Sandia National Laboratories

Exceptional service in the national interest

5. Nonlinear Problems

PYOMO

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

CCR Center for Computing Research

Nonlinear problems are easy…

Nonlinear problems are easy…

… to write in Pyomo (correct formulation and solution is another story)

Nonlinear problems are easy…

… to write in Pyomo (correct formulation and solution is another story)

- Agenda:
  - Introduction
  - Rosenbrock Example
  - Introduction to Scripting
  - Introduction to IPOPT
  - Recommendations for Nonlinear Problems
  - Formulation Matters Example
  - Exercises

Nonlinear: Supported expressions

Operation	Operator	Example
multiplication	`*`	`expr = model.x * model.y`
division	`/`	`expr = model.x / model.y`
exponentiation	`**`	`expr = (model.x+2.0)**model.y`
in-place multiplication[1]	`*=`	`expr *= model.x`
in-place division[2]	`/=`	`expr /= model.x`
in-place exponentiation[3]	`**=`	`expr **= model.x`

```
model = ConcreteModel()
model.r = Var()
model.h = Var()

def surf_area_obj_rule(m):
    return 2 * pi * m.r * (m.r + m.h)
model.surf_area_obj = Objective(rule=surf_area_obj_rule)

def vol_con_rule(m):
    return pi * m.h * m.r**2 == 355
model.vol_con = Constraint(rule=vol_con_rule)
```

Nonlinear: Supported expressions

Operation	Function	Example
arccosine	`acos`	`expr = acos(model.x)`
hyperbolic arccosine	`acosh`	`expr = acosh(model.x)`
arcsine	`asin`	`expr = asin(model.x)`
hyperbolic arcsine	`asinh`	`expr = asinh(model.x)`
arctangent	`atan`	`expr = atan(model.x)`
hyperbolic arctangent	`atanh`	`expr = atanh(model.x)`
cosine	`cos`	`expr = cos(model.x)`
hyperbolic cosine	`cosh`	`expr = cosh(model.x)`
exponential	`exp`	`expr = exp(model.x)`
natural log	`log`	`expr = log(model.x)`
log base 10	`log10`	`expr = log10(model.x)`
sine	`sin`	`expr = sin(model.x)`
square root	`sqrt`	`expr = sqrt(model.x)`
hyperbolic sine	`sinh`	`expr = sinh(model.x)`
tangent	`tan`	`expr = tan(model.x)`
hyperbolic tangent	`tanh`	`expr = tanh(model.x)`

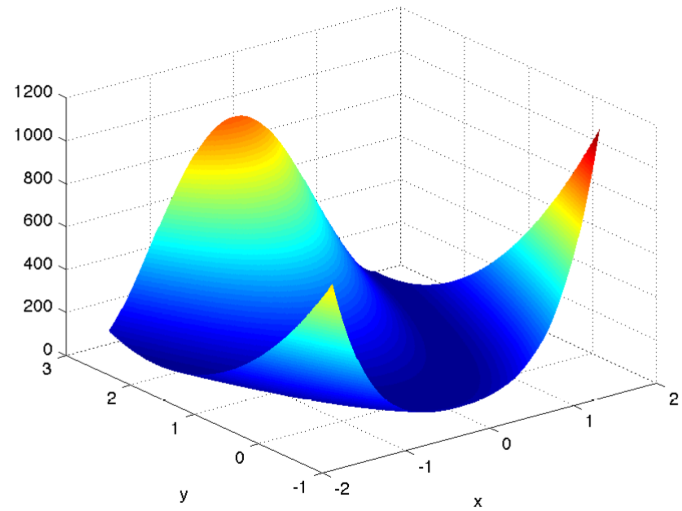
Caution: Always use the intrinsic functions that are part of the Pyomo package.

```
from pyomo.environ import * # imports, e.g., pyomo versions of exp, log, etc.)
from math import *                  # overrides the pyomo versions with math versions
```

Example: Rosenbrock function

$$\min_{x,y} \; f(x,y) = (1-x)^2 + 100\left(y - x^2\right)^2$$

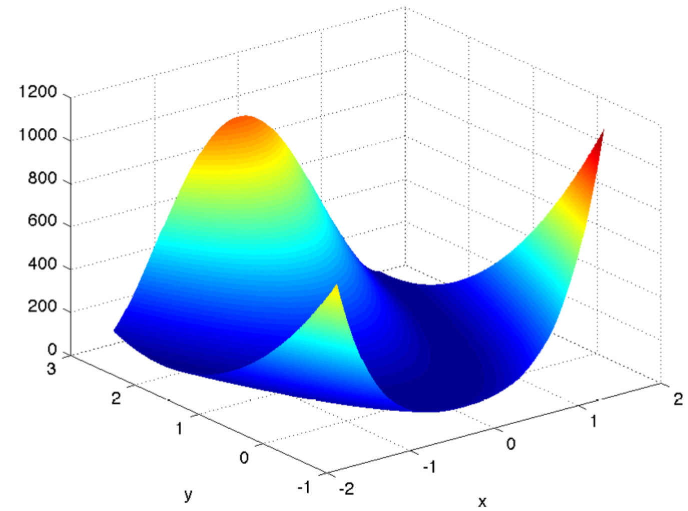
- Minimize the rosenbrock function using Pyomo and IPOPT
- Initialize at x=1.5, y=1.5



Example: Rosenbrock function

$$\min_{x,y} \; f(x,y) = (1-x)^2 + 100\left(y - x^2\right)^2$$

- Minimize the rosenbrock function using Pyomo and IPOPT
- Initialize at x=1.5, y=1.5




```
# rosenbrock.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)
```

Example: Rosenbrock function

```
# rosenbrock.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)
```

```
pyomo solve --solver=ipopt --summary --stream-solver rosenbrock.py
```

```
…
Variables:
  x : Size=1, Index=None, Domain=Reals
      Key  : Lower : Value : Upper : Fixed : Stale
      None :  None :   1.0 :  None : False : False
  y : Size=1, Index=None, Domain=Reals
      Key  : Lower : Value : Upper : Fixed : Stale
      None :  None :   1.0 :  None : False : False
…
```

Example: Rosenbrock function

```
…
Variables:
  x : Size=1, Index=None, Domain=Reals
      Key  : Lower : Value : Upper : Fixed : Stale
      None :  None :   1.0 :  None : False : False
  y : Size=1, Index=None, Domain=Reals
      Key  : Lower : Value : Upper : Fixed : Stale
      None :  None :   1.0 :  None : False : False
…
```

- How do I generate nicely formatted output?
- What if I want to solve this problem repeatedly with different initialization?
- What if I have data processing to do before hand?
- How can I use the power of Python to build optimization solutions?
- Write a Python script instead of using the “pyomo” command

Scripting brings the power of Python to Pyomo

Example: Scripting (Rosenbrock)

```
# rosenbrock_script.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)

solver = SolverFactory('ipopt')
solver.solve(model, tee=True)

print()
print('*** Solution *** :')
print('x:', value(model.x))
print('y:', value(model.y))
```

