Send mail when job aborts, begins, and ends
#$ -q long                  # Specify queue
#$ -N job_name      # Specify job name

module load python/3.6.4     # Required modules
module load ipopt

python3 pyomo_test.py        # Command to execute
```

Name it something logical, such as `submission_script`. Then type:

```
qsub submission_script
```

You should get an email when your job begins/ends/aborts. You can also monitor its process through

```
qstat –u username
```

More info is available at the ND CRC wiki Quick Start Guide.

Available Solvers

The CRC clusters supports several large-scale optimization solvers that are directly callable from Pyomo. This makes it increadily easy - often one only needs to change a few lines of code - to try different solvers for an optimization problem.

The following contains instructions on using solvers currently available on the CRC cluster. By modifying the `SolverFactory` in `pyomo_test.py`, you can try all of these solvers on the simple test problem.

CPLEX

First load the module:

```
module load cplex
```

This will update your environmental variables and the Gurobi executable should be callable from Pyomo:

```
solver=SolverFactory('cplex')
```

Here is the output for the test problem:

```
Welcome to IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer 12.7.1.0
  with Simplex, Mixed Integer & Barrier Optimizers
5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21
Copyright IBM Corp. 1988, 2017.  All Rights Reserved.

Type 'help' for a list of available commands.
Type 'help' followed by a command name for more
information on commands.

CPLEX> Logfile 'cplex.log' closed.
Logfile '/afs/crc.nd.edu/user/a/adowling/Private/tmp103f3a.cplex.log' open.
CPLEX> Problem '/afs/crc.nd.edu/user/a/adowling/Private/tmpdhMJXY.pyomo.lp' read
.
Read time = 0.01 sec. (0.00 ticks)
CPLEX> Problem name         :
/afs/crc.nd.edu/user/a/adowling/Private/tmpdhMJXY.pyomo.lp
Objective sense      : Maximize
Variables            :       4
Objective nonzeros   :       1
Linear constraints   :       3  [Less: 1,  Greater: 1,  Equal: 1]
  Nonzeros           :       6
  RHS nonzeros       :       3

Variables            : Min LB: 0.000000         Max UB: all infinite
Objective nonzeros   : Min   : 1.000000         Max   : 1.000000
Linear constraints   :
  Nonzeros           : Min   : 1.000000         Max   : 2.000000
  RHS nonzeros       : Min   : 0.3000000        Max   : 1.000000
CPLEX> CPXPARAM_Read_APIEncoding                         "UTF-8"
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 2 rows, 3 columns, and 5 nonzeros.
Presolve time = 0.00 sec. (0.00 ticks)
Initializing dual steep norms . . .

Iteration log . . .
Iteration:     1   Dual objective     =             0.850000

Dual simplex - Optimal:  Objective =  8.5000000000e-01
Solution time =    0.00 sec.  Iterations = 1 (0)
Deterministic time = 0.00 ticks  (4.51 ticks/sec)

CPLEX> Solution written to file
'/afs/crc.nd.edu/user/a/adowling/Private/tmp0ryhXG.cplex.sol'.
```

Gurobi

First load the module:

```
module load gurobi
```

This will update your environmental variables and the Gurobi executable should be callable from Pyomo:

```
solver=SolverFactory('gurobi')
```

Here is the output for the test problem:

```
Optimize a model with 3 rows, 4 columns and 6 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [3e-01, 1e+00]
Presolve removed 1 rows and 1 columns
Presolve time: 0.02s
Presolved: 2 rows, 3 columns, 5 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.0000000e+00   1.500000e-01   0.000000e+00      0s
       1    8.5000000e-01   0.000000e+00   0.000000e+00      0s

Solved in 1 iterations and 0.02 seconds
Optimal objective  8.500000000e-01
Freed default Gurobi environment
```

Ipopt

Two versions (modules) of Ipopt are available to CRC users: `ipopt/3.12.8` and `ipopt/hsl/3.12.8`. The latter supports HSL linear algebra routines whereas the former does not (and relies on the open-source library MUMPS). In general, the HSL linear algebra routines are more stable and are significantly faster for large-scale problems. All IPOPT users are strongly encouraged to use the HSL libraries. To obtain access to `ipopt/hsl/3.12.8`, ND users must apply for a free HSL academic license and then forward the approval email to CRCSupport@nd.edu.

To use either version of Ipopt, first load the module. For example,

```
module load ipopt/hsl/3.12.8
```

This will update your environmental variables and the ipopt executable should be callable from Pyomo:

```
solver=SolverFactory('ipopt')
```

If needed, you can also directly specify the path to the Ipopt executable to Pyomo:

```
solver=SolverFactory('ipopt',
executable="/afs/crc.nd.edu/x86_64_linux/i/ipopt/3.12.8-hsl/bin/ipopt")
```

Next, you may want to customize some of the Ipopt options. For example, specify the linear algebra routine (use MA57 from the HSL library):

```
solver.options['linear_solver'] = "ma57"
```

Here is the output when using MA57:

```
Ipopt 3.12.8: linear_solver=ma57


******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

This is Ipopt version 3.12.8, running with linear solver ma57.

Number of nonzeros in equality constraint Jacobian...:        0
Number of nonzeros in inequality constraint Jacobian.:        5
Number of nonzeros in Lagrangian Hessian.............:        0

Total number of variables............................:        3
                     variables with only lower bounds:        3
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:        0
Total number of inequality constraints...............:        2
        inequality constraints with only lower bounds:        1
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        1

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 -9.9999900e-03 2.70e-01 6.00e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1 -2.3096613e-02 1.48e-01 3.42e+00  -1.7 1.87e-01    -  1.10e-01 4.70e-01h  1
   2 -7.1865937e-02 0.00e+00 8.39e-01  -1.7 1.42e-01    -  4.87e-01 1.00e+00f  1
   3 -8.1191843e-01 0.00e+00 4.70e-01  -1.7 1.47e+00    -  1.00e+00 5.02e-01f  1
   4 -7.9286713e-01 0.00e+00 2.00e-07  -1.7 1.91e-02    -  1.00e+00 1.00e+00f  1
   5 -8.4402296e-01 0.00e+00 3.48e-03  -3.8 5.61e-02    -  9.40e-01 9.12e-01f  1
   6 -8.4949514e-01 0.00e+00 1.50e-09  -3.8 7.04e-03    -  1.00e+00 1.00e+00f  1
   7 -8.4999437e-01 0.00e+00 1.84e-11  -5.7 4.99e-04    -  1.00e+00 1.00e+00f  1
   8 -8.5000001e-01 0.00e+00 2.51e-14  -8.6 5.65e-06    -  1.00e+00 1.00e+00f  1

Number of Iterations....: 8

                                   (scaled)                 (unscaled)
Objective...............:  -8.5000001246787615e-01   -8.5000001246787615e-01
Dual infeasibility......:   2.5143903682766679e-14    2.5143903682766679e-14
Constraint violation....:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.........:   2.5342284749259295e-09    2.5342284749259295e-09
Overall NLP error.......:   2.5342284749259295e-09    2.5342284749259295e-09


Number of objective function evaluations             = 9
Number of objective gradient evaluations             = 9
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 9
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 9
Number of Lagrangian Hessian evaluations             = 8
Total CPU secs in IPOPT (w/o function evaluations)   =      0.004
Total CPU secs in NLP function evaluations           =      0.000

EXIT: Optimal Solution Found.
```

Coin-OR

Several solvers are availabe in the COIN-OR module.

The first step to using any of these solvers is loading the module:

```
module load coin-or
```

Clp

The Clp solver executable should now be callable from Pyomo:

```
solver=SolverFactory('clp')
```

If needed, you can explicitly specify the path to the executable:

```
solver=SolverFactory('clp', executable="/afs/crc.nd.edu/x86_64_linux/c/coin-or/clp/1.16.11/bin/clp")
```

Output for the test problem:

```
Coin LP version 1.16.11, build Jun  4 2018
command line - /afs/crc.nd.edu/x86_64_linux/c/coin-or/clp/1.16.11/bin/clp
/afs/crc.nd.edu/user/a/adowling/Private/tmpEsGwQH.pyomo.nl -AMPL
```

Cbc

The Cbc solver executable should now be callable from Pyomo:

```
solver=SolverFactory('cbc')
```

If needed, you can explicitly specify the path to the executable:

```
solver=SolverFactory('cbc', executable="/afs/crc.nd.edu/x86_64_linux/c/coin-or/c
bc/2.9.9/bin/cbc")
```

Output for the test problem:

```
Welcome to the CBC MILP Solver
Version: 2.9.6
Build Date: Jun  3 2018

command line - /afs/crc.nd.edu/x86_64_linux/c/coin-or/cbc/2.9.9/bin/cbc -
printingOptions all -import
/afs/crc.nd.edu/user/a/adowling/Private/tmp5jMyQq.pyomo.lp -stat=1 -solve -solu
/afs/crc.nd.edu/user/a/adowling/Private/tmp5jMyQq.pyomo.soln (default strategy
1)
Option for printingOptions changed from normal to all
 CoinLpIO::readLp(): Maximization problem reformulated as minimization
Presolve 2 (-1) rows, 3 (-1) columns and 5 (-1) elements
Statistics for presolved model