Example: Scripting (Rosenbrock)

```
# rosenbrock_script.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)

solver = SolverFactory('ipopt')
solver.solve(model, tee=True)

print()
print('*** Solution *** :')
print('x:', value(model.x))
print('y:', value(model.y))
```

Example: Scripting (Rosenbrock)

```python
# rosenbrock_script.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)

solver = SolverFactory('ipopt')
solver.solve(model, tee=True)

print()
print('*** Solution *** :')
print('x:', value(model.x))
print('y:', value(model.y))
```

Example: Scripting (Rosenbrock)

```
# rosenbrock_script.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)

solver = SolverFactory('ipopt')
solver.solve(model, tee=True)
```

```
print()
print('*** Solution *** :')
print('x:', value(model.x))
print('y:', value(model.y))
```

Example: Scripting (Rosenbrock)

```
# rosenbrock_script.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)

solver = SolverFactory('ipopt')
solver.solve(model, tee=True)

print()
print('*** Solution *** :')
print('x:', value(model.x))
print('y:', value(model.y))
```

```
python rosenbrock_script.py
```

Exercise: Scripting (looping over initial value)

- Modify rosenbrock_script.py to solve the rosenbrock problem for different initial values and a table of output that shows the initial values and the solution for both x and y. (I.e., complete the following table)

x_init	y_init	x_soln	y_soln
2.00	5.00	----	----
3.00	5.00	----	----
4.00	5.00	----	----
5.00	5.00	----	----

Example: Scripting (loop over initial value)

```
# rosenbrock_script_loop.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)


print('x_init, y_init, x_soln, y_soln')
y_init = 5.0
for x_init in range(2, 6):
    model.x = x_init
    model.y = 5.0

    solver = SolverFactory('ipopt')
    solver.solve(model)

    print("{0:6.2f}  {1:6.2f}  {2:6.2f}  {3:6.2f}".format(x_init, \
          y_init, value(model.x), value(model.y)))
```

Introduction to IPOPT

$$\min_{x} \quad f(x) \longleftarrow \text{Objective Function}$$

$$\text{s.t.} \quad c(x) = 0 \longleftarrow \text{Equality Constraints}$$

$$d^L \leq d(x) \leq d^U \longleftarrow \text{Inequality Constraints}$$

$$x^L \leq x \leq x^U \longleftarrow \text{Variable Bounds}$$

$$x \in \Re^n$$

$$f(x) : \Re^n \mapsto \Re$$

$$c(x) : \Re^n \mapsto \Re^m$$

$$d(x) : \Re^n \mapsto \Re^p$$

- f(x), c(x), d(x)
  - general nonlinear functions (non-convex?)
  - Smooth (C^2)
- The x variables are continuous
  - x(x-1)=0 for discrete conditions really doesn't work

Introduction to IPOPT

$$\min_{x} \quad f(x)$$ ← Cost/Profit, Measure of fit

$$\text{s.t.} \quad c(x) = 0$$ ← Physics of the system

$$d^L \leq d(x) \leq d^U$$ ←

$$x^L \leq x \leq x^U$$ ←

Physical, Performance, Safety Constraints

$$x \in \Re^n$$

$$f(x) : \Re^n \mapsto \Re$$

$$c(x) : \Re^n \mapsto \Re^m$$

$$d(x) : \Re^n \mapsto \Re^p$$

- f(x), c(x), d(x)
  - general nonlinear functions (non-convex?)
  - Smooth (C^2)
- The x variables are continuous
  - x(x-1)=0 for discrete conditions really doesn't work

Large Scale Optimization

- Gradient Based Solution Techniques

$$
\begin{aligned}
\min_x \quad & f(x) \\
\text{s.t.} \quad & c(x) = 0 \\
& x \ge 0
\end{aligned}
\quad \Longrightarrow \quad
\begin{aligned}
\nabla f(x) + \nabla c(x)^T \cdot \lambda - z &= 0 \\
c(x) &= 0 \\
X \cdot z &= 0 \\
(x \ge 0, z &\ge 0)
\end{aligned}
$$

Newton Step

$$
\begin{bmatrix} W_k & \nabla c(x_k) \\ \nabla c(x_k)^T & 0 \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta \lambda \end{pmatrix} = - \begin{bmatrix} \nabla f(x_k) + \nabla c(x_k)^T \lambda_k \\ c(x_k) \end{bmatrix}
$$

$$
\left(W_k = \nabla^2_{xx} \mathcal{L} = \nabla^2_{xx} f(x_k) + \nabla^2_{xx} c(x_k) \lambda \right)
$$

Active-set Strategy

Interior Point Methods

Original NLP

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & x \geq 0 \end{aligned}$$

Barrier NLP

$$\begin{aligned} \min_x \quad & f(x) - \mu \cdot \sum_i ln(x_i) \\ \text{s.t.} \quad & c(x) = 0 \end{aligned}$$

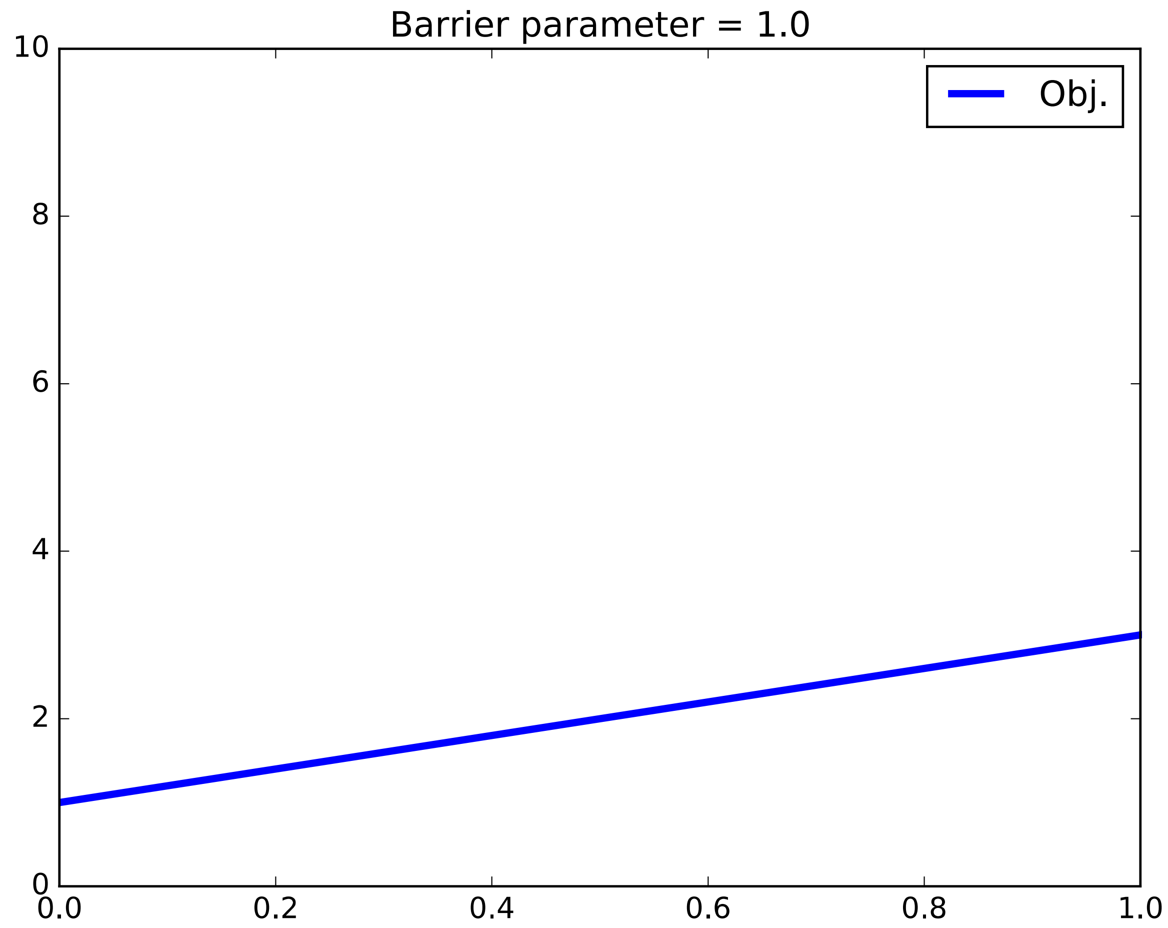
$$\text{as} \quad x \rightarrow 0, \quad ln(x) \rightarrow \infty$$