Problem has 2 rows, 3 columns (1 with objective) and 5 elements
There are 1 singletons with objective
Column breakdown:
3 of type 0.0->inf, 0 of type 0.0->up, 0 of type lo->inf,
0 of type lo->up, 0 of type free, 0 of type fixed,
0 of type -inf->0.0, 0 of type -inf->up, 0 of type 0.0->1.0
Row breakdown:
0 of type E 0.0, 1 of type E 1.0, 0 of type E -1.0,
0 of type E other, 0 of type G 0.0, 0 of type G 1.0,
1 of type G other, 0 of type L 0.0, 0 of type L 1.0,
0 of type L other, 0 of type Range 0.0->1.0, 0 of type Range other,
0 of type Free
Presolve 2 (-1) rows, 3 (-1) columns and 5 (-1) elements
0  Obj 0 Primal inf 1.299998 (2) Dual inf 0.999999 (1)
2  Obj -0.85
Optimal - objective value -0.85
After Postsolve, objective -0.85, infeasibilities - dual 0 (0), primal 0 (0)
Optimal objective -0.85 - 2 iterations time 0.002, Presolve 0.00
Total time (CPU seconds):       0.00   (Wallclock seconds):       0.01
```

Bonmin

Warning: Bonmin has been compiled using the HSL linear algebra routines. All users that have access to Ipopt with HSL also have access to Bonmin.

After you submit your HSL license approval email to CRCsupport@nd.edu, the Bonmin solver executable will be callable from Pyomo:

```
solver=SolverFactory('bonmin')
```

If needed, you can explicitly specify the path to the executable:

```
solver=SolverFactory('bonmin', executable="/afs/crc.nd.edu/x86_64_linux/c/coin-o
r/bonmin/1.8.6/bin/bonmin")
```

Output for the test problem:

```
Bonmin 1.8.6 using Cbc 2.9.9 and Ipopt 3.12.8
bonmin:
Cbc3007W No integer variables - nothing to do

******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

NLP0012I
              Num      Status      Obj             It       time
ocation
NLP0014I             1         OPT -0.85000002        6 0.00295
Cbc3007W No integer variables - nothing to do

    "Finished"
```

Couenne

Warning: Couenne has been compiled using the HSL linear algebra routines. All users that have access to Ipopt with HSL also have access to Couenne.

After you submit your HSL license approval email to CRCsupport@nd.edu, the Couenne solver executable will be callable from Pyomo:

```
solver=SolverFactory('couenne')
```

If needed, you can explicitly specify the path to the executable:

```
solver=SolverFactory('couenne', executable="/afs/crc.nd.edu/x86_64_linux/c/coin-
or/couenne/0.5.6/bin/couenne")
```

Output for the test problem:

```
Couenne 0.5.6 -- an Open-Source solver for Mixed Integer Nonlinear Optimization
Mailing list: couenne@list.coin-or.org
Instructions: http://www.coin-or.org/Couenne
couenne:
ANALYSIS TEST: Couenne: new cutoff value -8.5000000000e-01 (0.045811 seconds)
NLP0012I
              Num      Status      Obj             It       time
ocation
NLP0014I             1         OPT -0.85000002        9 0.004882
Loaded instance "/afs/crc.nd.edu/user/a/adowling/Private/tmps3yXIf.pyomo.nl"
Constraints:            2
Variables:              3 (0 integer)
Auxiliaries:            1 (0 integer)

Coin0506I Presolve 2 (-1) rows, 3 (-1) columns and 5 (-2) elements
Clp0006I 0  Obj -0.849815 Primal inf 0.00018408442 (1) Dual inf 0.999999 (1)
Clp0006I 2  Obj -0.85
Clp0000I Optimal - objective value -0.85
Clp0032I Optimal objective -0.85 - 2 iterations time 0.002, Presolve 0.00
Cbc3007W No integer variables - nothing to do
Clp0000I Optimal - objective value -0.85

    "Finished"

Linearization cuts added at root node:          3
Linearization cuts added in total:              3  (separation time: 6.5e-05s)
Total solve time:                        0.001362s (0.001362s in branch-and-boun
)
Lower bound:                                 -0.85
Upper bound:                                 -0.85  (gap: 0.00%)
Branch-and-bound nodes:                          0
Performance of                            FBBT:      2.6e-05s,        3 runs. fix
0 shrnk:    10.7635 ubd:   0.666667 2ubd:          0 infeas:          0
```

SCIP

Coming soon.

In [ ]:

```
```