$$\text{as} \quad \mu \rightarrow 0, \quad x^\star(\mu) \rightarrow x^\star$$

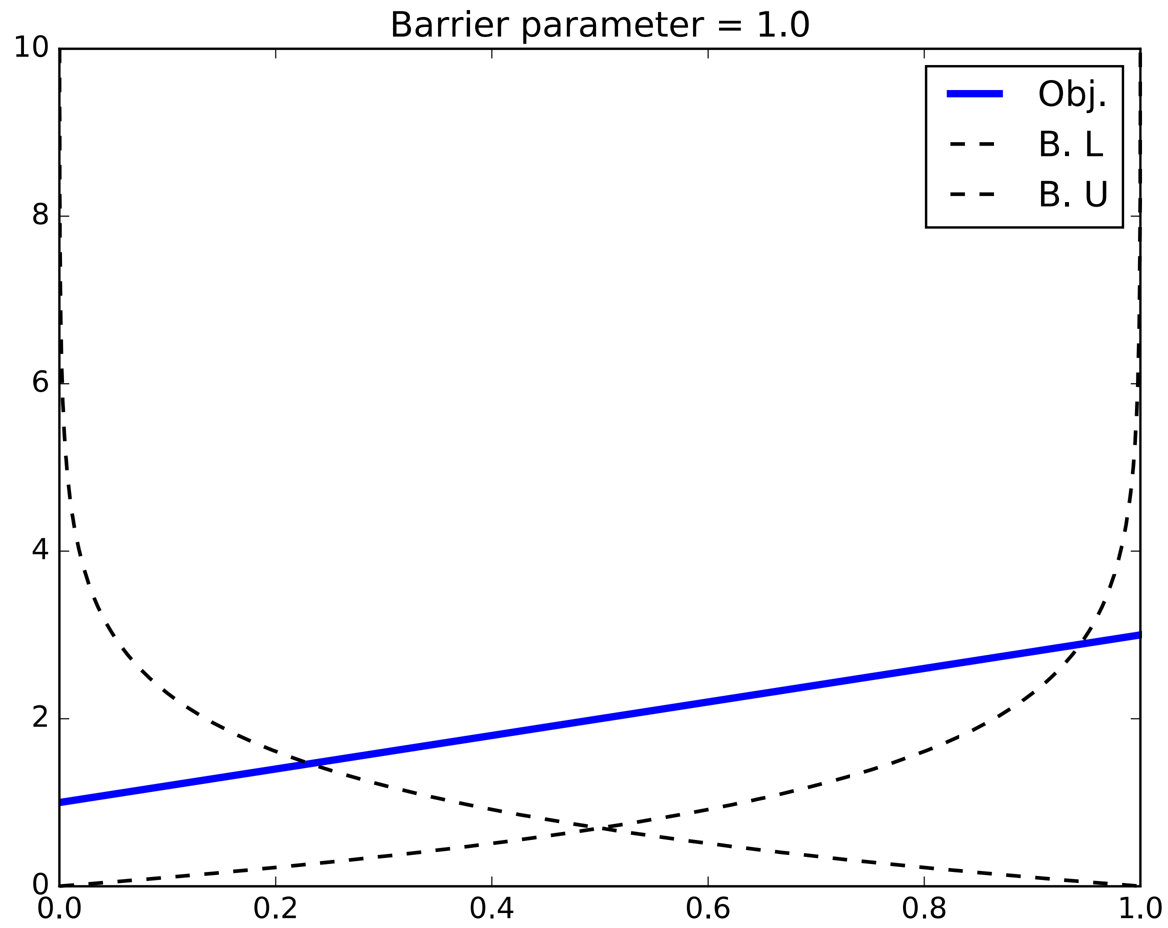
Fiacco & McCormick (1968)

- Initialize
  $$x_0 > 0, \quad \mu_0 > 0, \quad \text{Set } l \leftarrow 0$$
- Solve Barrier NLP for μ_l
- Decrease the barrier parameter
  $$\mu_{l+1} < \mu_l$$
- Increase $l \leftarrow l + 1$

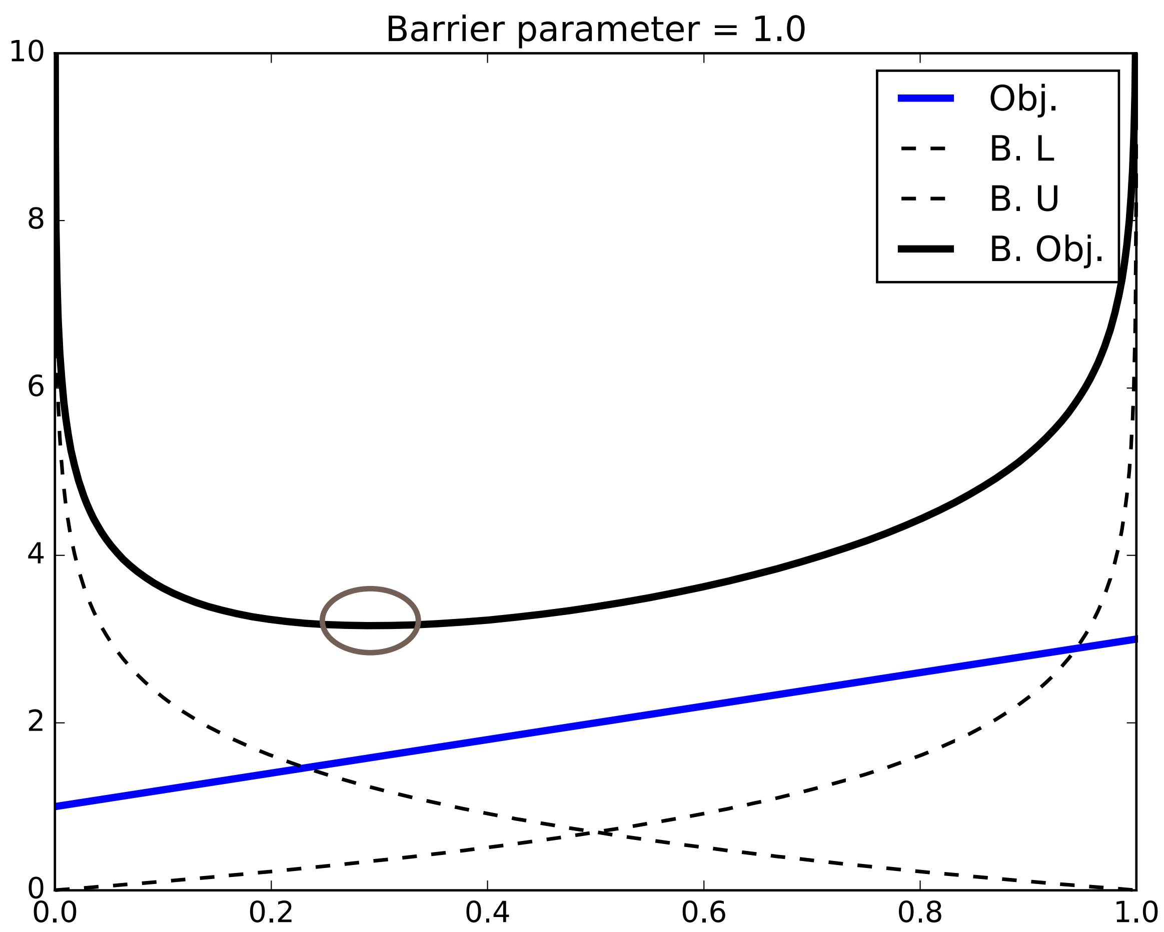
Effect of Barrier Term



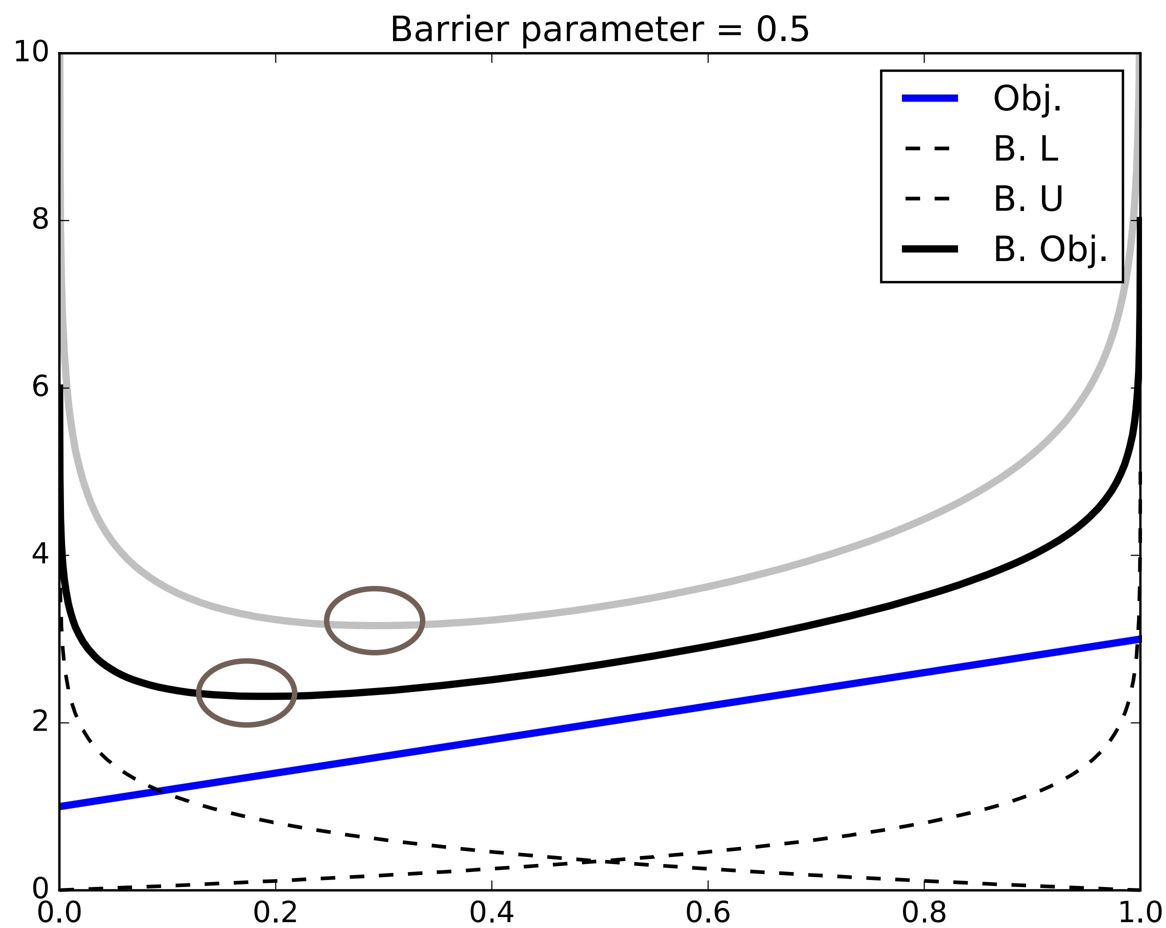
Effect of Barrier Term



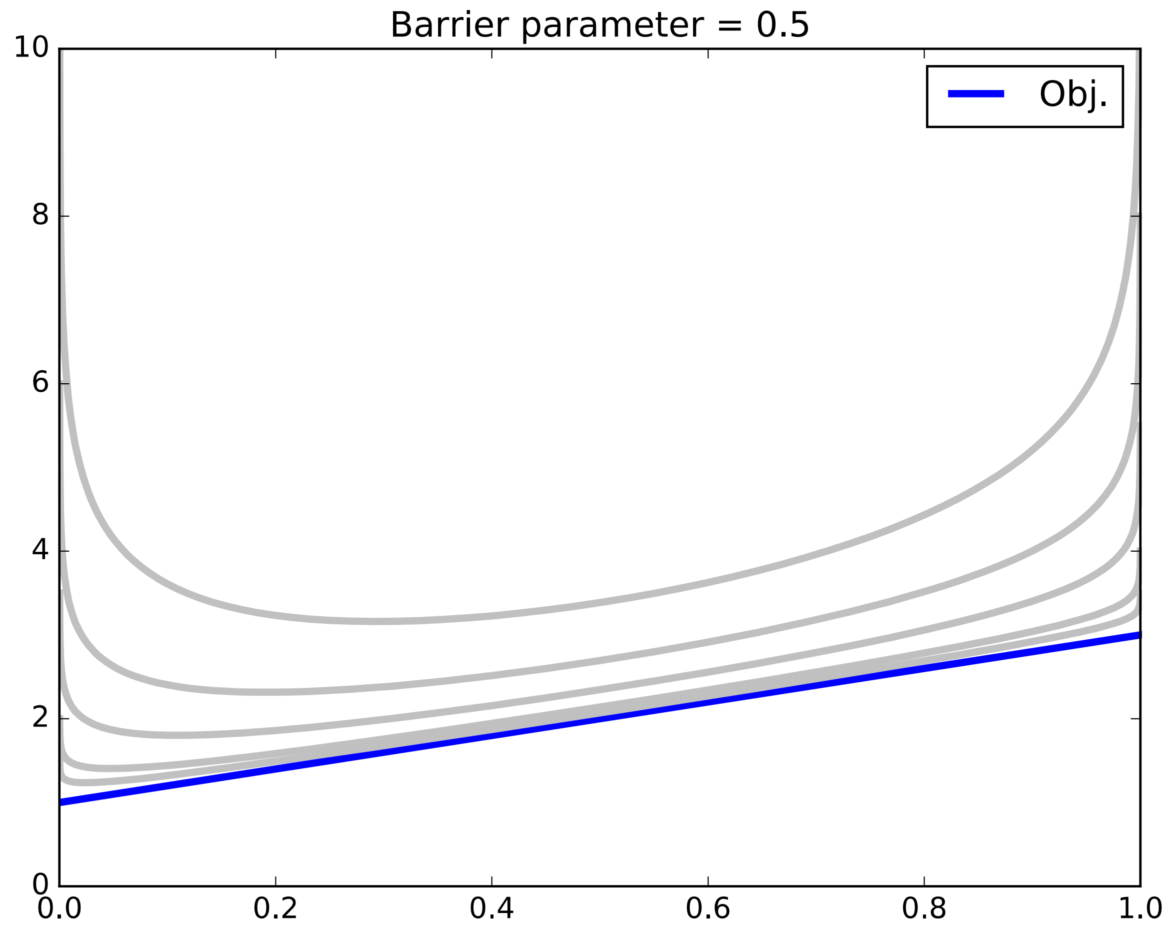
Effect of Barrier Term



Effect of Barrier Term



Effect of Barrier Term



Interior Point Methods

Original NLP

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & x \geq 0 \end{aligned}$$

Barrier NLP

$$\begin{aligned} \min_x \quad & \varphi_\mu(x) = f(x) - \mu \cdot \sum_i ln(x_i) \\ \text{s.t.} \quad & c(x) = 0 \end{aligned}$$

$$\text{as} \quad x \rightarrow 0, \quad ln(x) \rightarrow \infty$$

$$\text{as} \quad \mu \rightarrow 0, \quad x^\star(\mu) \rightarrow x^\star$$

Fiacco & McCormick (1968)

- Initialize
  $x_0 > 0, \quad \mu_0 > 0, \quad \text{Set } l \leftarrow 0$
- Solve Barrier NLP for μ_l
- Decrease the barrier parameter
  $\mu_{l+1} < \mu_l$
- Increase $l \leftarrow l + 1$

Solve Barrier NLP?
Barrier parameter update?
Globalization?

- KNITRO (Byrd, Nocedal, Hribar, Waltz)
- LOQO (Benson, Vanderbei, Shanno
- IPOPT (Waechter, Biegler)

Interior Point Methods

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t} \quad & c(x)=0 \\ & x \geq 0 \end{aligned}$$

$$\begin{aligned} \min_x \quad & f(x) - \mu \sum_i \ln(x_i) \\ \text{s.t} \quad & c(x)=0 \end{aligned}$$

$$\begin{aligned} \nabla f(x) + \nabla c(x)^T \lambda - z &= 0 \\ c(x) &= 0 \\ Xz &= \mu e \\ (x > 0, z &> 0) \end{aligned}$$

$$z=\mu X^{-1} e$$

$$\begin{aligned} \nabla f(x) + \nabla c(x)^T \lambda - \mu X^{-1} e &= 0 \\ c(x) &= 0 \\ (x &> 0) \end{aligned}$$

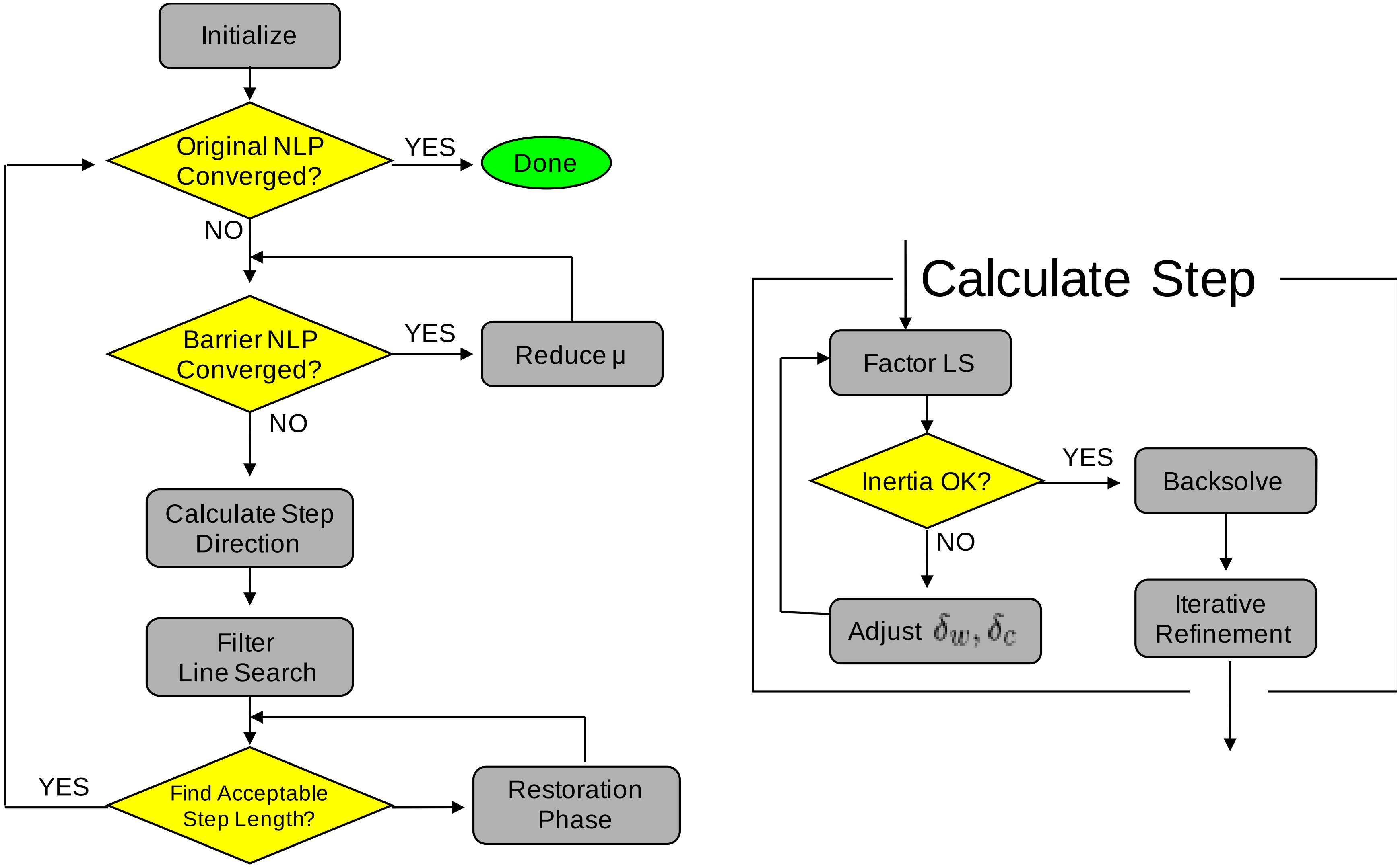
$$\begin{bmatrix} W_k + \Sigma_k + \delta_w I & \nabla c(x_k) \\ \nabla c(x_k)^T & -\delta_c I \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = - \begin{bmatrix} \nabla \psi_\mu (x_k) + \nabla c(x_k)^T \lambda_k \\ c(x_k) \end{bmatrix}$$

$$(W_k = \nabla^2_{xx} \mathcal{L}, \ \Sigma_k = Z_k X_k^{-1})$$

IPOPT: Other Considerations

- Regularization:
  - If certain convexity criteria are not satisfied at a current point, IPOPT may need to regularize. (This can be seen in the output.)
  - We do NOT want to see regularization at the final iteration (solution).
  - Can be an indicator of poor conditioning.
- Globalization:
  - IPOPT uses a filter-based line-search approach
  - Accepts the step if sufficient reduction is seen in objective or constraint violation
- Restoration Phase:
  - Minimize constraint violation
  - Regularized with distance from current point
  - Similar structure to original problem (reuse symbolic factorization)

IPOPT Algorithm Flowsheet



IPOPT Output

```
******************************************************************************
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 3.11.7, running with linear solver ma27.

Number of nonzeros in equality constraint Jacobian...:        2
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:        1

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

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  5.0000000e-01 2.50e-01 5.00e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
```

IPOPT Output

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  5.0000000e-01 2.50e-01 5.00e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  2.4298076e-01 2.33e-01 7.67e-01  -1.0 5.20e-01    -  7.73e-01 9.52e-01h  1
   2  2.6898113e-02 7.23e-05 4.09e-04  -1.7 2.16e-01    -  1.00e+00 1.00e+00h  1
   3  1.8655807e-04 1.83e-04 7.86e-05  -3.8 2.68e-02    -  1.00e+00 9.97e-01f  1
   4  1.8250072e-06 1.23e-12 2.22e-16  -5.7 1.85e-04    -  1.00e+00 1.00e+00h  1
   5 -1.7494097e-08 8.48e-13 0.00e+00  -8.6 1.84e-06    -  1.00e+00 1.00e+00h  1

Number of Iterations....: 5

                                   (scaled)                 (unscaled)
Objective...............:  -1.7494096510394117e-08   -1.7494096510394117e-08
Dual infeasibility......:   0.0000000000000000e+00    0.0000000000000000e+00
Constraint violation....:   8.4843243541854463e-13    8.4843243541854463e-13
Complementarity.........:   2.5050549017950606e-09    2.5050549017950606e-09
Overall NLP error.......:   2.5050549017950606e-09    2.5050549017950606e-09
```

- iter: iterations (codes)
- objective: objective
- Inf_pr: primal infeasibility (constraints satisfied? current constraint violation)
- Inf_du: dual infeasibility (am I optimal?)
- lg(mu): log of the barrier parameter, mu
- ||d||: length of the current stepsize
- lg(rg): log of the regularization parameter
- alpha_du: stepsize for dual variables
- alpha_pr: stepsize for primal variables
- ls: number of line-search steps

Exit Conditions

- Successful Exit
- Successful Exit with regularization at solution
- Infeasible
- Unbounded

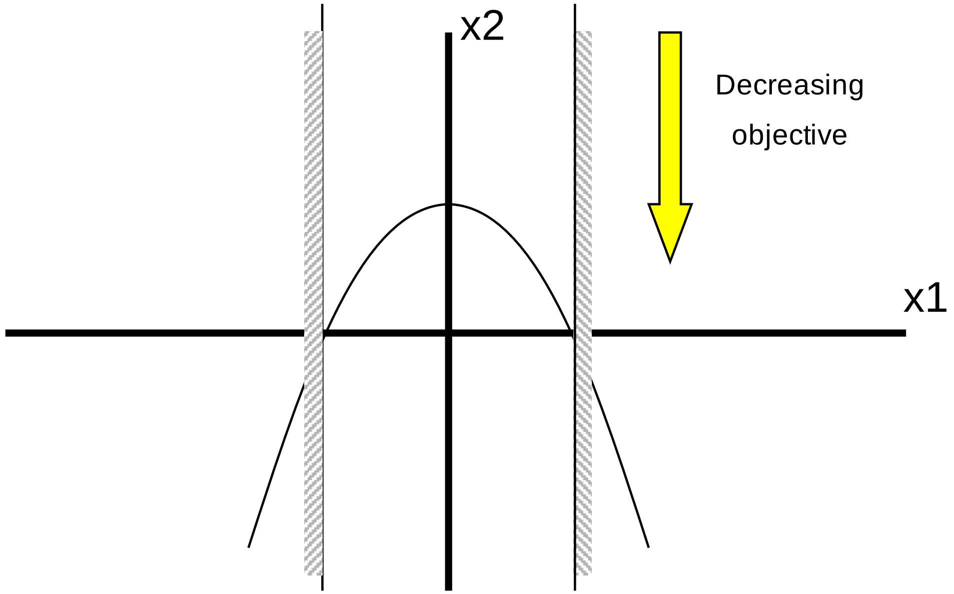
Exit Conditions: Successful

$$\min \quad x_2$$

$$-x_2 = x_1^2 - 1$$

$$-1 \le x_1 \le 1$$

Initialize at (x1=0.5, x2=0.5)



Exit Conditions: Successful

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  5.0000000e-01 2.50e-01 5.00e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  2.4298076e-01 2.33e-01 7.67e-01  -1.0 5.20e-01    -  7.73e-01 9.52e-01h  1
   2  2.6898113e-02 7.23e-05 4.09e-04  -1.7 2.16e-01    -  1.00e+00 1.00e+00h  1
   3  1.8655807e-04 1.83e-04 7.86e-05  -3.8 2.68e-02    -  1.00e+00 9.97e-01f  1
   4  1.8250072e-06 1.23e-12 2.22e-16  -5.7 1.85e-04    -  1.00e+00 1.00e+00h  1
   5 -1.7494097e-08 8.48e-13 0.00e+00  -8.6 1.84e-06    -  1.00e+00 1.00e+00h  1

Number of Iterations....: 5

                                   (scaled)                 (unscaled)
Objective...............:  -1.7494096510367012e-08   -1.7494096510367012e-08
Dual infeasibility......:   0.0000000000000000e+00    0.0000000000000000e+00
Constraint violation....:   8.4843243541854463e-13    8.4843243541854463e-13
Complementarity.........:   2.5050549017950606e-09    2.5050549017950606e-09
Overall NLP error.......:   2.5050549017950606e-09    2.5050549017950606e-09

. . .

EXIT: Optimal Solution Found.

Ipopt 3.11.1: Optimal Solution Found

*** soln
x1 = 1.0
x2 = -1.7494096510367012e-08
```

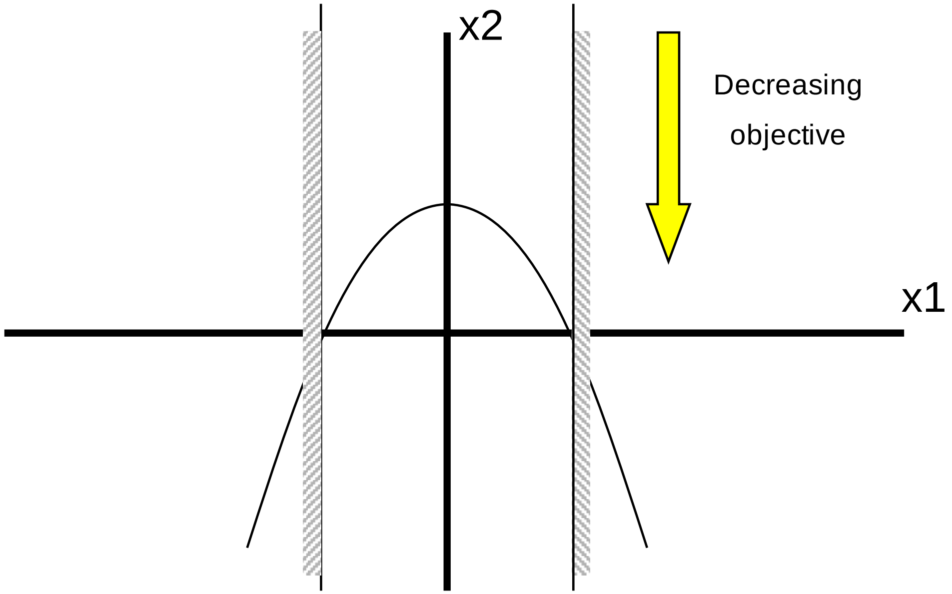
Exit Conditions: Successful w/ Regularization

$$\min \quad x_2$$

$$-x_2 = x_1^2 - 1$$

$$-1 \leq x_1 \leq 1$$

Initialize at (x1=0.0, x2=2.0)

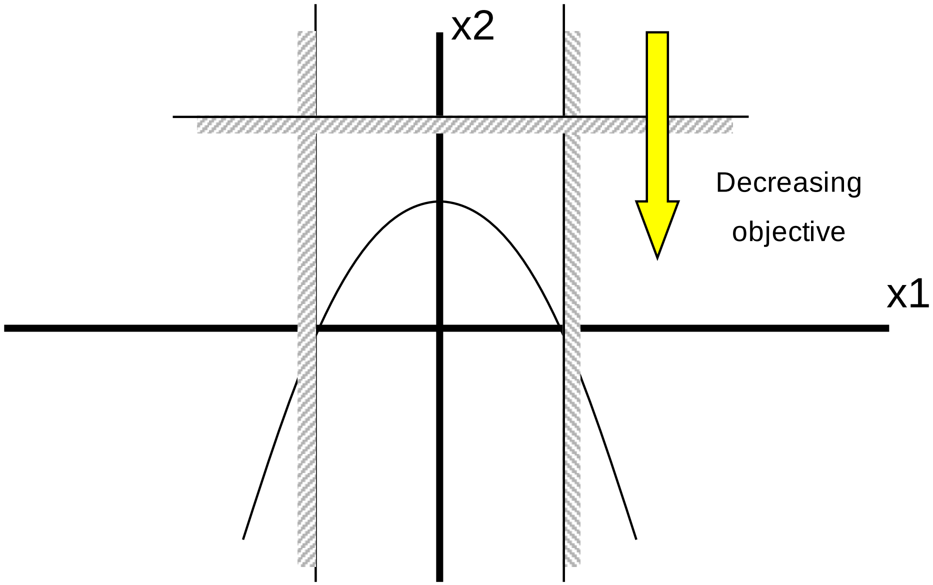


Exit Conditions: Successful w/ Regularization

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  2.0000000e+00 1.00e+00 0.00e+00  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  1.0000000e+00 0.00e+00 1.00e-04  -1.7 1.00e+00  -4.0 1.00e+00 1.00e+00h  1
   2  1.0000000e+00 0.00e+00 0.00e+00  -3.8 0.00e+00   0.9 1.00e+00 1.00e+00   0
   3  1.0000000e+00 0.00e+00 0.00e+00  -5.7 0.00e+00   0.5 1.00e+00 1.00e+00T  0
   4  1.0000000e+00 0.00e+00 0.00e+00  -8.6 0.00e+00   0.9 1.00e+00 1.00e+00T  0

Number of Iterations....: 4

                                   (scaled)                 (unscaled)
Objective...............:   1.0000000000000000e+00    1.0000000000000000e+00
Dual infeasibility......:   0.0000000000000000e+00    0.0000000000000000e+00
Constraint violation....:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.........:   2.5059035596800808e-09    2.5059035596800808e-09
Overall NLP error.......:   2.5059035596800808e-09    2.5059035596800808e-09

. . .

EXIT: Optimal Solution Found.

Ipopt 3.11.1: Optimal Solution Found

*** soln
x1 = 0.0
x2 = 1.0
```

Exit Conditions: Infeasible

$$\min \quad -x_2$$
$$\text{s.t.} \quad -x_2 = x_1^2 - 1$$
$$-1 \le x_1 \le 1$$
$$x_2 \ge 2$$



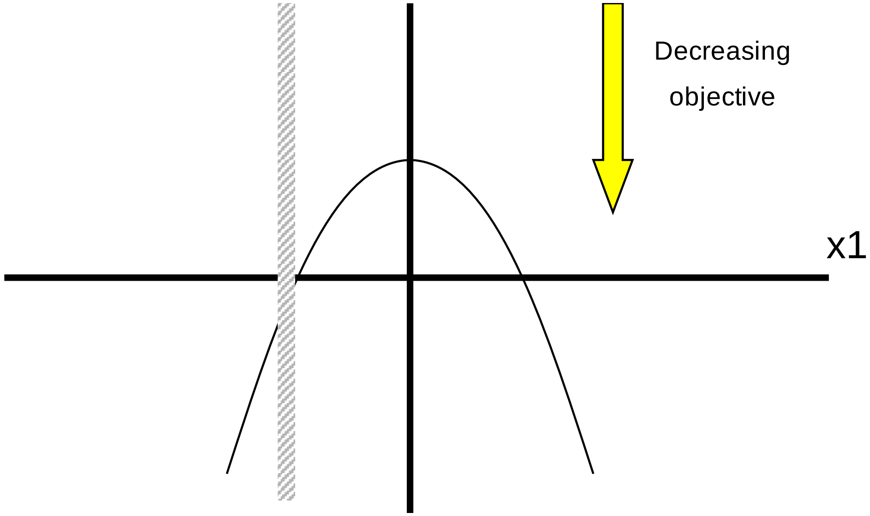
Exit Conditions: Infeasible

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
. . .
   5  2.0000016e+00 1.00e+00 4.74e+03  -1.0 3.47e+01    -  2.79e-02 4.16e-04h  7
   6r 2.0000016e+00 1.00e+00 1.00e+03   0.0 0.00e+00    -  0.00e+00 4.44e-07R  3
   7r 2.0010000e+00 1.01e+00 1.74e+02   0.0 8.73e-02    -  1.00e+00 1.00e+00f  1
   8r 2.0010010e+00 1.00e+00 1.32e-03   0.0 8.73e-02    -  1.00e+00 1.00e+00f  1
   9r 2.0000080e+00 1.00e+00 5.18e-03  -2.1 6.12e-03    -  9.94e-01 9.99e-01h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  10r 2.0000000e+00 1.00e+00 3.73e-06  -4.7 7.99e-06    -  1.00e+00 1.00e+00f  1
  11r 2.0000000e+00 1.00e+00 1.79e-07  -7.1 2.85e-08    -  1.00e+00 1.00e+00f  1

Number of Iterations....: 11

                                   (scaled)                 (unscaled)
Objective...............:   1.9999999800009090e+00    1.9999999800009090e+00
Dual infeasibility......:   1.0000000002321485e+00    1.0000000002321485e+00
Constraint violation....:   9.9999998000090895e-01    9.9999998000090895e-01
Complementarity.........:   9.0909091652062654e-10    9.0909091652062654e-10
Overall NLP error.......:   9.9999998000090895e-01    1.0000000002321485e+00

. . .

EXIT: Converged to a point of local infeasibility. Problem may be infeasible.

Ipopt 3.11.1: Converged to a locally infeasible point. Problem may be infeasible.
WARNING - Loading a SolverResults object with a warning status into model=unknown; message from solver=Ipopt 3.11.1\x3a Converged to a locally infeasible point. Problem may be infeasible.

*** soln
x1 = -6.353194883662875e-12
x2 = 2.0
```

Exit Conditions: Unbounded

$$\min \quad -x_2$$
$$\text{s.t.} \quad -x_2 = x_1^2 - 1$$
$$-1 \le x_1$$

Initialize at (x1=0.5, x2=0.5)



Exit Conditions: Unbounded

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
. . .
  45 -2.2420218e+11 1.00e+04 1.00e+00  -1.7 1.25e+19 -19.1 3.55e-08 7.11e-15f 48
  46 -2.2420225e+11 1.00e+04 1.00e+00  -1.7 3.75e+19 -19.6 1.20e-08 1.78e-15f 50
  47 -2.2420229e+11 1.00e+04 1.00e+00  -1.7 1.25e+19 -19.1 1.00e+00 3.55e-15f 49
  48 -3.7503956e+19 1.57e+27 8.36e+09  -1.7 3.75e+19 -19.6 1.18e-08 1.00e+00w  1
  49 -1.3750923e+20 3.92e+26 2.09e+09  -1.7 1.00e+20 -20.0 1.00e+00 1.00e+00w  1

Number of Iterations....: 49

                                   (scaled)                 (unscaled)
Objective...............:  -1.3750923074037683e+20   -1.3750923074037683e+20
Dual infeasibility......:   2.0888873315629249e+09    2.0888873315629249e+09
Constraint violation....:   3.9209747283936173e+26    3.9209747283936173e+26
Complementarity.........:   3.1115099971882619e+03    3.1115099971882619e+03
Overall NLP error.......:   3.9209747283936173e+26    3.9209747283936173e+26

EXIT: Iterates diverging; problem might be unbounded.

Ipopt 3.11.1: Iterates diverging; problem might be unbounded.
WARNING - Loading a SolverResults object with a warning status into model=unknown; message from solver=Ipopt
3.11.1\x3a Iterates diverging; problem might be unbounded.

*** soln
x1 = 0.5
x2 = 0.5
```

IPOPT Options

- Solver options can be set through scripts (and the pyomo command line)
- print_options_documentation yes
  - Outputs the complete set of IPOPT options (with documentation and their defaults)
- mu_init
  - Sets the initial value of the barrier parameter
  - Can be helpful to make this smaller when initial guesses are known to be good
- bounds_push
  - By default, IPOPT pushes the bounds a little further out.
  - This can be set to remove this behavior
  - E.g., sqrt(x), x>= 0
- linear_solver
  - Set the linear solver that will be used for the KKT system
  - Significantly better performance with HSL (MA27) over default MUMPS
- print_user_options
  - Print options set and whether or not they were used
  - Helpful to detect mismatched options

IPOPT Options

```
# rosenbrock_options.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)

solver = SolverFactory('ipopt')
solver.options['mu_init'] = 1e-4
solver.options['print_user_options'] = 'yes'
solver.options['ma27_pivtol'] = 1e-4
solver.solve(model, tee=True)

print()
print('*** Solution *** :')
print('x:', value(model.x))
print('y:', value(model.y))
```

IPOPT Options

```
ma27_pivtol=0.0001
print_user_options=yes
mu_init=0.0001
ma27_pivtol=0.0001
print_user_options=yes
mu_init=0.0001


List of user-set options:

                                    Name   Value                used
                        ma27_pivtol = 0.0001                     no
                            mu_init = 0.0001                    yes
                 print_user_options = yes                       yes

******************************************************************************
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
******************************************************************************

NOTE: You are using Ipopt by default with the MUMPS linear solver.
      Other linear solvers might be more efficient (see Ipopt documentation).


This is Ipopt version 3.11.1, running with linear solver mumps.
```

Modeling Tips

- Variable Initialization
  - Proper initialization of nonlinear problems can be critical for effective solution.
  - Strategies include:
    - Using understood physics or past successful solutions
    - Solving simpler problem(s) first, progressing to more difficult
- Undefined Evaluations
  - Many mathematical functions have a valid domain, and evaluation outside that domain causes errors
  - Add appropriate bounds to variables to keep them inside valid domain
  - Note that solvers use first and second derivatives. While sqrt(x) is valid at x=0, 1/sqrt(x) is not
- Problem Scaling
  - Scale model to avoid variables, constraints, derivatives with different scales.
- Formulation Matters!